International Journal of Computational Intelligence and Informatics, Vol. 6: No. 2, September 2016

An Experimental Investigation on Combinatorial Testing with Proficient Constraints

M. Bharathi
Assistant Professor
Department of Computer Science
Periyar University College
Pennagaram, TamilNadu
e-mail : abinovharshiniset@gmail.com

V. Sangeetha
Assistant Professor
Department of Computer Science
Periyar University College
Pappireddipatty, TamilNadu.
e-mail: sangee759@gmail.com

***Abstract*- Combinatorial testing is concentrated on pinpointing errors and faults arise owing to the interaction of altered parameters of software. On every occasion our processes have manifold variables that may intermingle with each other, we use combinatorial analysis. The variables may derive from diversity of sources such as dissimilar operating system, peripherals, and altered databases or from a network. In combinatorial testing, the assignment is to attest that distinct combinations of variables are controlled correctly by the system. Combinatorial methods aid to decrease the cost and increase the efficiency of software testing in numerous applications. Combinatorial testing delivers an enhanced way to shield all the thinkable combinations with improved swapping between cost and time. Pairwise testing is a most familiar test planning technique. Due to resource constraints, it is nearly always impossible to exhaustively test all of these combinations of parameter values. In this paper we propose an approach to generate test cases for pairwise testing by applying and prioritizing input parameter and their corresponding values to be selected and also providing solution for the constraint handling between parameters and values and removing invalid test cases and evaluate the results of 100% coverage with specified constraints.**

Keywords - combinatorial testing, pairwise testing, constraint handling, test case generation

I. Introduction

A. Generation of Test cases

A test engineer uses set of variables or conditions to discover either a system under persuade the test for functioning properly. These test cases are helpful to find the requirements in the software design. Formal or informal tests conducted in the requirements phase are called the beginning point of generation of tests. It plays a vital role to categorize the input domain to be developed. Formal or informal requirements are there. There is lot of requirement technique presented in the black box strategy [1].

B. Testing methods

Test cases are developed via a variety of test techniques to accomplish more effective testing. Software can be tested in two ways as in the figure 1. They are:

(i) White box testing and
(ii) Black box testing

White box testing is a testing strategy with the information of internal structure and coding inside the program. It is considered as a security testing. Black box testing is a testing strategy based on output requirements and without knowledge of internal structure and coding inside the program. Depending on how tests are generated, a technique could be called as a black box or white box technique. Black box testing is divided into three types

(i) Usage based
(ii) Fault based

1. Random testing

In random testing, test cases are preferred randomly from a quantity of uniform distribution without exploiting information from the specification or previously chosen test cases.

2. *Partition testing*

The input space of the test object may be separated into subsets depend on that all points in the identical subset result in a related behavior from test object. Equivalence partitioning, category partitioning and domain testing are belongs to partition testing.

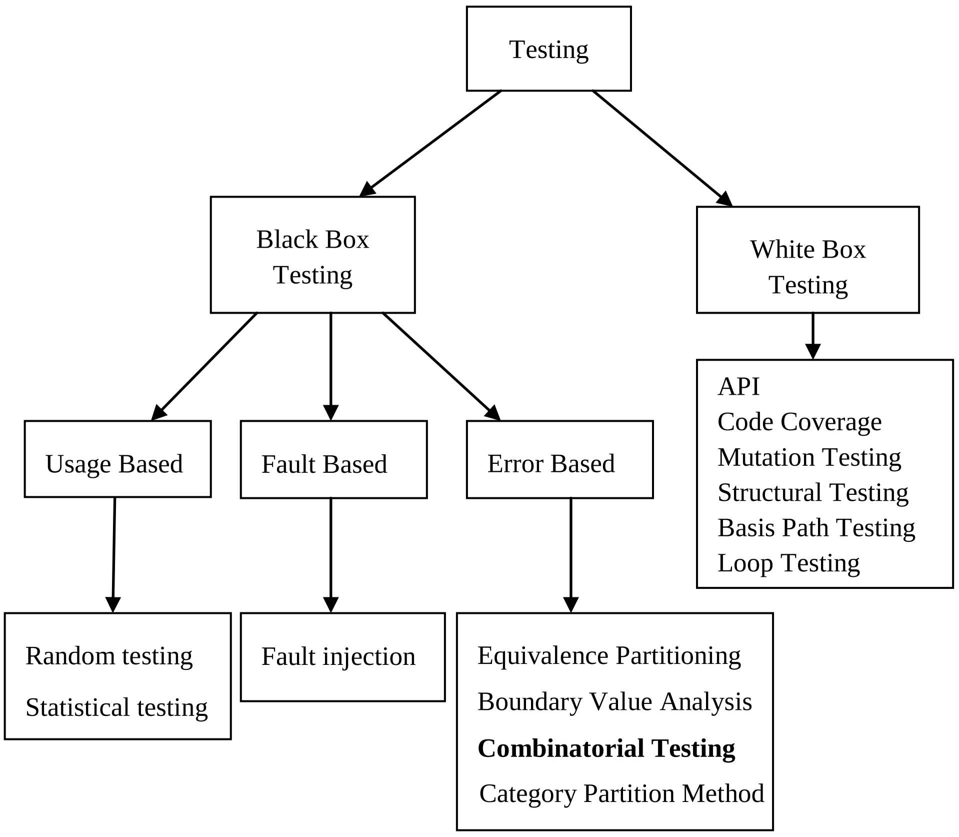


Figure 1. Classification of Testing

3. *Equivalence Partitioning (EP)*

The input domain data is separated into equivalence data classes in equivalence partitioning testing technique. The utmost requirements are sheltered to reduce the total number of test cases in the method. For instance, if you have to test for an input box containing numbers from 1 to 1000 then there is no need to write 1000 test cases for all 1000 valid input numbers plus other test cases for invalid data. This test case can be divided into three sets of input data called classes. Pick a single value range from 1 to 1000 as a valid test case. If you select other values between 1 and 1000 then result is same. If you select any value below 1 or above 1000, as an invalid input data test case. Equivalence partitioning uses smallest amount test cases to cover maximum requirements.

4. *Boundary Value Analysis (BVA)*

Boundary vale analysis technique is used to discover errors at boundaries instead of finding in centre of input domain. For example, test cases with test data accurately as the input boundaries of input domain (values 1 and 1000).Test data with values below the edge of input domain (values 0 and 999). Test data with values above the edge of input domain (values 2 and 1001). Boundary value analysis is called as a part of stress and negative testing.

5. *Category Partition Method (CPM)*

Category partition method is a specification-based testing developed by Ostrand and Balcer. It helps to produce test cases by refining the functional specification of a program into test specifications. This method consists of the following steps:

1. Decompose the functional specification into functional units that can be tested separately.
2. Identify the explicit input parameter and environment conditions.
3. Find categories (major properties or characteristics) of information that characterize each parameter and environment condition.
4. Partition each category into choices which include all the different kind of values that are possible for that category.
5. Determine the constraints among the choices of different categories.
6. Write the test specification which is a list of categories, choices and constraints in a predefined format using the test specification language.
7. Use a generator to produce test frames from the test specification.
8. For each generated test frame, create a test case by selecting a single element from each choice in that test frame.

Table I. Comparative study of random and partitioning testing

Researcher	Investigation Result
Duran and Ntafos *et al.,*	Based on certain conditions, random testing can be as effective as partition testing
Hamlet and Taylor *et al.,*	Duran and Ntfos model was unrealistic because the overall failure probability was too high
Gutjahr *et al.,*	Partition testing is more effective than random testing under realistic model.
Reid and yin, Lebne-Dengel, and Malayia *et al.,*	Random testing was less effective than partition testing methods.

The paper is structured as follows: Section II describes the background of combinatorial testing. Section III describes combinatorial testing techniques. Section IV describes our proposed approach of generation of test cases with constraints and removes invalid test cases and executes the test cases. Finally, Section V presents conclusion and a sketch for future work.

II. Back Ground

A. *Combinatorial testing*

Many software systems are highly configurable. It is usually infeasible to test all the possible configurations. To overcome this problem, Combinatorial Testing (CT) Techniques (Figure 2) have been developed specifically for such systems. Combinatorial methods aid to decrease the cost and increase the efficiency of software testing in numerous applications as in table II. CT is more efficient and effective compare with random testing. Also CT is an effective failure detection method for many types of software systems. CT which generates test groups by selecting values for each individual input parameters and by combining these parameter values. For a system with p parameters, each of which has v values, the number of all possible combinations of values of these parameters is v^p [2]. Pairwise testing is a most familiar test planning technique. But investigations of actual failures in a number of software and systems show that pairwise testing may not be sufficient so high strength CT (t-way for t>2) may be desired. Due to resource constraints, it is nearly always impossible to exhaustively test all of these combinations of parameter values. Thus, a strategy is needed to select a subset of these combinations. One such strategy, called t-way testing, requires every combination of any t parameter values to be covered by at least one test, where t is referred to as the strength of coverage and usually takes a small value. The notation of t-way testing can reduce the number of tests [2] For example, a system of 3 parameters that have 7 values each requires 7^3 tests for exhaustive testing. But it requires 49 for 2-way in pairwise testing. We can consider each combination of parameter values to represent one possible interaction among these parameters. However, testing has constraints on the combination of parameter values. As a result some combinations of parameter values are invalid. In this paper we propose an approach to generate test cases for pairwise testing by applying and prioritizing input parameter and their corresponding values to be selected and also providing solution for the constraint handling between parameters and values.

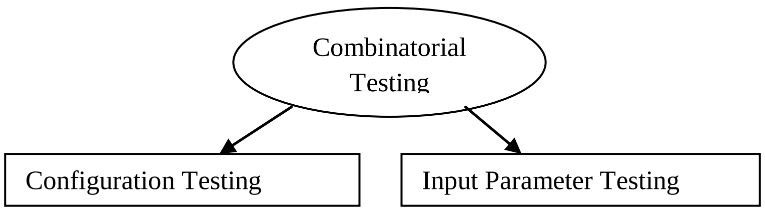


Figure 2. Combinatorial Testing Approaches

1. *Configuration Testing*

The process of testing a system beneath development on machines which have various combinations of hardware and software is Configuration testing. For instance, telecommunications software may be configured to function with dissimilar types of call (local, STD, ISD), billing (caller, phone card, 800), access (ISDN, VOIP, PBX), and hubs for billing (Windows Server, Linux/MySQL, Oracle). With all combinations of these, the software must work properly, so a distinct test suite could be functional to all pairwise combinations of these four major configuration items. Configuration coverage is the most developed form of combinatorial testing.

2. *Input parameter testing*

Some type of input will be processed even though an application has no configuration options. For an instance, there are 10 ways to alter some text to be highlighted in a word processing application: subscript, superscript, underline, bold, italic, emboss, shadow, strikethrough, small caps, or all caps. These modification settings receive as an input by the Font-processing function within the application must process the input and alter the text on the screen suitably. Some of the options are combined like bold and small caps, but some are unable to coexist like subscript and superscript. For the font-processing function work properly for all valid combinations of these input settings requires a systematic testing. There are 2^{10} = 1,024 possible combinations with 10 binary inputs. Base on the empirical analysis reported above shows that failures appear to involve a little number of parameters, and that testing all 3-way combinations may detect 90% or more of bugs. Testing that detects better than 90% of bugs may be a cost-effective choice for a word processing application but we require making sure that all 3way combinations of values are tested. So the test suite is created to cover all 3-way combinations known as a covering array [1].

B. *Test Generation using combinatorial designs*

There are three steps to generate test case using combinatorial designs as in the figure 3.

1. Build a model,
2. Generate a combinatorial design and
3. Generate tests.

If test cases are to be generated then the process starts with a model of the input space. The application environment is modeled when test configurations are to be generated. The modeling of input space or environment is not rip-off.

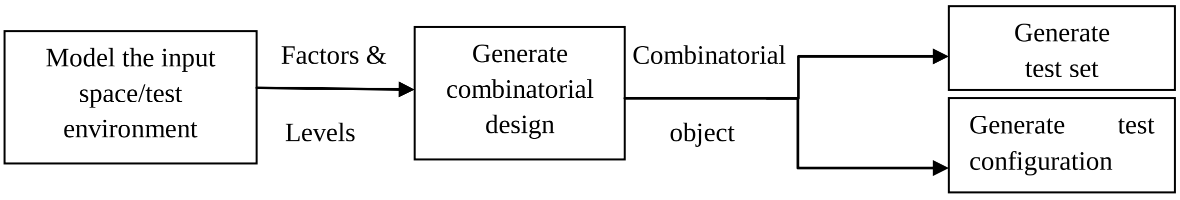

Model the input space/test environment
Factors &
Levels
Generate combinatorial design
Combinatorial
object
Generate test set
Generate test configuration

Figure 3. Generation of Test Cases using Combinatorial Designs

In the second step, the modeling of input to a combinatorial design generates combinatorial object which is an array of factors and levels.

C. *Generation of combinatorial object*

There are several procedures used to generate combinatorial object.

1. *Fault Model :* In fault model, test inputs and configurations issued certain type of faults when certain combination of t>1 input values which causes the program entered into invalid state. A t-way interaction fault occurs due to the interaction of two factors.
2. *Fault Vectors :* A vector consisting of specific values or levels of each factor is known as a run. A run is considered a fault vector if a test derived from fault triggers.

Table II . Applications of combinatorial testing

Year	Name of the tester	Applications	Methodology
2016	Rick Kuhn, Raghu Kacker, Larry Feldman, Greg Witte [3]	Testing for cybersecurity and reliability	combinatorial testing
2016	Rick Kuhn, Raghu Kacker [4]	To build navigation graphs for dynamic web applications	combinatorial testing
2015	Hagar, J. D., Wissink, T. L., Kuhn, D. R., & Kacker, R. N. [5]	Testing with large organizations	combinatorial testing
2013	Hopcroft *et al.,*	To find maximum matching in bipartitie graph	combinatorial testing
2011	Alizadeh *et al.,*	Interior point methods in semi-definite programming	combinatorial testing
2010	S Oster *et al.,*	Software product lines	combinatorial testing
2008	Zhzng *et al.,*	Intelligent information Processing	combinatorial testing
2007	Lei *et al.,* [6]	Concurrent Programs	combinatorial testing
2003	Xu *et al.,*	Configuration testing and browser testing	combinatorial testing
2002	Burr and Young *et al.,*	Testing the email system with AETG	combinatorial testing
2000	White and Almezen *et al.,*	Testing GUI objects.	combinatorial testing
1996	Williams and Probert	Testing network interface	combinatorial testing
1994	Borroughs and Jain *et al.,*	Quality and efficiency of internet protocol testing	combinatorial testing
1993	Huller *et al.,*	Ground system of satellite communication.	combinatorial testing
1992	Brownlie *et al.,*	test case generation	Orthogonal Array Testing Strategy
1992	Brownlie *et al.,*	Weather Forecast Generator (PELMOREX)	pairwise combinatorial coverage
1985	R.Mandl *et al.,*	ADA compiler	combinatorial coverage

3. *Latin squares :* Latin squares are used to generate less factor combinations. A Latin square of order t is t X t matrix of t-symbols. no symbols appears more than once in a row or a column.
4. *Mutually Orthogonal Latin Squares :* Two Latin squares are considered mutually orthogonal if the matrix obtained by juxtaposing the two has unique elements.
5. *Binary factors :* A factor is considered binary if it can assume one of the two possible values. Each factor combination selected will generate at least one test input or test configuration. The process of selecting a subset of combinations from the complete set, suppose a program to be tested requires three inputs, one parallel to each input variable. Each variable can take only one of two distinct values.
6. *Multi-Valued Factors :* Test configurations are constructed from one or more factors where each factor assumes a value from a set containing more than two values.
7. *Orthogonal Arrays :* If each parameter value appears the identical number of times in the test suite then the property of orthogonal array is that they are balanced.
8. *Covering Arrays :* If each parameter value appears at least once but not essentially the identical number of times in the test suite then the property of orthogonal array is that they are not balanced.

9. *Arrays of Strength > 2* : The strength of the array is t=2 in all the preceding techniques. Due to pairwise interactions, these tests are targeted at discovering errors. Sometimes higher strength is required to accomplish superior assurance to ensure accuracy of software [1].

III. Combinatorial Testing Techniques

A. *Pairwise Testing*

Pairwise testing is a combinatorial testing technique which tests all possible combinations of set of input parameters. All pairs of combinations have been combined together at least once during the testing process. Instead of testing each and every combination, all individual pairs of interactions are tested. Investigations show that software defects are triggered by a single input parameters or a combination of two input parameters [8]. To demonstrate the concept of pairwise testing, consider a system with four parameters A, B, C and D. Each of the first three parameters consist two values {SINGLE, DOUBLE}, {B1, B2}, {C1, C2} and the fourth parameter consist of three values {D1, D2, D3} respectively. In order to test all combinations would require 2 X 2 X 2 X 3 = 24 test cases. But a generated pairwise test case set includes 6 test cases covered in all parameter interactions for this circumstance.

B. *Constraint handling*

In testing, some combination of parameters and values are frequently invalid and untestable because they do not exist for the system under test. The pairwise testing technique does not handle the constraints between input parameters and values. We must review the results obtained from pairwise testing and manually delete bad pair test cases themselves [7]. Most existing test generation methods cannot deal with any constraints and finally ignore them. Ignoring constraints generate invalid test sets. When the generated test set contains many invalid test cases, this will cause a loss of combination coverage. Hence, proper constraint handling is a key issue for test set generation. We must specify the constraints before test set generation [5], [8], [9] [10].

IV. Proposed Approach

The proposed technique focuses on generation of test sets without and with constraints and compares the results. The entire process consists of four steps:

A. Select parameters and values
B. Define constraints
C. Generate test cases and
D. Evaluate test cases

A. *Select parameters and values*

The parameters and values are collected from the database or environments. To illustrate the concept of pairwise testing, consider a real life application system with four parameters and their respective values and their symbolic representations are shown in Table 3.

Table III . Parameters and Value representation

Parameters	Values
Vehicle	2wheel , 4wheel
Fuel	petrol , diesel , gas
Accommodation	single , double, suite
Specialization	ac , non-ac

There are two vehicle types, three fuel types, three accommodation options and two specialization options. Different end users may use different combinations of these parameter values.

B. *Define parameters and constraint values*

Some combination of parameter value is unacceptable from Table III which will generate valueless test cases and must be removed from the result of the test set. Table III shows the possible input parameters and their respective values. ACTS tool is used for defining parameters and their values. Generation of test set without constraints will cover valueless test cases. Invalid pair test cases combine 2wheel with ac, or 2wheel with diesel or 2wheel with gas. In Table IV, we give a formal syntax that can be used to specify the constraints for parameters in Table 3.

Table IV: Defined Constraints

Constraint	Description
If(Vehicle=‘2wheel’) then (Fuel=‘Petrol’)	If Vehicle is 2wheeler, then fuel must be Petrol
If(Vehicle=‘2wheel’&&Fuel=‘Petrol’)then (Specialization=‘non-ac’)	If Vehicle is 2wheeler and fuel is Petrol then Specialization must be non-ac
If(Vehicle=‘2wheel’) then (Specialization=‘non-ac’)	If Vehicle is 2wheeler, then Specialization must be non-ac

C. *Generate Test cases*

1. *Generate Exhaustive test cases:*

First we generate a pairwise test case for first two parameters until cover all values and then extend it to another parameter. Repeat the same process until all the parameters and values are covered from Table. In order to test all combinations required 2 x 3 x 3 x 2 = 36 test cases. In this example, exhaustive testing requires 36 test cases. Table V shows all possible combinations 36 exhaustive test cases.

2. *Generate pair-wise test cases :*

But pair-wise combinatorial testing requires only 9 test cases without constraints and 11 test cases with constraints. The entire test suite covers all possible pairwise combinations between system parameters.

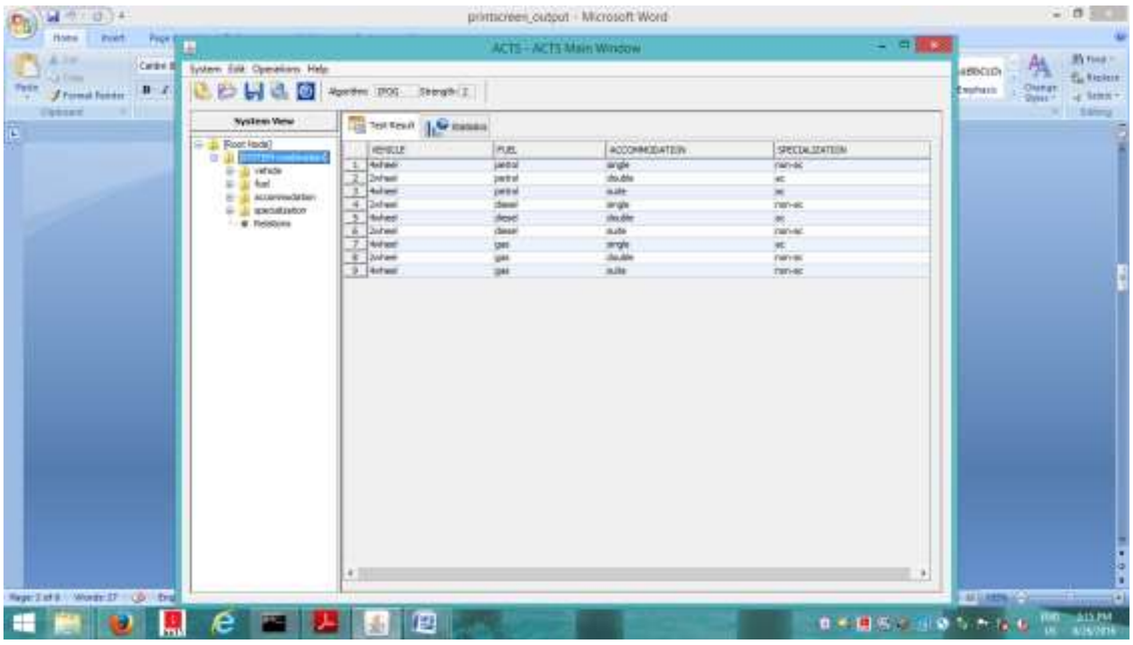
In this situation, we must perform validity check with replace method of constraint handling used to determine whether all the constraints are satisfied by a test case with the specified constraints or not. If invalid test cases are detected, invalid parameter values of that test case should be changed by another valid parameter value and removes the invalid test case. The resulting final test set satisfies all combinations with specified constraints. The ACTS tool implements the IPOG algorithm [2] to generate combinatorial test sets. When we add parameter and their corresponding values and acts tool then automatically generates a t-way test set, where t is the strength of coverage. The generated test cases can be saved in multiple formats for testing purposes. Figure 4 shows 2- way test case generation using ACTS tool.

Table V. All possible combinations

Test case	vehicle	fuel	accommodation	specialization
1	2wheel	petrol	single	Ac
2	2wheel	petrol	single	non-ac
3	2wheel	petrol	double	Ac
4	2wheel	petrol	double	non-ac
5	2wheel	petrol	suite	Ac
6	2wheel	petrol	suite	non-ac
7	2wheel	diesel	single	Ac
8	2wheel	diesel	single	non-ac
9	2wheel	diesel	double	Ac
10	2wheel	diesel	double	non-ac
11	2wheel	diesel	suite	Ac
12	2wheel	diesel	suite	non-ac
13	2wheel	gas	single	Ac
14	2wheel	gas	single	non-ac
15	2wheel	gas	double	Ac
16	2wheel	gas	double	non-ac
17	2wheel	gas	suite	Ac

18	2wheel	gas	suite	non-ac
19	4wheel	petrol	single	Ac
20	4wheel	petrol	single	non-ac
21	4wheel	petrol	double	Ac
22	4wheel	petrol	double	non-ac
23	4wheel	petrol	suite	Ac
24	4wheel	petrol	suite	non-ac
25	4wheel	diesel	single	Ac
26	4wheel	diesel	single	non-ac
27	4wheel	diesel	double	Ac
28	4wheel	diesel	double	non-ac
29	4wheel	diesel	suite	Ac
30	4wheel	diesel	suite	non-ac
31	4wheel	gas	single	Ac
32	4wheel	gas	single	non-ac
33	4wheel	gas	double	Ac
34	4wheel	gas	double	non-ac
35	4wheel	gas	suite	Ac
36	4wheel	gas	suite	non-ac

As the size of the parameter and their value increase, the size of corresponding test sets increase. Managing this combinatorial growth with regard to both accuracy and execution time is still an open research issue. Greedy algorithm is recently used for t-way combinatorial testing [11] [12]. However, the efficient generation of t-way combinatorial test suites remains an ongoing research topic.



Algorithm: IPOG Strength: 2

	VEHICLE	FUEL	ACCOMMODATION	SPECIALIZATION
1	4wheel	petrol	single	non-ac
2	2wheel	petrol	double	ac
3	4wheel	petrol	suite	ac
4	2wheel	diesel	single	non-ac
5	4wheel	diesel	double	ac
6	2wheel	diesel	suite	non-ac
7	4wheel	gas	single	ac
8	2wheel	gas	double	non-ac
9	4wheel	gas	suite	non-ac

Figure 4. Generated 2 way test suites

D. *Evaluate test cases*

To evaluate the future approach, we consider the number of parameters and each has different parameter values and evaluates the test cases and compares the results generated with existing algorithms using existing combinatorial testing tools. A main issue of combinatorial testing is to identify the constraints that exist between certain parameters. But pair-wise combinatorial testing requires only 9 test cases without constraints and 11 test cases with constraints. The entire test suite covers all possible pairwise combinations between system parameters. Due to handling the right constraints between parameters will reduce the number of invalid test cases generated by pair-wise combinatorial testing. Figure 4 shows comparison chart of test case generation with constraints and without constraints. In this chart , 4-way test case generation generates 36 test cases without constraints. But 21 test cases are generated with constraints. we conclude that to handling the right constraints between parameters can reduce fewer number of invalid test cases generated by pair-wise combinatorial testing.

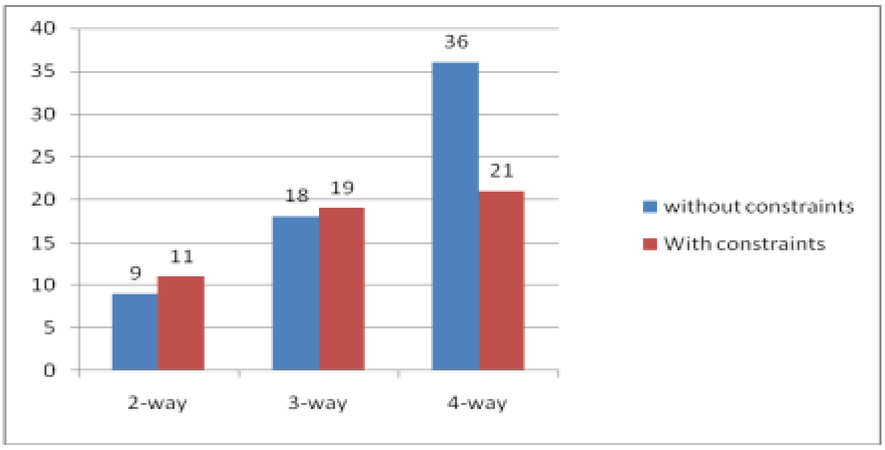


Figure 4. Comparison chart of test case generation with constraints and without constraints

V. Conclusion and Future Work

Due to handling the right constraints between parameters will reduce the number of invalid test cases generated by pair-wise combinatorial testing. The wide use of combinatorial testing will help to significantly reduce the cost of software testing while increasing software quality. It will also improve the productivity of software developers by reducing the time and effort they spend on testing. In this paper we have presented the proposed approach to generate test set for t-way testing and to handle constraints for avoiding invalid test cases. There are numerous directions to persist our work. First we execute a tool for higher level t-way strength coverage and carry out similar studies for real world applications to evaluate effectiveness of our approach. Researchers have anticipated and still are proposing new, valuable and well-organized techniques to get rid of impurities the existing methods of testing and to find feasible ways of civilizing the testing activity.

References

[1] B. Vani, R. Deepa Lakshmi, A Chronological Survey on Combinatorial Testing Strategies, Advances in Natural and Applied Sciences, pp.25-32, 2014.

[2] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun and J. Lawrence, IPOG: A General Strategy for T-Way Software Testing, 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, pp.549-556, 2007.

[3] Rick Kuhn, Raghu Kacker, Yu Lei and Justin Hunter, Combinatorial Software Testing, Vol.42, No.8, pp.94-96, 2009.

[4] D.Richard Kuhn, Raghu N.Kacker, Yu Lei, Practical Combinatorial Testing, National Institute of Standards and Technology, 2010

[5] L. Yu, F. Duan, Y. Lei, R. N. Kacker and D. R. Kuhn, Constraint handling in combinatorial test generation using forbidden tuples, IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp.1-9, 2015.

[6] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun and J. Lawrence, IPOG: A General Strategy for T-Way Software Testing, 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, pp.549-556, 2007.

[7] Sangeeta Sabharwal, P. B. (2016). Construction of covering arrays for Pair-wise Testing Using Probabilistic Approach in Genetic Algorithm. *Arabian Journal for Science and Engineering* , 2821-2835

[8] Mats Grindal, Jeff Offutt and Jonas Mellin, Handling Constraints in the Input Space when Using Combination Strategies for Software Testing, pp.1-39, 2006.

[9] Sompong Nakornburi and Taratip Suwannasart, Constrained Pairwise Test Case Generation Approach Based on Statistical User Profile, Proceedings of the International Multiconference of Engineers and Computer Scientists, Vol.1, pp.16-18, 2016.

[10] Q. Feng-an and J. Jian-hui, An Improved Test Case Generation Method of Pair-Wise Testing, 16th Asian Test Symposium, pp.149-154, 2007

[11] Renée C. Bryce , Yu Lei , D. Richard Kuhn and Raghu Kacker, Combinatorial Testing, pp.1-13, 2010.

[12] Linbin Yu, Yu Lei, Mehra Nourozborazjany, Raghu N. Kacker, and D. Richard Kuhn, An Efficient Algorithm for Constraint Handling in Combinatorial Test Generation, IEEE Sixth International Conference on Software Testing, Verification and Validation(ICSTW), pp.242-251, 2013.