Journal of Theoretical and Applied Information Technology
31st October 2018. Vol.96. No 20

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

NEURAL NETWORK BASED APPROACH FOR IMPROVING COMBINATORIAL COVERAGE IN COMBINATORIAL TESTING APPROACH

[1]RAMGOUDA PATIL, [2]V CHANDRA PRAKASH

[1]Research Scholar, [2]Professor,

Koneru Lakshmaiah Education Foundation, Deemed to be University,

Department of Computer Science and Engineering, Andhra Pradesh, India

E-mail: [1]ramgoudap@gmail.com, [2] vchandrap@kluniversity.in

ABSTRACT

Recent advancements in technology has shown significant impact on social life, where computers have attracted huge attention due to its importance in socio-economic progress. Due to the growth in various computer technologies, software-based application has played pivotal role in the social and economic development. However, poor quality of software module may cause industrial loss; hence software quality improvement remains an attractive research field. Several techniques have been presented for improving the software quality by developing software testing methods. In this field of software testing, combinatorial testing is considered as the most promising scheme for improving software testing and quality improvement by reducing the number of test cases. This combinatorial testing strategy can help to provide a better solution for given software product. In this work, we have focused on software testing using combinatorial testing with the help of IPOG approach which is used for test case generation of the 2-way test scenario. Later, neural network scheme is incorporated for test case generation which provides most suitable test scenario for combinatorial coverage. For given software product, if random testing is performed and its test cases are available, then for this software we can easily identify how much combinatorial coverage is already performed, and how many new test cases are to be added to those available test cases of random testing so that appropriate testing coverage is achieved. A comparative scheme is presented which shows that proposed approach gives the best solution for test case generation for software testing.

Keywords: *Combinatorial Testing, Neural Network, IPOG, Software Testing, Test Case Generation, combinatorial coverage.*

1. INTRODUCTION

Demand for computers and the computer-based applications have gained enormous attraction for various applications in the real-world scenario to enhance the economic prudent and social growth. In addition to this, it is widely exploited in research field and applications [1]. As a result of this growth, software development is also considered as a key component which can affect the social and economic development. Hence, a huge number of software applications are developed every year. However, maintaining the software quality remains an unnoticed aspect in this field. Software testing methods have been adopted widely to cope up with this issue of software quality maintenance. Several types of software defects are present in software products, to handle such defects numerous testing methods are also available such as, mutation testing [2], metamorphic testing [3], and structural testing [4] etc. According to the mutation testing, a tester is allowed to revise the available source code with limited constraints. This can help the tester to apply other effective tests, software weakness/bug identification and location identification of a buggy section of the code. Metamorphic testing can solve the oracle problem in testing based on the property of Software under Test (SUT); structural testing aims to find faults related to the internal structure of SUT. Recently, pair-wise testing is also discussed for monitoring the significance of software application. This process helps to find the bug or software failure conditions by performing testing where all possible tests can be carried out with the help of one test [5]. Generally, any unusual combinatorial interaction causes an issue of software failure or bug. The need of software

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

reliability and quality are becoming more essential as the dependency of human being on software is increasing. To ensure quality and reliability the software is needed to be tested on specific requirement. Combinatorial testing is one of the methods used to perform t-way testing. The need of combinatorial has increased because most failures are caused by one parameter (single-mode fault) or by the interaction of two parameters (double-mode fault). Sometimes failures may be caused because of interaction between more than two parameters (t-way). Kuhn et al. [6] presented an empirical study which shows that software failure occurs due to incorrect combinations of conditions. Software failure conditions are triggered due to single interaction but later it was found that this issue can be provoked by 3, 4, 5, and 6-way interactions. The process of generating these interactions is also known as coverage testing which is considered a complex task for manual processing, hence, to deal with this issue, Combinatorial Testing (CT), also called Combinatorial Interaction Testing (CIT) is presented for quality assurance. According to combinatorial testing, a covering array is considered as test suite which performs testing for all possible combination to obtain the desired combination which can improve the coverage performance. This approach helps us to find the interactions among the test suite which are responsible for the software failure.

Nowadays, software functionality has become very complex, varied execution environments such as distributed and networked etc. also causes complexity and design issues such as high re-configurability are also demanded which can be utilized for various software platforms by executing software in optimized mode. However, adoption of the growing software development technology may lead to the more complicated software or product development, where multiple numbers of interactions are present which may cause software failure. Due to these issues, software testing models suffer from performance issues. In contrast to this, combinatorial testing offers various advantages to overcome the coverage issues. Combinatorial testing follows various aspects such as combinatorial testing creates multiple test cases to form the covering array. This array contains various test sets where each row/ set of the parameter can be utilized for a specific test. In this, the collection of tests covers all *t-way combination* where,'t' denotes a total number of parameters available in the combination. An explanatory example of 2-way testing for 4 parameters is depicted in table 1, according to this table, column 1 and 2 can generate 9 possible combinations, and the number of test cases can be reduced to 9 from 81.

Table 1: 2-way Coverage Example

	c_1 OS	c_2 Browser	c_3 Access Type	c_4 Audio
t_1	Mac	Firefox	Modem	Creative
t_2	Linux	Netscape	Modem	Digital
t_3	Mac	IE	ISDL	Digital
t_4	Mac	Netscape	VPN	Maya
t_5	Windows	Firefox	VPN	Digital
t_6	Windows	Netscape	ISDL	Creative
t_7	Windows	IE	Modem	Maya
t_8	Linux	Firefox	ISDL	Maya
t_9	Linux	IE	VPN	Creative

Based on recent studies, it has been proved that combinatorial testing can provide effective performance for certain applications by reducing the number of test cases for testing purpose. This process is known as the specification-based technique, where no knowledge of the implementation of SUT (Software-Under-Test) is required. Moreover, test case generation process can be automated for fast processing.

Selection of optimal interaction of combination is a challenging task in this field. Conventional testing models uses manual selection model which requires more time and cost. These issues can be addressed by using any machine learning or artificial intelligence scheme by taking the advantage of the training process and due to their significant nature of prediction model. Hence in this work, we focus on development of combinatorial testing scheme and further we incorporate artificial intelligence based neural network model, to predict the most suitable interaction which can lead to improved coverage to assure the quality.

In some software development environments, once the software is implemented some set of testing is performed. Our intention in this paper is to study about how much combinatorial coverage is achieved by current test cases. Further we want to know, how many new test cases are needed to added to the current test cases to achieve complete combinatorial testing. The aim of the research is to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

perform coverage analysis and suggest additional test case to perform higher order coverage.

Main contributions of this work are as follows:

(a) Implementation of combinatorial testing model by considering synthetic database.
(b) Incorporation of a neural network model for suitable interaction prediction for improving the coverage.

In order to understand the concepts of Combinatorial Testing (CT) and Combinatorial Coverage, following definitions are important to understand.

Some Basic Definitions

Covering array(CA) : A covering array CA(N;t,k,v) is an $N\times k$ array with entries in $\{1,2,\ldots,v\}$, for which every $N\times t$ sub-array contains each t-tuple of $\{1,2,\ldots,v\}^t$ among its rows.

Example 1: Table 2 represents a covering array with N=4, k=5, v=2 and t=2.

Combinatorial Coverage: Assuming P test cases for t-wise in covering array and Q be the number of test sets. The combinatorial coverage (CC) can be defined as

$$CC = \frac{Q}{P} * 100$$

In other words, we can define Combinatorial Coverage as the percentage of test cases covered from covering array by the test set.

Generally, combinatorial coverage can be computed based on any one or both of the two methods viz., *Variable-value configuration and Simple t-way combination coverage*

Variable-value configuration (VVC): *When a set of t-Variables is given, VVC is defined as the set of t valid values, one for each of the variable. Given by*

$$VVC = \frac{Np}{Tp} * 100$$

Where Np is the number of t wise test cases generated and Tp is the Total number of test cases in t-wise coverage.

Example 2: Table 2 shows test sets for four binary variables P, Q, R and S

Table: 2 Combinatorial Test Case, initial fuzz testing Parameters

P	*Q*	*R*	*S*
0	*1*	*0*	*0*
0	*0*	*1*	*1*
1	*0*	*0*	*1*
0	*1*	*1*	*1*
1	*1*	*0*	*1*

For PQ all pair wise combinations tested hence VVC is 100%. For QR also 4 out of 4 combinations are generated, hence VVC is 100%

Similarly VVC for PR=75%, PS=75%, QS=75% and RS=75%,

If we consider 3-wise coverage for the same matrix then, combinations generated are PQR, PQS, PRS, QRS, If we compute their *Variable-value configuration (VVC): PQR=62.5%, PQS=62.5%, PRS=50%, QRS=62.5%.*

Simple *t*-way combination coverage: When a test set of n variable is given, simple t-way combinatorial coverage(S) is defined as the proportion of t-way combinations of n variables for which all variable-values Configurations are fully covered

$$S = \frac{T}{Tc} * 100$$

Where, T is the number of t-ways parameters covered completely by test set and Tc is total numbers of t-way test parameters.

Example 3: Table 2 above shows an example of four binary variables, *P, Q, R* and S where each row represents a test. Of the six possible 2-way variable combinations, PQ, PR, PS, QR, QS, and RS only *PQ* and *QR* have all four binary values covered, so simple t-way coverage (t=2) for the four tests in Figure 1 is 2/6 = 33.3%. There are six t-way (t=3) variable combinations, PQR, PQS, PRS, and QRS each with eight possible

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

configurations: 000, 001, 010, 011, 100, 101, 110, and 111. Out of the four combinations, none has all 8 configurations covered, so simple t-way (t=3) coverage for this test set is 0%.

(*t* + *k*)-way combination coverage (C_{tk}): *For a given test set that provides 100% t-way coverage for n variables, (t+k)-way combination coverage is the proportion of (t+k)-way combinations of n variables for which all variable-values configurations are fully covered.*

Let P number of configurations where VVC are fully covered and Q be Total number of configurations in *(t+k)-way combinations then*

$$C_{tk} = \frac{P}{Q} * 100$$

Table:3 Extended Combinatorial Test Cases extended from Table 2

P	*Q*	*R*	*S*
0	*1*	*0*	*0*
0	*0*	*1*	*1*
1	*0*	*0*	*1*
0	*1*	*1*	*1*
1	*1*	*0*	*1*
1	*0*	*0*	*0*
0	*0*	*1*	*0*
0	*1*	*1*	*0*

Example 4: Suppose the test sets in Table 2 are extended as shown in Table 3 then all four combinations of pairs of variables are covered, so 2-way coverage is 100%. Out of the four 3-way combinations PQR, *PQS, PRS, QRS* only the combination QRS has all 8 variable value combinations so (2+1)-way = 3-way coverage is 25%.

Tuple Density: *Sum of t and the fraction of the covered (t+1)-tuples out of all possible (t+1)-tuples*

Example5: As shown in the previous example, the test set in Table 3 provides 100% coverage of 2-way combinations and 25% coverage of 3-way combinations, so the tuple density of this test set is 2.25.

(*p, t*)-completeness: *For a given set of n variables, (p, t)-completeness is the proportion of the C(n, t) combinations that have configuration coverage of at least p.*

Example 6: For Table 2 above, there are C(4, 2) = 6 possible variable combinations and $C(4, 2) \times 2^2 =$ 24 possible variable-value configurations. From these, 19 variable-value configurations are covered and the only ones which are missing are *PQ*=11, PR=11, PS=10, QR=01, QR=10. But only two, QS and RS, are covered with all 4 value pairs. As per the basic definition of simple *t*-way (t=2) coverage, we have only 33% (2/6) coverage, but 83.33% (20/24) for the variable-value configuration coverage metric. All 2-way combinations have at least 75% configuration coverage, so (.75,2)-completeness for this set of tests is 100%. Although the example in table 1 uses variables with the same number of values, this is not essential for the measurement, and the same approach can be used to compute coverage for test sets in which parameters have differing numbers of values.

Rest of the manuscript is organized as follows: the section II deals with recent studies in the field of the combinatorial testing scheme, section III describes proposed combined model for combinatorial testing, an experimental analysis is presented in section IV and finally concluding remarks are presented in section V.

2. MOTIVATION AND RELATED WORKS

In this section, we present a brief discussion about some recent studies in the field of combinatorial testing. Various recent studies show that the combinatorial testing is widely adopted in various fields such as material designing, agriculture, biological applications and software testing. Zhang et al. [7] presented a combinatorial approach for test generation approach using pseudo-Boolean optimization process. For any given testing model, implementation and testing for each case become uneconomic hence an optimal approach is required to stop the automatic test cases and retains the maximum coverage for each case. This approach of test-case generation aims at covering array generation which can cover all t-way parameter combinations. This *article* focuses on the one-test-at-a-time algorithm for combinatorial test case generation. Further, uneconomic nature of test case generation is also discussed, and approximation ratio is considered as 0.8 and 0.9. Moreover, a self-adaptive technique is also introduced which helps to stop optimal process

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

automatically and generates the optimal number of test cases.

Ziegel et al. [8] discussed the advancements in the pharmaceutical industrial field and discussed the increasing significant production with the help of more experimental study. This work mainly focused on increasing the industrial development. According to this, combinatorial and higher throughput methods are considered as standard methods for industrial applications, whereas conventional methods are not capable to improve the quality of product manufacturing. Kampel et al. [9] discussed t-way covering for improving the quality of the software. Authors discussed that the combination of covering array can be used to formulated larger arrays. According to this model, input space model is formulated using software under test unit and t-way test suits and its component.

Petke et al. [10] studied combinatorial interaction testing technique for software testing purpose by considering various feature parameters of the system. According to some researchers, it was concluded that optimization techniques could enhance the performance of software testing modules. Based on this assumption, simulated annealing and greedy approaches have been presented for combinatorial testing. However, simulated annealing faces computation delay issues but provides better and effective solution for considered interaction. Recently, use of mobiles and applications are growing rapidly where software testing plays the role. Conventional testing approaches require more time for computation, and computation complexity issues fail to provide desired performance for testing. To deal with these issues, Vilkomir et al. [11] developed a combinatorial testing scheme for mobile application testing. It includes mobile device selection using combinatorial methods. Younis et al. [12] also utilized the combinatorial testing model which is currently growing in various industrial applications due to demand of application of product quality testing. In this field of combinatorial testing, combinatorial instance mismatching or explosion degrades product coverage and design testing accuracy. Authors have suggested that parallelization can improve the testing process by optimizing implementation cost using multi-core system architecture. In this field, the IPOG algorithm is considered as a promising technique for improving the system performance. This approach is further improved using multi-core computation strategy.

In product testing model, test case generation is considered as an important aspect. In this approach, random testing is considered as a promising technique and adopted widely in various industrial applications. However, the number of test case generation causes computation complexity in the system and leads to an application coverage. To overcome this issue, adaptive random testing is proposed as an improvement in the conventional random testing which includes fixed-size-candidate-set adaptive random testing and restricted random testing to improve the coverage of testing [13].

Borazjany et al. [14] introduced a new approach for improving the system performance with the help of a testing model. This process mainly focuses on the input space modeling for combinatorial testing improvement. This model is carried out based on the assumption as follows: input structure modeling and input parameter modeling. According to this approach, input structure modeling helps to identify the relationship among various components and parameter identification is performed in that stage. Furthermore, unit integration testing is also implemented to improve the testing performance by increasing the coverage.

The studies discussed above shows that various techniques of combinatorial testing have been presented by researchers in this field of testing. These testing models are widely adopted in industrial and software applications. Increasing demand for quality products and software urges for better quality software, hence combinatorial testing is used to improve the software testing performance by increasing the coverage of software system. However, computational complexity and computation time are the challenging parameters for combinatorial testing. Moreover, proper test case generation and selection is one of the crucial tasks for researchers. Therefore, there is a need to cover these issues by developing enhanced or artificial intelligence-based scheme for testing.

In some software development companies and environments, once the software is implemented some set of Fuzz testing is performed. Fuzz testing uses random values, but if we run 100,000 fuzz tests, how much of an improvement is that

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

compared with 50,000 tests? If we are running tests on binary executables and don't have the source code, we cannot measure path coverage in the code. Using combinatorial coverage, we can provide a quantitatively defensible answer to the question of how two test suites compare. For example, we might find that the 50,000 fuzz tests achieve 80% of 3-way coverage and other proportions of 4-way etc., but the 100,000 fuzz tests have 85% 3-way coverage, so it may not be cost effective to do the larger test set. In [17] similar work is implemented for practical recommendation for using combinatorial coverage in specifying requirements of combinatorial coverage analysis.

3. PROPOSED MODEL

Based on literature review study, it can be understood that, there is a need to improve the combinatorial testing performance which can be obtained developing an enhanced artificial intelligence technique. In this section, we present a new approach to combinatorial testing with the help of artificial intelligence scheme. In this field of combinatorial testing, a conventional IPOG scheme is implemented which can perform multi-way testing [15]. The IPOG scheme is used because of following two-fold design objectives: first, IPOG scheme helps to generate random strategy generation which can be implemented for any software configuration i.e. there should be no restriction on the system configuration parameters raised by generated strategy. Moreover, this assumption improves the performance of computational approaches over algebraic approaches. In next objective, *t-way* testing demand for more time and space requirements when compared with pairwise testing because time increasing number of combination leads to increased coverage. The general framework of IPOG testing approach is formulated using two main steps as follows: Let us consider that, a system is provided with t or more parameters where IPOG technique is implemented resulting in the formulation of *t-way* test for the initial t parameters. Later it can be extended up to *t way* test set for *t+1- parameters* and then it can be extended further until *t-way* test set is built for all parameters. This extension of t-way testing coverage can be applied by following two steps as horizontal growth and vertical growth. According to horizontal growth, existing test can be extended by adding one value to the existing test set whereas vertical growth is introduced by adding and producing a new test. Algorithm 1 shows the complete procedure of IPOG test generation model. This approach considers two main arguments which are the strength of coverage and parameter set. The number of parameter set is assumed to be equal or greater than t.

Algorithm 1: IPOG-Test

IPOG-Test (ini, t, Parameter set P_s)
{
1. Initialize test set t_s to be an empty set.
2. Sort the parameter in set P_s in a non-increasing order of their domain sizes, and denote them as P_1, P_2 ⋯ and P_k.
3. Add into the test-set t_s a test for each combination of values of the first t parameters
4. For (int $i = t + 1; i \leq k; i++$)
{
5. Let π be the set of all t −way combinations of values involving parameter P_i and any group of $(t-1)$ parameters among the first $i-1$ parameters
6. // horizontal extension of parameter P_i Column extension
7. For (each test $\tau = (v_1, v_2, \ldots, v_{i-1})$ in test set ts)
{
8.. Choose a value v_i of P_i and replace $\tau = (v_1, v_2, \ldots, v_{i-1}, v_i)$ so that τ' cover the most number of combinations of values in π
9. Remove from π the combinations of values covered by τ'//
10. // Vertical extension process, New row will be created(Row extension)
11. for (each combination σ in set π)
{
12. if (there exists a test that already covers σ)
{
13. Remove σ from π
}
14.else
{
15. change an existing test, if possible, or otherwise add a new test to cover σ and remove it from π}
}
}
16. Return test set t_s

The combinations in set π are covered in the following two steps: first step is known as horizontal growth. As per the algorithm, a value is added for the parameter pi (lines 7 - 9) is an extension. Added values are selected in a stingy

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

way so that the added value can cover maximum number of combinations in set π (line 8). Each time a value is added, the set of combinations covered due to this addition are removed from set π (line 9). Another step is known as vertical growth: This step covers the remaining uncovered combinations, one at a time, either by changing an existing test or by adding a new test. When we change a test to cover a combination, only don't care values can be changed. A don't care value is a value that can be replaced by any value without affecting the coverage of a test set. If no existing test can be changed to cover σ, a new test needs to be added in which the parameters involved in σ are assigned the same value in σ and the other parameters are assigned that don't care about values.

After achieving this solution, we discuss about the complexity issues of IPOG algorithm. The complexity of this depends on the storage of π which helps to cover each new parameter. Let us consider that total number of parameters are denoted by n and domain size is denoted by d, hence the required space to store π can be denoted as $O(d^t \times n^{t-1})$. The time complexity is dominated by horizontal extension.

Above discussion presents a brief explanation about test case generation for testing purpose. Here our main aim is to improve the product testing coverage with the help of test suits. Generally, conventional techniques use manual inspection of product testing which requires more time and manual effort which makes it cost inefficient for real-time implementations. In this approach, prediction of test suit is a crucial task for researchers. However, machine learning techniques have been used widely in various researches. Corma et al. [16] used neural network scheme for combinatorial testing using prediction method. Similarly, in this work, we use neural network-based approach to select the best suitable test case which can improve the overall coverage of the testing application. A brief discussion about neural network and its use in combinatorial testing is presented in next subsection.

3.1 Neural Network Approach for Combinatorial Testing

Artificial neural network is one of the machine-learning approaches and it is adopted in various real-time applications. In this work, we aim on the prediction of best test suit which can improve the coverage of the system. Artificial neural network contains various components such as interconnected processing unit which is also known as neuron. In other words, neural network is formulated with the help of several neurons which can be organized in different topology or neural architecture. A basic unit of neural network composition is as follows:

- This network model contains various connection in the form of set which are also known as inputs i.e. $x_i(z)$.
- Each input data is further characterized and represented with the help of weight W_{ij} which shows relation between current layer and previous layer
- A propagation rule is also defined i.e. back propagation of forward-propagation for computation purpose.
- Finally, an activation function is also defined which helps to determine the output term as y_k

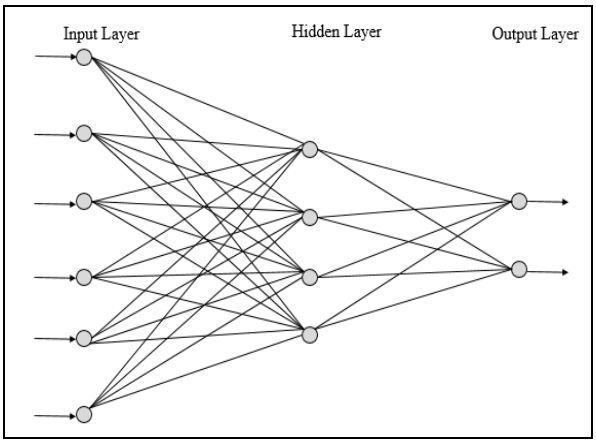

Figure 1: Neural Network

Neural network technique provides faster computation and achieves the answer for any given problem with the reduced delay. According to neural network process, any problem can be resolved by following two main steps which are: training/learning and testing. During training process, neural network is provided some samples of the data which belongs to the input of the problem. Further, mathematical correlation is applied between samples for further analysis. In neural network applications, multilayer perceptron is considered as most promising technique which is

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

generally implemented for prediction purpose as depicted in figure 1. According to multilayer perceptron, all neurons are grouped in the form of layers. Here, previous layer neuron is considered to input two the next layer. In this process, neurons of input layer contain all the features of input problem hence based on these parameters input and output layers of the network are determined which also known as single class classifier and multi-class classifier.

The proposed approach comprises of two approaches, test set generation and neural network prediction. In order to present the complete scenario, let us consider different variables and their values as presented in Table 4.

Table 4: Variables & Notations, Case study of Configuration testing

S. No.	Considered Variable	Variable Parameters
1	Database	DB_1, DB_2, DB_3 $\{Mysql, Oracle, Sybase\}$
2	Processor	P_1, P_2 $\{Intel, AMD\}$
3	Operating system	OS_1, OS_2, OS_3, OS_4 $\{Windows-7, Linux, Xp, MAC\}$
4	Browser	B_1, B_2 $\{Firefox, IE\}$

Based on the variables, total combinations are 46 which are as follows: $\{DB_1, P_1\}$,$\{DB_1, P_2\}$, $\{DB_1, OS_1\}$, $\{DB_1, OS_2\}$, $\{DB_1, OS_3\}$, $\{DB_1, OS_4\}$, $\{DB_2, P_1\}$, $\{DB_2, P_2\}$, $\{DB_2, OS_1\}$, $\{DB_2, OS_2\}$, $\{DB_2, OS_3\}$, $\{DB_2, OS_4\}$ etc. In the next phase, neural network model is constructed which contains input, hidden and output layers. Input layers take variables as input values, hidden layers are equal to the number of attributes and finally, predicted output is obtained from output layer. Hidden layers are connected through computing nodes. Every computing node in the hidden layer and succeeding hidden layer connection can provide the connectivity between the processing nodes. This process is repeated until the output layer is obtained. A pictorial representation is given in figure 2.

In order to obtain the desired performance, proposed approach is followed as mentioned in algorithm 2.

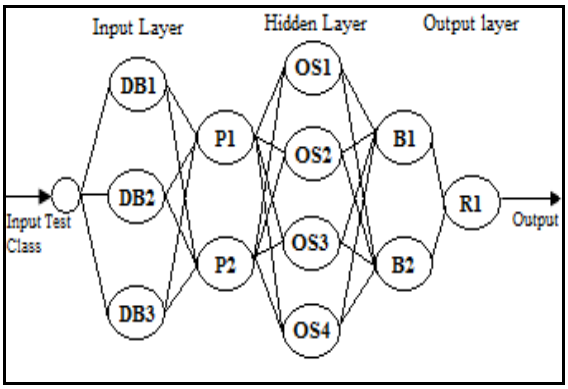

Figure 2: Proposed Neural Network Modeling

Algorithm 2: Test Case Generation and Neural Network Prediction

init(TestGen(),NNP())
{
1. Initialize the test case parameters as given in table 4
2. Formulate a vector of similar size as given test case parameters
3. Number of hidden layer selection
4. Check whether hidden layer is also the first layer of the network
5. Pair generation using IPOG, considering that all elements are present in the hidden layer (for training database)
6. Test case vector formulation for further process
7. If hidden layer is not the first layer of the network

{
8. Find current layer number
9. Select last element of the row as stored in the vector matrix
10. Data value selection from hidden layers; at each layer

{
11. Pair formulation using hidden layer value obtained from the matrix
12. Perform filtering the data by comparing with the previous hidden layer.
13. Update vector in the matrix
14. Predicted vector with the help of hidden layers
15. Add the additional predicted output
16. Coverage analysis

}
}
}

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

With the help of this complete process, following test cases will be generated which can be used for neural network learning. These test vectors are presented Table 5.

Table 5: Test Vectors for Case study of Configuration testing

Test vector	Vector elements			
	Vector 1	Vector 2	Vector 3	Vector 4
TV1	DB_1	P_1	OS_2	B_2
TV2	DB_1	P_2	OS_3	B_1
TV3	DB_1	P_2	OS_4	B_1
TV4	DB_1	P_2	OS_1	B_2
TV5	DB_1	P_1	OS_3	B_2
TV6	DB_2	P_2	OS_3	B_2
TV7	DB_2	P_1	OS_2	B_1
TV8	DB_2	P_1	OS_4	B_1
TV9	DB_3	P_1	OS_4	B_1
TV10	DB_3	P_1	OS_1	B_1
TV11	DB_3	P_2	OS_2	B_2
TV12	DB_3	P_2	OS_3	B_2

Initially, we have implemented t-way coverage testing on the considered vector where total 3 possible combinations are found missing which are as follows: (a) 3,1,1,1 (b) 2,1,4,1 and (c) 3,1,4,1. Since our experiment is carried out for two way testing hence total missing combinations are given in table 6.

Table 6: Missing Combinations

Combination ID	Combination Variable
1	A, B
2	A, C
3	A, C
4	A, C
5	A, D
6	B, C
7	C, D

According to the analysis missing combination 3,1,4,1 has maximum two-way missing combinations which are as given in table 7 where with all three missing combinations; total two-way coverage is obtained as 0.84091.

Table 7: Missing Combination Values

Combination Variable	Combination Value
A, B	3,1
A, C	2,4
A, C	3,1
A, C	3,4
A, D	3,1
B, C	1,4
C, D	4,1

In order to improve the coverage, we try to predict most suitable test case for the computation. In this process, first of all we consider 3,1,4,1 as new test vector and coverage is obtained as 0.95455 which is later improved by adding 2,1,4,1 as new test vector and the total coverage is obtained is 0.97727.

4. EXPERIMENTAL STUDY

This section provides a complete experimental study of proposed neural network based combinatorial testing model. Performance of proposed approach is measured in terms of coverage, domain size and total number of parameters. Complete experimental study is carried out using windows 7 operating system with 1.8 GHz central processing unit with 8GB RAM memory.

According to the experimental study, the initial parameter is fixed to 15 with domain size 4 with a varied from 3 to 6. In this process, experimental results are presented in table 5 where test size ratio, time ratio, size and time parameters are considered for analysis. Proposed model is compared with the conventional IPOG algorithm as depicted in table.

Table 8: Comparative analysis of IPOG approach and Neural-Network based approach

	IPOG Approach		Neural-Network Based-IPOG		Time and size ratio analysis	
t-way	Time	Size	Time	Size	Time Ratio	Size Ratio
3	0.561	181	0.16	204	0.29	1.12
4	16.587	924	0.77	1056	0.046	1.14
5	230.22	4519	11.10	6785	0.048	1.50
6	2152.11	20384	310.26	32543	0.144	1.59

Journal of Theoretical and Applied Information Technology
31st October 2018. Vol.96. No 20

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 8 shows comparative analysis of IPOG and proposed Neural Network based IPOG approach based on the time and size ratio analysis. This analysis is carried out for 3 to 6 way testing. Proposed Neural Network based -IPOG shows improved performance when compared with IPOG. As discussed in previous section, we have considered an experiment where *t*-way study is presented. Based on this analysis, total number of combination graphical representation is given in figure 3.

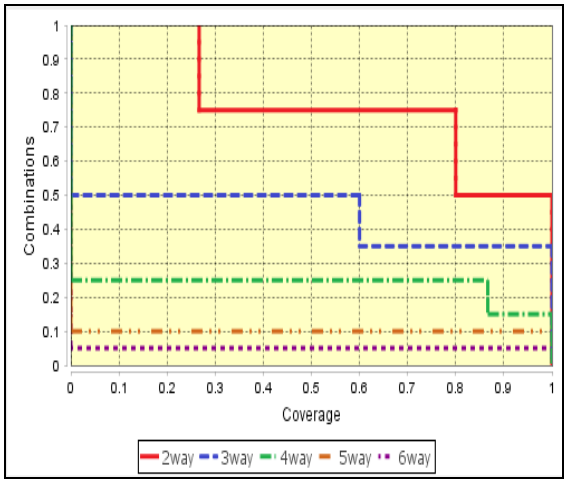

Figure 3: Combinations and Coverage

As shown in Figure 3, combinations and coverage are measured for varied number of testing. This analysis shows that 50% combinations are capable to obtain the 100% coverage using two-way testing. Similarly, 3-way testing requires 35 combinations for 60% coverage.

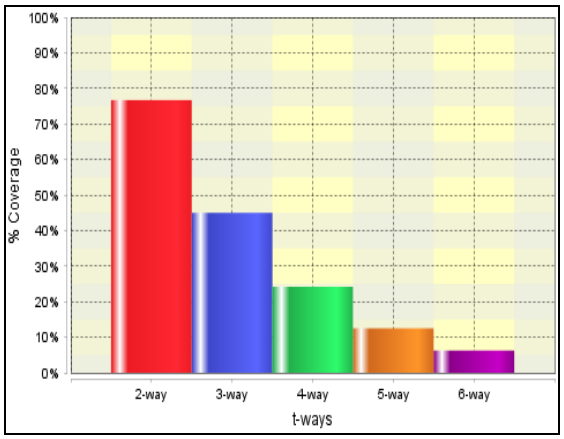

Figure.4: percentage of coverage versus t-way testing up to 6-way CT

Similarly, figure 4 presents a comparative analysis by considering varied number of testing. In this analysis, *t*-ways are varied from 2-way testing to 6-way testing and corresponding to this percentage coverage is computed. Study shows that 2-way testing obtains more testing coverage when compared with other testing. Figure 5 represents the coverage heat map.

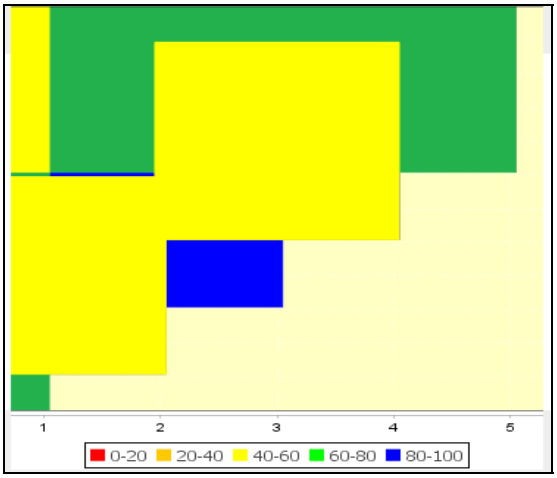

Figure 5: Coverage Heat map

5. CONCLUSION

In this paper we have presented a novel approach for two-way testing using neural network model to improve the testing coverage. According to this work, first of all IPOG test case generation strategy is implemented for 2-way test case and total coverage is measured. In subsequent phase, neural network model is implemented, where hidden layers contain input parameters and output is obtained in terms of best missing test case which can enhance the overall coverage of the system. Based on this approach, an extensive experimental study is presented for 2-way testing and further IPOG experiments are also depicted for 6-way testing. From this study it is concluded that proposed approach can be used for improving the coverage for two-way testing.

In this research work we have assumed that some previous fuzz testing has executed. On the test data of fuzz testing, we are finding combinatorial coverage for pair-wise testing. If the fuzz testing test case is not properly selected then this work may not provide the optimum results.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REFERENCES

[1] Ahmed, J., Siyal, M.Y., Najam, S. and Najam, Z., 2017. Challenges and Issues in Modern Computer Architectures. In Fuzzy Logic Based Power-Efficient Real-Time Multi-Core System (pp. 23-29). Springer Singapore.

[2] Clegg, B.S., Rojas, J.M. and Fraser, G., 2017, May. Teaching software testing concepts using a mutation testing game. In Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track (pp. 33-36). IEEE Press.

[3] Jiang, M., Chen, T.Y., Kuo, F.C., Towey, D. and Ding, Z., 2017. A metamorphic testing approach for supporting program repair without the need for a test oracle. Journal of systems and software, 126, pp.127-140.

[4] Pandey, A. and Banerjee, S., 2017. Bio-Inspired Computational Intelligence and Its Application to Software Testing. In Handbook of Research on Soft Computing and Nature-Inspired Algorithms (pp. 429-444). IGI Global.

[5] Lopez-Herrejon, R.E., Javier Ferrer, J., Chicano, F., Haslinger, E.N., Egyed, A. and Alba, E., 2014, July. A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (pp. 1255-1262). ACM.

[6] D. R. Kuhn, D. R. Wallace and A. M. Gallo, "Software fault interactions and implications for software testing," in IEEE Transactions on Software Engineering, vol. 30, no. 6, pp. 418-421, June 2004.

[7] Zhang, Z., Yan, J., Zhao, Y. and Zhang, J., 2014. Generating combinatorial test suite using combinatorial optimization. Journal of Systems and Software, 98, pp.191-207.

[8] Ziegel, E.R., 2003. Experimental design for combinatorial and high throughput materials development.

[9] Kampel, L., Garn, B. and Simos, D.E., 2017, March. Combinatorial Methods for Modelling Composed Software Systems. In Software Testing, Verification and Validation Workshops (ICSTW), 2017 IEEE International Conference on (pp. 229-238). IEEE.

[10] J. Petke, M. B. Cohen, M. Harman and S. Yoo, "Practical Combinatorial Interaction Testing: Empirical Findings on Efficiency and Early Fault Detection," in IEEE Transactions on Software Engineering, vol. 41, no. 9, pp. 901-924, Sept. 1 2015.

[11] S. Vilkomir and B. Amstutz, "Using Combinatorial Approaches for Testing Mobile Applications," 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops, Cleveland, OH, 2014, pp. 78-83.

[12] Younis, M.I. and Zamli, K.Z., 2010. MC-MIPOG: A parallel t-way test generation strategy for multicore systems. ETRI journal, 32(1), pp.73-83.

[13] R. Huang, X. Xie, T. Y. Chen and Y. Lu, "Adaptive Random Test Case Generation for Combinatorial Testing," 2012 IEEE 36th Annual Computer Software and Applications Conference, Izmir, 2012, pp. 52-61.

[14] M. N. Borazjany, L. S. Ghandehari, Y. Lei, R. Kacker and R. Kuhn, "An Input Space Modeling Methodology for Combinatorial Testing," 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, Luxembourg, 2013, pp. 372-381.

[15] Lei, Y., Kacker, R., Kuhn, D.R., Okun, V. and Lawrence, J., 2007, March. IPOG: A general strategy for t-way software testing. In Engineering of Computer-Based Systems, 2007. ECBS'07. 14th Annual IEEE International Conference and Workshops on the (pp. 549-556). IEEE.

[16] Corma, A., Serra, J.M., Argente, E., Botti, V. and Valero, S., 2002. Application of artificial neural networks to combinatorial catalysis: Modeling and predicting ODHE catalysts. ChemPhysChem, 3(11), pp.939-945.

[17] Kuhn, D. Richard, Raghu N. Kacker, and Yu Lei. "Measuring and specifying combinatorial coverage of test input configurations." *Innovations in systems and software engineering* 12.4 (2016): 249-261.