International Journal of Engineering & Technology, 7 (1.3) (2018) 74-81

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

A Neural Network Based Strategy (NNBS) for automated construction of test cases for testing an embedded system using combinatorial techniques

Lakshmi Prasad Mudarakola [1*] and J. K. R. Sastry [2]

[1]Ph.D Scholar, Dept. of Computer Science & Engineering, NBKR Institute of Science and Technology Vidyanagar, Nellore, AP, India
[2] Dept. of Electronics and Computer Science Engineering, Koneru Lakshmaiah Educational Foundation Vaddeswram, Guntur District, AP, India.
**Corresponding author E-mail: prasad.hinduniv@gmail.com, drsastry@klce.ac.in*

Abstract

Testing an embedded system is required to locate bugs in software, diminish risk, development, repairs costs and to improve performance for both users and the company. Embedded software testing tools are useful for catching defects during unit, integration and system testing. Embedded systems in many cases must be optimized by engaging crucial areas of the embedded systems considering all factors of the input domain. The most important concern is to build a place of test cases depend on design of the requirements that can recognize more number of faults at a least rate and point in time in the major sections of an embedded system. This paper proposes a Neural Network Based strategy (NNBS) to generate optimized test cases based on the considerations of the system. A tool called NNTCG (Neural Network Test Case Generator) has been build up based on the method proposed in this paper. Test cases are generated for testing an embedded system using NNTCG and the same are used to determine the expected output through the neural network and the output generated from the actual firmware. The faulty paths within the firmware are determined when the output generated by the neural network is not same as the output generated by the firmware.

Keywords: *Automated Test Case Generation; Input Domain Testing; Output Domain; Embedded Systems; Combinatorial Testing; Neural Networks.*

1. Introduction

Testing plays a vital role in evaluating the application including its components, behavior, performance, and robustness. One of the main criteria is to be as defect-free as possible. Expected behavior, performance, and robustness should be formally described and measurable. Verification and Validation activities are also very important to focus on the quality of the software product. Testing is most often considered as a detective measure of quality, it is strictly related to corrective measures such as debugging.

Combinatorial testing is a combination based technique that provides an efficient way to choose combinations of program inputs or features for testing. It is an efficient testing technique to test hardware/software that reveals failures in a certain system based on input or output combinations. It has been practical over the years to test system configurations, graphical user interfaces, software product lines, web forms, protocols, etc.

Embedded systems are clever devices that are infiltrating our daily lives such as the cell phone in your pocket, all the wireless infrastructure behind it, the Palm Pilot on your desk, the Internet router your e-mails are channeled through, your big-screen home theater system, the air traffic control station as well as the delayed aircraft it is monitoring.

Now a day, most of the software's formulates up 90 percent of the value of these devices. Testing timing constraints is as significant as testing functional performance for an embedded system.

Designing a test case is place an important role in forming out the appropriate PCOs which depends on the kind of testing to be performed, such as functional, structural, load, and so on and also how to exploit these PCOs information to send and to expect to receive through them, and in what order.

A Neural network is an inter association of several computing nodes that are systematized into three distinct partitions that include Input layer, output layer and a set of middle processing layers. The neural network can be recognized for generating the test cases of embedded system based on the constraints of the system.

The neural networks must be used for undertaking testing which is proved to generate less number of test cases. There are many test case generation methods that exist in the literature and all of them slightly differ in terms of number of test cases generated.

In this paper, a neural network based strategy is developed and used for generating the combinatorial test cases for testing an embedded system. A comparative analysis also is provided that shows that the proposed strategy produces minimal of the testing cases and therefore is found to be more effective compared to other test case generation methods.

2. Literature Survey

Now a day, combinatorial testing is widely used in several applications to test the entire system in order to reduce the time and cost. Numbers of articles have been published surveying the idea of generating the test cases using different types of combinatorial testing. A wide variety of different approaches and accomplishments have been developed for generating combinatorial test sets in the literature.

More number of test cases van be generated by using an automated tool. However, no one can guarantee the usefulness of all those test cases that are generated. It is always useful to learn a model based on historical evidence collected out of testing. Suh kind of models can be effectively used for generating the test cases which will reveal more of bugs existing in the system. C. Anderson, et al., [1] used the neural networks to guide software testing activities.

Many inputs to a system are inter-related. These inter-relations can be used for generating test cases. A table can be organized and the interactions among the inputs can be included into the table as rows. The Table is then used to enumerate the rows available in the table. Through use of some deterministic procedures or random procedures based on their suitability Cohen D.M., et al., [2].

Many systems and tools have been developed for generating test case based on a chosen set of variables at a time. AETG is such a system proposed by D.M. Cohen, et al., [3]. The method proposed by them considers all combinations of the chosen set of input variables. The number of rows generated grows logarithmically considering the number of input variables selected for generation of test cases making it in feasible when the number of Input variables increases and becomes quite un-wieldy when the number of input variables grows beyond 10 variables. The test cases generated can be used for testing variety of applications.

Many inputs to a system comprise input domain for the system. Sometimes excellent test cases can be generated when the input variables are ordered according to some criteria such as the frequency of usage of such variables. In pair-wise testing which is specification based all possible domain values are to be considered while generating the test cases Yu Lei, et al., [4]. While generating the test cases, it should be ensured that every pair if input variables or combination of input variables must be represented by at least a single test case Lei, Y., et al., [5].

The code is not made visible to the user. Testing of binary code can only be done using the inputs to the system and with the knowledge about the expected outputs from the system. One can generate effective test cases if the software is developed using fewer number of components. The test configurations in this case are few in number. It is not feasible to generate the required number of test configurations if the number of components that comprise the software increases drastically. S.A. Ghazi, et al., [6] had used genetic algorithms to maximize source code coverage through generated test cases considering different combinations of input variables and also fixing the upper limit on the number of input combinations that can be chosen for generating the test cases.

In literature many algorithms have been in existence for generating the test cases required for undertaking testing of software systems. The famous methods include TConfig, AETG, TCG, Annealing, IPO etc. Many heuristic search methods are also being used for generation of test cases using combinatorial methods. Greedy methods are having been proven to be quite effective considering small size of the test suites. These methods can generate the test cases quite fast and also will accommodate various test scenarios. However, the size of the test suites in this case is quite high and do not guarantee acceptable performance guarantee. Density based algorithms have been proposed Bryce R. C., et al., [7] addresses the performance guaranty even in the presence of large size of test cases.

Regression testing is about testing the effected code due to changes made to the software. It is the question of selecting the test cases that are related to the effected code and then use the same for undertaking the effected codes. The conception of generating test cases based on input pairs in a way represents NP problem which only can be solved by greedy, heuristic and algebraic methods. Most recent advancement to the issue of test case generation is by using the method called any colony optimization (ACO), invented by Kewen Li, et al., [8]. The ACO method will generate only few test cases that cover maximum number of input variables. Regression testing becomes feasible solution when the numbers of test cases involving more input variables are less in number.

Neural networks are being used extensively for many purposes including generating test cases based on combinatorial methods. A model is leant using the input variables and expected output. The constants used to undertake modeling is adjusted such that the model generates the expected output for given set of input variables. Lilan Wu, et al., [9] used a process called back-propagation as a part of modeling a neural network to adjust the coefficients used for building a Neural network(NN). The method presented by the authors could produce all the faults existing in the software that could have triggered while undertaking functional testing.

The method of developing a NN model has father been extended by Yogesh Singh, et al., [10] considering each node as an Object that has been modelled using some computational metrics. The model has been used for assessing the quality of the software. They have used the model using data published by NASA for predicting the effort required to develop quality software.

Finding rare faults that occur in rare input combinations and the interrelations between the input variables is quite difficult. Many tools have been in existence to generate test cases based input pairs. Test data is generated in such a way that at least one test case covers a given input pair. Covering arrays have been invented that can be fitted with all input variables into different rows in the array, Model checking has been in use for long time to see whether a model can have developed using which a test case can be developed and also to verify whether all the input pair combinations have been included into the model. Covering arrays methods has been integrated with the modeling checking methods to arrive at more formidable system. R. Kuhn, et al., [11] have presented a method using which covering arrays are converted into executable test cases.

Testing of a system can be undertaken to cover any of the features with which software is developed. Interacting with the system gives quite a good idea of the type of test case that must be inputted. The sequence and the ordering of the test cases can also be better understood. Xiang Chen, et al., [12] have developed an algorithm that considers the density of the variables and the domain values of those variables. The weighted density based algorithm developed by them is coupled with the covering arrays method. For prioritization of test cases, they have used ant colony optimization (ACO) method.

Generating the test cases and use them for undertaking testing in exhaustive is time consuming and expensive. All combinations of input variables are to be considered for generating test cases that helps to test the system in totality. Generation of test cases using the input pairs reduces the number of test cases while still holding to test the entire system exhaustively. An advanced method has been invented by J.D. McCaffrey, et al., [13] called bee colony optimization (BCO) for generating test cases based on input pairs. The algorithm produced fewer test cases compared to many other methods for testing the same system. However, more time is required for generating the test cases using BCO.

Generating pair-wise test cases considering all input variables and complete domain values of each of the input variable is quite complex. A vector containing the values that should be considered for is variable must be constructed. The number of vectors to be considered is equivalent to number of input variables aiming the processing of too many vectors makes the system much complicated to generate the test cases. The problem as such is near to NP complete problem. Genetic algorithm requires few inputs for generating all the required test cases. James D. McCaffrey et al., [14] have proposed a method which uses a genetic algorithm to generate the test cases considering fewer test cases to start-width. However, it may take more time than other methods to ensure that all the required test cases have been generated.

Rick Kuhn, et al., [15] had introduced the combinatorial software testing which delivers the first in-depth book on practical combinatorial testing and emphases on real-world software testing, including cost considerations. They also presented step-by-step procedures for applying advanced combinatorial test techniques.

Most of the methods that have been presented to generate test cases based on certain number of input variables considered at a time try to reduce the size of input variables and at the same find all the test cases that can executively test entire software system. The problem of generating fewer test cases using minimum input variables that can exhaustively test the system is quite complicated. Xiang Chen, et al., [16] has used Particle swarm optimization (PSO) for filtering out the generated test cases using input pairs. PSO is a meta-heuristic search technique that aims to cover all the paths that exists within the test cases. A single test case is selected that covers a path existing in the system.

It has been found that not every input variable contributes to a fault. It has also been found that few of the interactions among the input variables contribute to the faults heavily. D. Richard Kuhn, et al., [17] has presented a review on key methods, concepts, and tools that can be used for generating the test cases. They have discussed on the use of formal methods for determining the expected results when a test case is presented for undertaking the testing.

Black box testing is a way to generate the test cases based on the knowledge of input variables, interconnections among the input variables and the expected outcomes. Requirement specification is also used as the basis. Most of the combinatorial methods considered input variables and very few have attempted to generate the test cases based on the output domain. Some embedded systems deal with very few output variables and therefore is the best case to consider output domain compared to the input domain for generating the test cases in faster manner and also ensure that all critical conditions have been thoroughly tested. Chandra Prakash V, et al., [18] [19] have presented a method for generating the test cases considering the output domain and the criticality specification of an embedded systems using genetic algorithms. Few test cases related to criticality have been fed as input using which test cases have been generated which are then added to the test cases generated through a combinatorial testing method has been considered.

Every organization has to decide on the approach to be followed for undertaking in terms of whether testing has to be carried using automated tools or carry testing manually. Kristina Smilgyte, et al., [20] have presented a method considering neural network based application show the effort and time required for undertaking testing using automated tool or by using manual methods.

Generating pair wise test cases in such a way the fewer pairs are selected and at the same time generating test cases that tests all paths existing in the source code is quite completed. To solve this issue Kristina Smilgyte, et al., [20] have presented a test case generation method using neural networks and Manisha Patil et al., [21] have used simulated annealing which together produced excellent results in terms of reducing the number of test case and to reduce time required to generate those test cases.

It is evident that optimization is certainly required while generating test cases using input pairs. Priti Bansal, et al., [22] have presented a method that selects the initial test cases using hamming distance between the input variables. The have also presented an algorithm that find cross over points based on which the individual test cases can be combined leading to reduction in number of test cases that must be generated. They have implemented the method through a tool called PWiseGen which is open source.

R. Raju, et al., [23] had introduced a neural network approach for randomized unit testing based on genetic algorithm. It has been found that finding faults that are produced due to interaction among the input variables is quite complicated. Many search based techniques have been introduced to find the faults that could be triggered due to interaction. An Overview on the use of search based techniques to find faults that can happen due to interaction have been presented by Huayao Wu, et al., [24].

An advanced search based technique has been proposed by H. L. Zakaria, et al., [25] called Migrating Bird optimization (MBO) for optimizing the test cases which are generated through a pair wise test case generation method. Two algorithms have been presented by them, a basic algorithm (MBO) and an improved MBO called iMBO. Multiple neighborhoods structures have been considered in the case of iMBO for optimizing the test cases. It has been proved by them that these two algorithms produce fewer test cases and in quite a reduced time.

It has been shown that genetic algorithms combined with other methods would produce optimum test cases while at the same time supporting the exhaustive test cases wherever required. R. Qi, et al., [26] have presented an algorithm called “hybrid optimization” method which is enhancement of a method based on genetic algorithm and also coupled with another algorithm called hill climbing. The approach produced acceptable results.

Yet another advanced method has been presented by Akihisa Yamada, et al., [27] for generating test cases specially to determine the test tuples that have been ignored during a search method especially when the greedy search method has been employed. They have ignored the constraints that are imposed on the test tuples. They have improved the algorithms further by affecting the constraints using the un-satisfied tuples as well.

3. Pilot Project On Temperature Monitoring and Controlling Of Nuclear Reactor Systems (TMCNRS)

A Pilot project has been developed which is meant for monitoring and controlling temperatures within a nuclear reactor system (TMCNRS), the hardware block diagram of which is shown in Figure 1.

Temperature sensors fixed in the nuclear reactor tubes measures the temperatures, and the analog signals representing the temperatures are amplified by signal conditioners and the same are fed as input to A/D converter. The firmware placed in the microcontrollers reads the temperatures into digital form and the same are placed in RAM.

Two pumps and a Buzzer are connected to the system. The pumps control the flow of coolant into nuclear reactor tubes. The operation of the pumps is controlled based on whether a sensed temperature is greater than a reference temperature or otherwise. The pumps are operated through relays which can be made to be open or closed by triggering a signal on the output PIN to which the relays are connected. A host connected to the Micro Controlled

through RS232C interface is used for feeding the reference temperatures which are used for comparing with the sensed temperatures.

The TMCNRS is protected through password protection. Keyboard which is connected to the micro controller system is used for imputing the password. An LCD is also connected to the Micro controller system for displaying the temperatures that are sensed by the sensors and also for displaying alerts when huge variations are sensed in temperature gradients.

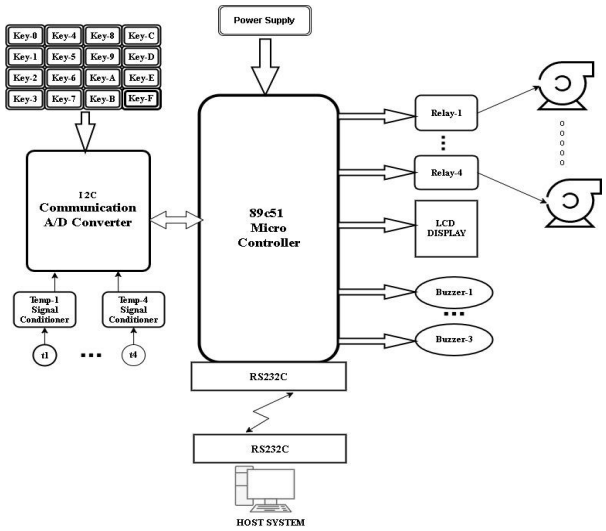

Fig. 1: System Integration Diagram of TMCNRS

The requirements that must met by TMCNRS are shown in Table 1.

Table 1: Requirement Specification of TMCNRS

S. No.	Functional requirements
Req. 1	Temperature 1 must be sensed once in 10 milliseconds and the same must be displayed on LCD and transmitted to a remote HOST. The Relay connected to PUMP 1 must be opened when the reference temperature 1 is less than the sensed temperature 1.
Req. 2	Temperature 2 must be sensed once in 10 milliseconds and the same must be displayed on LCD and transmitted to a remote HOST. The Relay connected to PUMP 2 must be opened when the reference temperature 2 is less than the sensed temperature 2
Req. 3	The buzzer must be activated if the temperature gradient between the temperature 1 and temperature 2 is beyond the expected regions.

4. Neural Network Based Strategy (NNBS) For Generating Pair Wise Test Cases

4.1. Introduction to Artificial Neural Networks

An artificial neural network is a computational model and is represented as directed graph with weights in which artificial neurons are nodes and directed edges with weights are connections between neuron outputs and neuron inputs. The symbolic representation of the same is shown in Figure 2.

The Artificial Neural Network obtains input from the external world in the form of pattern and image in vector form. All the inputs are mathematically represented by the notation x (n) for n number of inputs. Each input is multiplied by its equivalent weights. Weights are the information used by the neural network to solve a problem. Naturally weight represents the strength of the interconnection between neurons inside the neural network.

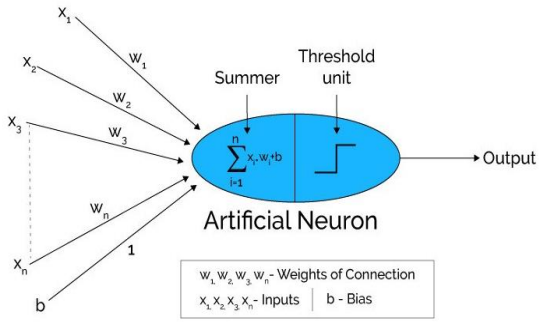

Fig. 2: Artificial Neural Network

All the inputs with their weights are summed up within a computing unit (artificial neuron). In case the weighted sum is zero, bias is added to make the output not- zero or to scale up the system response. All the time the Bias value is equal to '1'.

The sum keeps up a correspondence to any numerical value ranging from 0 to infinity. In order to bind the response to arrive at desired value, the threshold value is set up. For this, the sum is accepted through activation function. The activation function is put of the transfer function used to get desired output. There are linear and non-linear activation functions.

Some of the widely used activation function are binary, sigmoidal (linear) and tan hyperbolic sigmoidal functions (nonlinear). In Binary, the output has only two values 0 and 1. The threshold value is set up i.e. if the net input weight is >1, an output is set to be 1 otherwise zero. In case of Sigmoidal Hyperbolic, it has a function of 'S' shaped curve. The last one is tan hyperbolic function which is used to fairly accurate output from net input. The function is defined as f (x) = (1/1+ exp (-σx)) where σ—steepness parameter.

4.2. Architecture of Artificial Neural Networks

A typical neural network consists of large number of artificial neurons called units, which is organized in a series of layers and comprise different layers – Figure 3 shows such architecture.

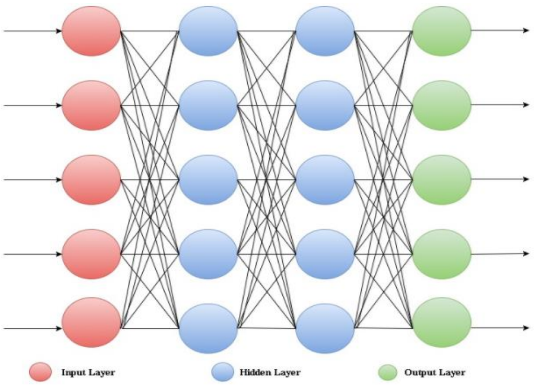

Fig. 3: Architecture of Artificial Neural Networks

Input layer—it contains those units (artificial neurons) which obtain input from the outside world on which network will learn, recognize about or otherwise process.

Output layer—it contains units that reply to the information about how it's learned any task.

Hidden layer—these units are in between input and output layers. The job of hidden layer is to transform the input into something that output unit can use in some way.

Types of Artificial Neural Networks

There are two Artificial Neural Network topologies – Feed Forward and Feed backward, where some feedback loops are allowed. Feed forward and feed backward ANN is shown in Figure 4 and Figure 5.

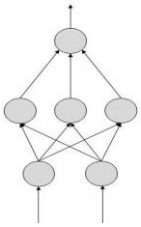

Fig. 4: Feed Forward ANN

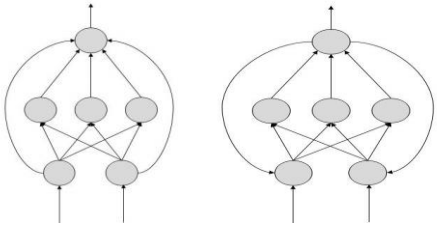

Fig. 5: Feedbacks ANN

4.3 Neural network equalizations of a processing program

Neural networks are being used for many purposes including the use of the same for software testing. A software testing model can be realized through a neural network. The variables that constitute a test case can be a sub-set of Input nodes situated in an input layer. Different types of outputs that should be obtained out of executing a test case can be situated as output nodes in the output layer.

A test case is generally generated keeping in view of testing a particular path of execution of existing in any program. A path of execution as such can be represented as a set of processing nodes where processing is undertaken in a sequential manner. Each processing node can be situated one after the other in adjacent hidden layers. At each of the processing node, processing is represented in terms of summation of the input variables which are multiplies with some kind of weights and the sub is added with a Bia and the resultant values are transformed to contain the processing output with a defined range. Thus any kind of processing can be represented in terms of summation equivalence.

An efficient algorithm is developed to design a neural network representing the testing requirements and the uses the neural network to determine the actual output that should generated out of executing a test case. It is possible to test whether a program is working properly by subjecting the test case to the program and the neural network and the comparing the both outputs. If the outputs are same then it can be determined that the program is properly developed or else it can be concluded that there exist a bug in a program path in which the test variables are situated.

4.4. Algorithm for generating test cases based on the input pairs, expected outputs and using the same for the development of neural network

NNBS_Algorithm (): -

Step 1: *Trace out the input variables from the testing specification stated for testing an embedded system*

Step 2: *Generate all pairs of input variables*

Step 3: *Trace out the conditions that exist among the input variables and Construct conditional matrix based on the requirements of the given embedded system.*

Step 4: *Eliminate the redundant rows in the conditional matrix that are transitive in nature*

Step 5: *Determine the number of hidden layers and processing nodes based on the optimized conditional matrix.*

Step 6: *Derive the formula for the processing nodes which are present in the hidden layer based on the conditional expression that must be fulfilled at that particular node.*

Step 7: *Construct neural network based on determined number of hidden layers and processing nodes.*

Step 8: *Learn the neural network through a recursive process such that the weights and bias are adjusted and find an activation function in such a way that a desired and expected outputs are evaluated through the processing nodes, weights of those nodes, and the transformation function*

Step 9: *The paths contained in the neural networks are enumerated which are typical test cases that must be tested. The variables that are situated on the path together formulate the test case.*

Step 10: *The software under test is tested by subjecting the test case both to the software and the neural network through selection of most appropriate values that must represent the variables. The outputs obtained by both the methods are compared to determine whether the processing of the test case by the software has gone through or failed. If the outputs are not same, then it can be construed that the path of the software containing the input variable has a bug in it.*

5. Neural Network Based Strategy (NNBS) for Generating Pairwise Test cases

The algorithm stated above is applied on to the pilot project is presented in section 4. The experimental results obtained through application of the algorithm to the pilot project are presented below.

Step-1: *Trace out the input variables from the testing specification stated for testing an embedded system. The testing requirements have been presented in Table 1. Input variables related to the pilot project have been traced out thorough analysis of the requirement specification of the pilot project. The Input variables traced out are shown in Table 2.*

Table 2: Input Variables Traced out of Test specification of the Pilot Project

S. No	Input Variable	Description
1.	I1	Temperature value of input1
2.	I2	Temperature value of input2
3.	I3	Temperature value of input3
4.	I4	Temperature value of input4
5.	R1	Reference temperature for the input 1
6.	R2	Reference temperature for the input 2
7.	R3	Reference temperature for the input 3
8.	R4	Reference temperature for the input 4

Step-2: *Generate all pairs of input variables. Figure 6 shows are all generated input pairs*

(I1, R1), (I1, R2), (I1, R3), (I2, R1), (I2, R2), (I2, R3), (I3, R1), (I3, R2), (I3, R3), (I1, I1), (I1, I2), (I1, I3), (I2, I1), (I2, I2), (I2, I3), (I3, I1), (I3, I2), (I3, I3), (I3, I4), (I4, I1), (I4, I2), (R1, R1), (R1, R2) (R1, R3), (R2, R1) (R2, R2) (R2, R3), (R3, R1) (R3, R2), (R3, R3).

Fig. 6: *Generated input pairs*

Step-3: *Trace out the conditions that exist among the input variables and construct conditional matrix that exhibits the relationships. Table 3 shows the relationships.*

Table 3: Conditional Matrix Table

S.No	Input pairs	Conditions		Output	
		Condition1	Condition2	Output1	Output2
1.	(I1,R1)	(I1<R1)	(I1>R1)	PUMP-1 OFF	PUMP-1 ON
2.	(I1,R2)	-	-	-	-
3.	(I1,R3)	-	-	-	-
4.	*(I2,R1)*	-	-	-	-
5.	*(I2,R2)*	(I2<R2)	(I2>R2)	PUMP-2 OFF	PUMP-2 ON
6.	*(I2,R3)*	-	-	-	-
7.	*(I3,R1)*	-	-	-	-
8.	*(I3,R2)*	-	-	-	-
9.	*(I3,R3)*	(I3<R3)	(I3>R3)	PUMP-3 OFF	PUMP-3 ON
10.	*(I1,I2)*	Abs (I1-I2)<2	Abs (I1-I2)>2	Buzzer-1 OFF	Buzzer-1 ON
11.	*(I1,I3)*	-	-	-	-
12.	*(I2,I1)*	Abs (I2-I1)<2	Abs (I1-I2)>2	Buzzer-1 OFF	Buzzer-1 ON
13.	*(I2,I3)*	Abs (I2-I3)<2	Abs (I1-I2)>2	Buzzer-2 OFF	Buzzer-2 ON
14.	*(I3,I1)*	-	-	-	-
15.	*(I3,I2)*	Abs (I3-I2)<2	Abs (I1-I2)>2	Buzzer-2 OFF	Buzzer-2 ON

Step-4: *Eliminate the redundant rows in the conditional matrix that are transitive in nature. Table 4 shows the pruned conditional matrix.*

Table 4: Optimized Conditional Matrix Table

S.No	Input pair	Conditions		Output	
		Condition 1	Condition 2	Output 1	Output 2
1.	*(I1,R1)*	(I1<R1)	(I1>R1)	PUMP-1 OFF	PUMP-1 ON
2.	*(I2,R2)*	(I2<R2)	(I2>R2)	PUMP-2 OFF	PUMP-2 ON
3.	*(I3,R3)*	(I3<R3)	(I3>R3)	PUMP- 3 OFF	PUM-P3 ON
4.	*(I1,I2)*	Abs (I1-I2)<2	Abs (I1-I2)>2	Buzzer-1 OFF	Buzzer-1 ON
5.	*(I2,I1)*	Abs (I2-I1)<2	Abs (I1-I2)>2	Buzzer-1 OFF	Buzzer-1 ON
6.	*(I2,I3)*	Abs (I2-I3)<2	Abs (I1-I2)>2	Buzzer-2 OFF	Buzzer-2 ON
7.	*(I3,I2)*	Abs (I3-I2)<2	Abs (I1-I2)>2	Buzzer-2 OFF	Buzzer-2 ON

Step-5: *Determine the number of nodes in the input and output layers and the number of processing nodes in terms of number of hidden layers and the number of processing nodes contained in each of the hidden layer.*

From the conditional matrix it can be seen that there are 8 input nodes and 8 outputs nodes, two hidden layers and in each hidden layer there 8 processing nodes.

Since two conditions are needed according to the TMCNRS, it needs two hidden layers in order to construct a neural network.

Step-6: *Derive the formula for the processing nodes which are present in the hidden layer.*

Step-6(a): *The computation condition of processing nodes for the first hidden layer for the above TMCNRS system in case of buzzer is given by the equation 1.*

$$\sum Abs\ (Ti*w_i - Ti+1*w_{i+1}) + b > 2 \text{ for } i=1 \text{ to } n \text{ with increment of } i++ \quad (1)$$

In the above equation, Ti is the temperature values (T1, T2…Tn) and w_i is the weights ($w_1, w_2 \ldots w_n$).

Step-6(b): *The Computation condition for the above TMCNRS system in case of pump is given by the equation 2.*

$$\sum (Ti*w_j + b) > (Ref_i*w_k) \quad \text{for } i=1,j=1,k=1 \text{ to } n \text{ with increment of } i++,j=j+2,k=k+2 \quad (2)$$

In the above equation, Ti is the temperature values (T1, T2…Tn) and w_i is the weights ($w_1, w_2 \ldots w_n$) and Ref_i are the reference temperatures ($Ref_1, Ref_2 \ldots Ref_n$)

According to the TMCNRS, if Abs (T1-T2) +b > 2 then corresponding buzzers is to be in ON state otherwise it is in OFF state. Similarly, if the T1>Ref1 then corresponding pump is in ON state otherwise it is in OFF State.

Step-7: *The weights, bias and activation function are adjusted in such a way that to achieve the required output based on requirements specified by the system.*

For the given system i.e. TMCNRS, all the weights are fixed to be 1 and -1 and bias as 1.

Step-8: *A Neural network is constructed based on determined number of hidden layers and processing nodes. Figure 7 shows the designed neural network.*

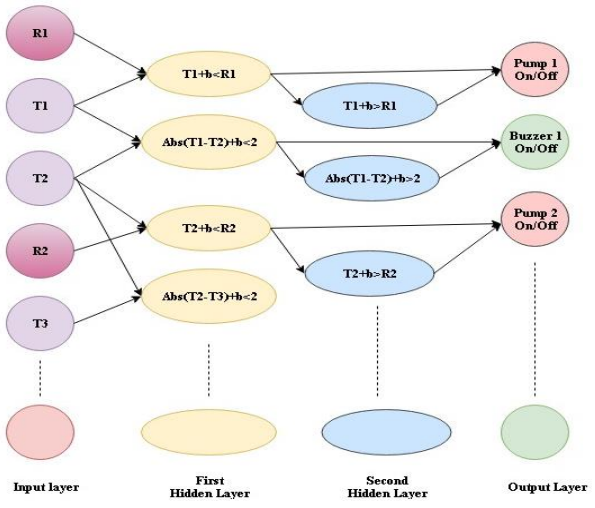

Fig.7: Neural Network relating to TNCNRS

Step-9: *The paths contained in the neural network are enumerated. Table 5 shows 6 paths that are enumerated from the neural networks.*

Table 5: Paths Contained Within the Neural Networks

Path Number	Input variable-1	Input Variable-2	Output Variable	Output Variable-Value
1.	T1	R1	PUMP-1	ON
2.	T1	R1	PUMP-1	OFF
3.	T2	R2	PUMP-2	ON
4.	T2	R2	PUMP-2	OFF
5.	T1	T2	BUZZER-1	ON
6.	T1	T2	BUZZER-1	OFF

Step 10: *One can choose 6 Test cases representing each of the paths contained in the neural network. Choice of proper values to the input variables, one can know the expected output value form the neural network by subjecting the chosen values to the input variables. Table 6 shows the generated test cases and the output generated by the neural network.*

Table 6: Generated Test cases through test paths in the Neural Network

Test case Number	T1	T2	R1	R2	Output generated from the Neural Network
1.	28	-	31	-	PUMP-1 OFF
2.	32	-	31	-	PUMP-1 ON
3.	-	28	-	32	PUMP-2 OFF
4.	-	33	-	32	PUMP-2 ON
5.	28	29	-	-	Buzzer-1 OFF
6.	28	31	-	-	Buzzer-1 ON

Step 11: *The test cases generated as shown in Table 6 are subjected to the firmware and the output generated by the firmware is shown in Table 7.*

Table 7: Firmware Test Results

Test case Number	T1	T2	R1	R2	Output generated through Neural network	Output generated by Firmware
1.	28	-	31		PUMP-1 OFF	PUMP-1 OFF
2.	32	-	31		PUMP-1 ON	PUMP-1 OFF
3.	-	28	-	31	PUMP-2 OFF	PUMP-2 OFF
4.	-	32	-	31	PUMP-2 ON	PUMP-2 ON
5.	28	29	-	-	Buzzer-1 OFF	Buzzer-1 OFF
6.	28	31	-	-	Buzzer-1 ON	Buzzer-1 ON

It can be seen that the output generated by the firmware differs from that of the output generated by neural network in respect of test case-2 indicating a bug contained in the program path-2.

6. Experimental Results

A tool called NNTCG (Neural Network Test Case Generator) has been presented based on NNBS. The generated test suite for TMCNRS with the configuration 4-sensors, 4-pumps and 3-buzzers is shown in Table 8. The test suites included all test cases that are essential to carry out testing of the input domain of TMCNRS.

T1, T2...Tn represent the temperatures sensed by the corresponding sensors, the reference temperatures for which are 30, 32, and so on for the respective temperatures.

The test suites sizes generated by AETG, IPO and NNBS for different criteria are shown in the Table 9 for comparative study.

By inspection, the results show that the NNBS have produced optimal results when compared to the other techniques like AETG, IPO etc.

Table 8: Test suite generated from Input Domain of TMCRS using NNBS

Test case No.	Input Vector				Expected Output	
	Temp1 (Ref. Temp= 30)	*Temp 2 (Ref. Temp= 32)*	*Temp 3 (Ref. Temp= 34)*	*Temp4 (Ref. Temp= 36)*	**PSC** (Binary)	**BSC** (Binary)
1.	29	31	33	35	0000	000
2.	29	32	33	36	0000	101
3.	29	33	33	37	0101	101
4.	30	31	34	35	0000	010
5.	30	32	34	36	0000	000
6.	30	33	34	37	0101	101
7.	31	31	35	35	1010	010
8.	31	32	35	36	1010	010
9.	31	33	35	37	1111	000

Table 9: Sizes of Test suite generated by AETG, IPO and NNBS

System	S1	S2	S3	S4	S5	S6
AETG	11	17	35	25	12	193
Pair wise (IPO)	9	17	34	26	15	212
NNBS	9	9	16	12	4	100

S1: 4 (3-value parameters),
S2: 13 (3-value parameters),
S3: 61 parameters (15 (4- value parameters), 17 (3- value parameters), 29 (2- value parameters)),
S4: 75 parameters (1 (4- value parameters), 39 (3- value parameters), 35 (2-value parameters)),
S5: 100 (2- value parameters),
S6: 20 (10- value parameters)

7. Conclusion and Future Enhancements

A Neural Network Based Strategy (NNBS) is developed to automatically create combinatorial test cases for testing an embedded System. The above technique yields best result with high efficiency. This work can be widened to build up t-wise combinatorial testing where t stands for 3, 4 and so on. Neural network is a fantastic replica method which can be used to model the input variables and output elements by modifying comprehensive connectivity between them.

References

[1] C. Anderson, A. Von Mayrhauser, R. Mraz (1995), "On the use of neural networks to guide software testing activities," Proceedings of International Test Conference, pp.720-729.

[2] Cohen D.M., D.M., Dalal, S. R. (1996), "Method and system for automatically generating efficient test cases for systems having interacting elements," patent, 1996.

[3] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G. C. Patton (1997), "The AETG System: An Approach to Testing Based on Combinatorial Design", IEEE Transactions on Software Engineering, Vol. 23, No. 7, pp. no.437-443.

[4] Yu Lei and K.C. Tai (1998), "In-parameter-order: a test generation strategy for pair wise testing," Proceedings of Third IEEE International High-Assurance Systems Engineering Symposium, 1998, pp. 254-261.

[5] Lei, Y and Tai, K. C. (2002),"A Test Generating Strategy for Pairwise Testing", IEEE Transactions on Software Engineering.

[6] S.A. Ghazi and M.A. Ahmed (2003), "Pair-wise test coverage using genetic algorithms", CEC, The 2003 Congress on Evolutionary Computation, Vol. 2, pp. 1420-1424.

[7] Bryce R.C, & Colbourn, C. J. (2007), "The Density Algorithm for Pair Wise Interaction Testing", Journal of Software: Testing, Verification and Reliability.

[8] Kewen Li and Zhixia Yang (2008), "Generating Method of Pair-Wise Covering Test Data Based on ACO," ETT and GRS, 2008 IEEE International Workshop on Geoscience and Remote Sensing

and International Workshop on Education Technology and Training, Vol.2, pp. 776-779.
[9] Lilan Wu, Bo Liu, Yi Jin, Xiaoyao Xie (2008), "Using back-propagation neural networks for functional software testing," 2nd International Conference on Anti-counterfeiting, Security and Identification, ASID, pp.272-275.
[10] Yogesh Singh, Arvinder Kaur, Ruchika Malhotra (2008), "Predicting Testing Effort using Artificial Neural Network", Proceedings of the World Congress on Engineering and Computer Science(WCECS).
[11] R. Kuhn, Yu Lei and Raghu Kacker (2008), "Practical Combinatorial Testing: beyond Pair wise", IEEE Computer Society - IT Professional, Vol.10, No. 3.
[12] Xiang Chen, Qing Gu, Xin Zhang and Daoxu Chen (2009), "Building Prioritized Pairwise Interaction Test Suites with Ant Colony Optimization," QSIC, 9th International Conference on Quality Software, pp. 347-352.
[13] J.D. McCaffrey (2009), "Generation of pairwise test sets using a simulated bee colony algorithm," IRI '09, IEEE International Conference on Information Reuse & Integration, pp. 115-119.
[14] James D. McCaffrey (2009), "Generation of Pairwise Test Sets using a Genetic Algorithm", 33rd Annual IEEE International Computer Software and Applications Conference.
[15] Rick Kuhn and Raghu Kacker, Yu Lei and Justin Hunter, "Combinatorial Software Testing", IEEE, 0018-9162, 2009.
[16] Xiang Chen, Qing Gu, Jingxian Qi and Daoxu Chen (2010), "Applying Particle Swarm Optimization to Pairwise Testing", COMPSAC, 2010 IEEE 34th Annual Computer Software and Applications Conference, pp.107-116.
[17] D. Richard Kuhn, Raghu N. Kacker and Yu Lei (2010), "Practical combinatorial testing", NIST Special Publication.
[18] Chandra Prakash Vudatha, Dr. Sastry KR Jammalamadaka, Bala Krishna Kamesh Duvvuri, Dr. Reddy L.S.S (2011), "Automated Generation of Test Cases from Output Domain of an Embedded System using Genetic Algorithms", Proceedings of the 3rd International Conference on Electronics Computer Technology (ICECT), 2011.
[19] Chandra Prakash Vudatha, Dr. Sastry KR Jammalamadaka, S. Nalliboena, B.K.K. Duvvuri, L.S.S. Reddy (2011), "Automated generation of test cases from output domain and critical regions of embedded systems using genetic algorithms," 2nd National Conference on Emerging Trends and Applications in Computer Science (NCETACS), pp.1-6.
[20] Kristina Smilgyte, Jovita Nenortaite (2011), "Artificial Neural Networks Application in Software Testing Selection Method", Springer Link Lecture notes, Hybrid Artificial Intelligent Systems, Lecture Notes in Computer Science, Vol. 6678, pp. 247-254.
[21] Manisha Patil and P.J. Nikumbh, "Pair-wise Testing Using Simulated Annealing", Published by Elsevier Ltd, 2012.
[22] Priti Bansal, Sangeeta Sabharwal, Shreya Malik, Vikhyat Arora, and Vineet Kumar (2013), "An Approach to Test Set Generation for Pair-Wise Testing Using Genetic Algorithms," Search Based Software Engineering, vol. 8084, pp.294-299.
[23] R. Raju, P. Subhapriya (2013), "A Neural Network Approach for Randomized Unit Testing Based On Genetic Algorithm", International Journal of Engineering and Advanced Technology (IJEAT), Vol.2, Issue.No.3.
[24] Huayao Wu and Changhai Nie (2014), "An overview of search based combinatorial testing," In Proceedings of the 7th International Workshop on Search-Based Software Testing (SBST 2014). ACM, New York, NY, USA, 27-30.
[25] H. L. Zakaria and K. Z. Zamli (2015), "Migrating Birds Optimization based strategies for Pair wise testing," 9th Malaysian Software Engineering Conference (MySEC), Kuala Lumpur, pp. 19-24.
[26] R. Qi, Z. Wang, P. Ping and S. Li (2015), "A hybrid optimization algorithm for pair wise test suite generation," 2015 IEEE International Conference on Information and Automation, Lijiang, pp. 3062-3067.
[27] A. Yamada, A. Biere, C. Artho, T. Kitamura and E. H. Choi (2016), "Greedy combinatorial test case generation using unsatisfiable cores," In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), Singapore, pp. 614-624.