ISSN 2347 - 3983
Volume 7, No. 11 November 2019
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter047112019.pdf
https://doi.org/10.30534/ijeter/2019/047112019

Generating Test cases for Testing Embedded Systems using Combinatorial Techniques and Neural Networks based Learning Model

Dr. Sasi Bhanu[1] J, Dr. Baswaraj D[2], Sunitha Devi Bigul[3], Dr. JKR Sastry[4]
[1, 2, 3] CMR Institute of Technology, kandlakoya, Medchal, Hyderabad, bhanukamesh1@gmail.com
[4]Koneru Lakshmaiah Education Foundation University, Vaddeswaram, Guntur District, drsastry@kluniversity.in

ABSTRACT

Embedded Systems must be bug-free. Comprehensive Testing of embedded systems is required to eliminate the risk and reduce Time and also to improve the performance of the embedded systems. Tools have been in existence for undertaking different kinds of Testing that include integraTion and system Testing. Embedded systems must be tested considering different perspecTives that include input, output, mulTi-input, mulTi-output, and input-output relaTionships. No tool is in existence for Testing the embedded systems, both the information and output domain. There is a need to learn a model represenTing the embedded systems, and the output generated out of the model must tally with the expected production to guarantee that the embedded system is tested comprehensively considering the input and output domain.

In this paper, a method has been presented that learns the embedded system through building the Neural network and the correctness or exactness of the model through comparing the outputs generated through Neural network with the outputs collected through manual Testing. The paths exisTing in the firmware that does not yield the expected output are determined and listed, so that code relaTing to those paths is corrected. The Testing method proposed in this paper covers the Input-Output perspecTive of the embedded system

Key words: Combinatorial methods, testing embedded systems, Test case generation, Neural Networks, Input-output domain

1. INTRODUCTION

Testing an embedded system is required for evaluaTing firmware and all the components that comprise the firmware — testing of the integrated system done for assessing the performance, robustness, and especially the behavior. Behavior assessment of the embedded systems is most important, and the embedded systems generally built for adapTive and real-Time systems. Testing of embedded systems must be carried considering verifiable methods. SomeTimes Testing must be carried considering different perspectives to confirm that an embedded system is designed and developed to meet all kind of relaTionships that exists among inputs and outputs.

Combinatorial Testing methods are in use for Testing conventional software systems. The arrangements are designed focusing on the selecTion of combinaTion of inputs that cover all the paths exisTing in the software. Embedded systems must be tested considering hardware, software and both. The behavior of hardware or software based on the inputs provided to those elements. Combinatorial methods have been proved to be efficient that fewer test cases generated that exhausTively cover the Testing of loaded software. Combinatorial methods are in use for Testing GUI, system configuraTions, protocols, web forms, loaded software, and the like.

Embedded systems are in use slowly in many fields, including automobile, health, airspace, networking, cell phones, home automation systems, and the proper functioning of these systems is essential that the users of those devices are confident of using those devices.

Embedded systems must be tested, considering both hardware and software. 80% of the Testing relates to Testing software. SomeTimes nonfunctioning of the hardware can also be verified through Testing software components that relate to hardware. Apart from testing of hardware and functioning of the software, it is also necessary to check the occurrence of the signals relating to the pattern of existence, sequencing, Timing, and validity. The embedded systems must test functions, end-to-end processing, etc.

The designers of an embedded system face a challenge in designing and developing test cases. It is challenging to generate test cases that are suitable for undertaking different kinds of Testing to be carried that include load, functions, paths, structure, and nonfunctional requirements such as response time and throughput. Testing of the embedded systems must also be undertaken considering different constraints imposed on the system.

Neural networks are built using the relaTionships between the inputs in a way the relaTionships depict the logic built into a software system. The model is built using three layers that include an input layer, output layer, and many hidden layers. Neural networks can be used to generate fewer test

cases that are sufficient to test a system exhausTively. The constraints imposed on the input variables also are imposed while considering the relaTionships among the variables.

Embedded systems are to be tested considering input, output, mulTi-output, mulTi-input, and input-output domains. Neural networks are best suited for Testing the embedded systems as underlying principles match the Input-output domain of the embedded systems.
In this paper, a method presented that uses combinatorial methods and neural networks using which test cases required for Testing embedded systems considering hardware, software, and both generated. Comparative analysis has performed that show how the proposed method produces minimal test cases when compared other exisTing methods.

2. LITERATURE SURVEY

Several applications of different types are being tested using combinatorial methods. The minimum cost incurred for Testing a system using combinatorial methods. Minimal Time required for undertaking Testing of a system using combinatorial methods. Many articles have been published in the literature, presenting the way the Testing of convenTional systems can be carried using combinatorial methods. Several approaches and methods published in the literature relating to Testing of convenTional loaded system, but very few have attempted to present the way embedded systems can be tested using the combinatorial methods.

Many tools are in existence for generating the test cases, but no tools, as such, are in reality for undertaking actual Testing using the generated test cases. No one guarantee that every aspect of a system will be thoroughly tested using the generated test cases. C. Anderson et al. [1] have asserted that one should learn a model based on facts observed out of the behavior of a system. Historical evidence collected out of Testing a system. Models help a lot in generating the requisite number of test cases required for Testing a system thoroughly and comprehensively. The authors have used a neural network-based model for prototyping system and then use the neural network for generating the test cases.

Cohen D. M. et al., [2] has used some determinisTic and random procedures for generating the test cases using a Table into which the relaTionships between the input variables entered. The table filled with data related to the relaTionships that exist among the input variables. The relaTionships between the variables generated through enumeraTion of the rows contained in the table.

D. M. Cohen et al., [3] have presented a method that considers selecTion a set of variables at a Time-based fulfillment of specific conditions and then generates test cases based on all combinations of the selected set of variables. The number of test cases created grows logarithmically, and it becomes infeasible to process those test cases or find the suitability of the same. The method proposed by them becomes infeasible when the numbers of input variables are more than 10.

The input domain of any system comprises of input variables. Input variables, when ordered or sequence based on some criteria, someTimes lead to the generation of most appropriate test cases. Yu Lei et al. [4] have considered all possible domain values of each variable in the domain. They [Lei Y et al.] [5] have further considered that every pair of input variables must be present at least in one test case.

Users have no idea about the wat the code is written and, therefore, not in a position to judge the combinaTion of test cases to be considered that help Testing the system thoroughly. The only option for the user is to use the combination of the input variables and the kind of output expected out of using the input combinaTions as test variables. It is possible to generate adequate test cases if the numbers of input variables are minimal in number. The numbers of test configurations to be used are minimal in number when the input variables considered are few. Genetic algorithms tried by S.A. Ghazi et al. [6] for generating test cases from few test case seeds such that the test cases made will cover most of the source code. They have used upper limit constraint while selecTing input variables.

Many methods presented in the literature, the most important among them being IPO, Annealing, AETG, TCG, TConfig for generaTing the test cases required for Testing convenTional loaded systems. Combinatorial methods and heurisTic search methods used combined for generating an opTimum number of test cases. Greedy methods presented leads to fewer test cases. Test cases are made as fast as possible using greedy methods, but there is no control over the number of test cases made, or there is no guarantee that the generated test cases will cover all the perspecTives of a system. The size of test cases made is significant, and performance in terms of the extent of Testing carried is also not guaranteed. Bryce R. R. et al., [7] have addressed the issue of generaTing large size of test cases and also ensure the performance related to undertaking the Testing of any convenTional loaded system.

Changes to software will happen as the customer requirements keep changing. Testing of the software is done every Times changes takes place to the software. Test cases are generated for Testing of the software initially, and then out of these test cases are selected to test the affected modifications on the software. The generation of test cases based on the input pairs is a kind of NP problem. Such sort of NP problems can only be solved using algebraic, heurisTics or greedy methods. Kewen Li et al. [8] have presented the Ant colony method that generates few test cases that cover maximum input variables. Regression Testing would become lot easier when the numbers of input variables used are less in number to create the test cases.

Software Testing is being done these days through learning a Neural network model; however, the way the neural network learned differed in many ways. Every neural network designed based on the inputs fed and the kind of outputs expected out of the system. A method called Back-propagation is used by Wu et al. [9] for adjusting the coefficients which are the most critical model parameters. They have presented the way software Testing can be carried using the learned neural network. They have experimented Testing an exisTing software and have shown that 90% of the code gets tested using test cases generated by the method proposed by them.

Yogesh Singh et al. [10] considered each node within a neural network as an object modeled as a process of computaTion. The NN model presented by them used for assessing the quality of software. The authors have handled test cases of specific software as data for undertaking Testing of the software. They have also evaluated the level of effort required for undertaking Testing using the model presented by them.

Combinatorial methods involve combinaTion of input variables or the inter-relaTions between the variables for the generation of test cases. However, the test cases are not good enough that rare faults that occur, especially within embedded systems tested as it is quite difficult to simulate the occurrence of such. Many tools exist that generate test cases covering at least one existence of a test pair. R. Kuhn et al., [11] have considered the concept of covering arrays that places the input variables into different rows of a collection. Various models built depicTing either the functionality of a system or test scenarios. The models are checked to see if the models cover all input pairs. The authors have combined the covering arrays and modeling checking methods to generate test cases.

Once the software developed, the users start interacTing using the GUI designed for the system. The users will get idea on the type of test cases required, the sequence in which the test cases inputted, and the order in which the test cases administered for Testing the system. The density and the domain values of the variable are most important aspects considered while generating the test cases. Xiang Chen et al. [12] have studied the density of variables as weighted factors and also the covering array method for making the test cases while ate the same Time use the Ant Colony OpTimizaTion method for opTimizing the process used for generation of test cases.

Exhaustive testing of the embedded system is Time-consuming and is also quite expensive. Thorough Testing requires consideraTion of all combinaTion of input variables. Input-pair based test case generation method produces the least number of test cases than considering all the combinaTion of input variables while at the same Time exhausTing Testing of the enTire system carried. J.D. McCaffrey et al. [13] have used bee colony opTimizaTion while considering input-pairs for the generation of test cases. Their methods generated few test cases but have taken more time to create those test cases.

GeneraTing test cases are complex, considering all input pairs and the domain values of the input variables that form the input pairs. Several vectors with each vector containing specific values selected out of the domain of input variables get created. Many such vectors get created as many chosen values from the realm of each variable. The problems of dealing with too many variables are like the NP-hard problem. James D. McCaffrey et al. [14] used a genetic algorithm to generate test cases considering seed test cases that involve few input variables. As the test case generation gets exhausted more input variables get added some variables are dropped. At any instance fewer input-variables are on scope making it easy to generate the test cases. While this approach creates all the test cases required, it is quite time-consuming.

A book published by Rick Kuhn et al. [15] covers the essential aspects relating to combinatorial Testing. They have emphasized the requirements of real-world Testing that focus on the cost of Testing. They have presented advanced approaches to software Testing using combinatorial Testing.

It becomes complex to test a system when many test cases are generated using a few input variables and several test cases. Particle swarm opTimizaTion technique is a Metaheuristic search technique used by Xiang Chen et al. [16] for generating the test cases based on the input variable pairs such that all paths exisTing in the software tested. A single test case created by them represents a test case created by them. The test case made representing a way is opTimum as it covers all the edge conditions required for undertaking the Testing.

A fault can occur within software due to the existence of variables. The failures also can happen when a combination of input variables used. A review of different tools, concepts, and methods utilized for generating the test cases is by D. Richard Kuhn et al. [17]. They have shown drawbacks existing in each of the methods.

Most research focusses on in input domain for generaTing the test case. These approaches covered just one face of the Testing of the system. Many other facets considered for the generation of test cases based on the output domain, mulTi-input domain, mulTi-output domain, and input-output domain. The number of output variables is less compared to number of input variables in respecTive of some systems; in such cases consideraTion of output domain only becomes critical.

Testing in and around the CriTicality regions, exhaustively, is essential when it comes to the embedded systems, while random Testing is sufficient in the other areas. Chandra Prakash V et al., [18] [19] have used the output domain of an

embedded system and the criTicality regions to generate test cases using genetic algorithms. The seed test cases are those related to criTicality regions of the embedded systems.

Some organizaTions are using automated tools for generating test cases and also automate the very Testing of the embedded systems. Kristina Smilgyte et al. [20] have developed a tool that uses a neural network as the model for generating the test cases. They have presented the effort and Time required for undertaking the Testing of a system using the tool.

The main objecTive of Testing a system is to undertake Testing of the system using few test cases so that the Time and money required for undertaking Testing of software is minimum — a genetic algorithm-based technique presented by Kristina Smilgyte et al. [20] and simulated annealing based technique presented by Patil et al. [21]. Both the methods reduced the number of test cases required for undertaking the Testing of a software-based system exhaustively.

The selection of the input variables is the most crucial aspect when it comes to the generation of test cases. One has to use certainly the optimization techniques for generating test cases through input pairs. Hamming distance between the input variables considered by Priti Bansal et al. [22], based on which the input variables get selected. The input variables which are close to each other based on the hamming distance are the best choice for selecTion so that these variables can be used further to generates test cases using combinatorial methods. The authors have developed an algorithm that finds the cross over points using which the individual test cases created.

Combinations of approaches are used by many authors to address the issue of Time, cost, and the use of a few test cases. Genetic Algorithms, random Testing is used by R. Raju et al. [23] for generation of test cases. They have emphasized that it is complicated and complex to test the interaction among the input variables based on the generated test cases. Hayao Wu et al. [24] presented detailed overview of Testing software based on the interaction between the input variables. They have also explained the use of search-based techniques for finding the faults that might happen due to interacTion among the variables.

H. L. Zakaria et al. [25] presented an advance search technique called MBO (MigraTing search opTimizaTion) for achieving the opTimizaTion of test case generation using input pairs. They have presented MBO, and an improved MBO called iMBO. The main focus of their method is to consider structures that are input pairs in the same neighborhood.

A method called hybrid optimization used by R. Qi et al. [26] is an enhancement of the purpose-built on a genetic algorithm combined with a hill-climbing algorithm. This approach is also proved to produce acceptable results, especially when specific systems are developed based on the concepts of hierarchical structuring.

Some test tuples get ignored when search methods used for selection of the Input variables. Some test tuples usually get overlooked when the Greedy method employed for searching. The greedy approach as such, ignores the constraints imposed while undertaking the search. Akihisa Yamada et al. [27] have improved the greedy way while imposing the test conditions and restrictions.

A number of approaches presented by Sastry et al., [28][29][30][31][32][33] [34][35][36][37][38][39][40] [41] [42][43][44][45] for Testing standalone and distributed embedded systems must of which focused on the way Testing of the embedded systems carried and also concentrate on the way test cases are generated. While these contributions focused on may perspecTives which are different from the perspecTives of Input-Output domain

3. PROTOTYPE MODEL

A prototype embedded application developed, which is meant for moniTion and control the temperatures within nuclear reactor tubes. The application developed with the principal undertaking the Testing using the Internetwork based combinatorial method proposed in this paper. The prototype used as an experimental model. Figure 1 shows the Hardware interconnecTivity diagram of the prototype model.

The nuclear reactor tubes fitted with temperature sensors for sensing the temperatures exisTing in the reactor tubes. The temperature sensor produces analog signals proportional to the temperatures sensed. The amplified analog signal is inputted to an A/D converter to transform the signals to digital data read by a program that is part of firmware running within a prototype model. The temperatures thus read are read within RAM of the Microcontroller based system.

Threshold values defined for each of the temperatures within the nuclear reactor tube and the temperatures maintained within the threshold value by controlling temperature through the injection of coolants using the pumps. The operator is alerted when tremendous various occur across the gradients of the temperatures exisTing in both the tubes. The pumps are controlled using relays. Power to the pump-fed when coolant is to be pumped into the reactor tube or otherwise the power to the pump is switched off through proper rely operaTion. The microcontroller based system is interfaced with a PC through an RS232C connecTion for feeding the threshold values, are used for controlling the temperatures within the nuclear reactor tubes — the temperatures within the nuclear reactors displayed on the LCD connected to the Microcontroller based system. The functional requirements of the prototype system shown in Table 1.

3.0 GENERATING TEST CASES THROUGH NUERAL NETWORKS

3.1 An overview on Nueral networks

A neural network is a directed graph having nodes that are Artificial neutrons. The nodes are interconnected using edges that have specific weights. The commencing nodes are called input nodes, and the terminating nodes are called output nodes. The intermittent nodes are processing nodes, which generally called the hidden nodes. The nodes included in several layers are structured. The first layer is called the input layer, and the last layer called the output layer; there can be many hidden layers situated between the input layer and the output layer. The typical layout of the network shown in figure 2, and the architecture of neural network shown in figure 3.

The inputs from the external world (Sensors, File Records, Manual Inputs, Patterns, and Vectors) fed as inputs to the nodes situated in the input layer. The input nodes mathematically represented as X(i) where i stands for n number of inputs. Every data is normalized to fit into a range — the inputs connected to the nodes situated in the first hidden layer — every connection between any pair of nodes assigned with weights. The problem is to find the values attached to the weights. The value of any node is summation of inputs multiplied by the respective weights. Bias is added to the summed value to bring the value into a specific range. The bios value is zero in the beginning.

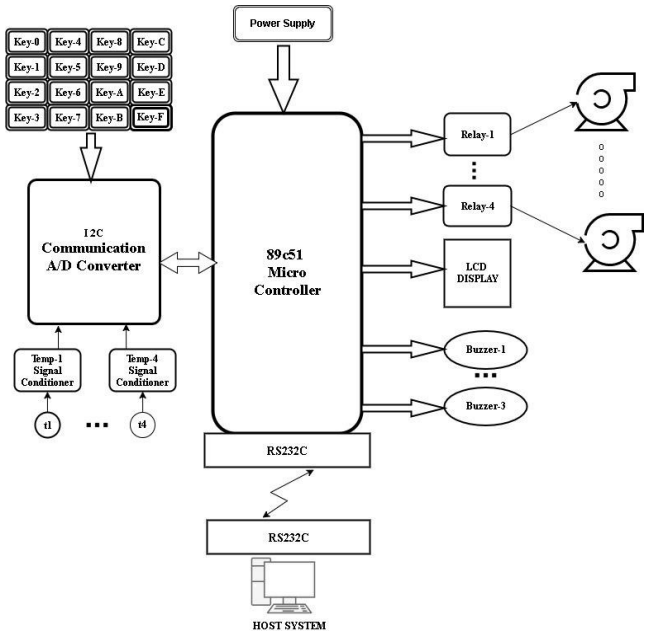


Figure 1: System IntegraTion Diagram of Prototype Model

Every summed up value subjected to an activation function such that summed value will be in acceptable limits. The primary purpose of activation function is to direct the summed up data to the expected output values. There are many types of activation functions that are either linear or non-linear. Sigmoidal, hyperbolic sigmoidal functions (Nonlinear), and Binary are some of the activation functions used. The output of an activation function is either 0 or 1 when the activation function is binary. The function related to a shaped curve 'S' represents Sigmoidal Hyperbolic. The activation function "Tan Hyperbolic" gives more accurate output estimation. The function defined in equation 1.

ff (x) = (1/1+ exp (-σx)) where σ—steepness parameter {1}

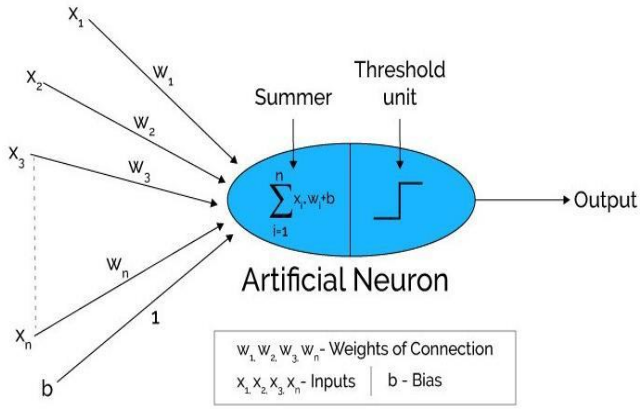


Figure 2: Layout of an Artificial Neural Network

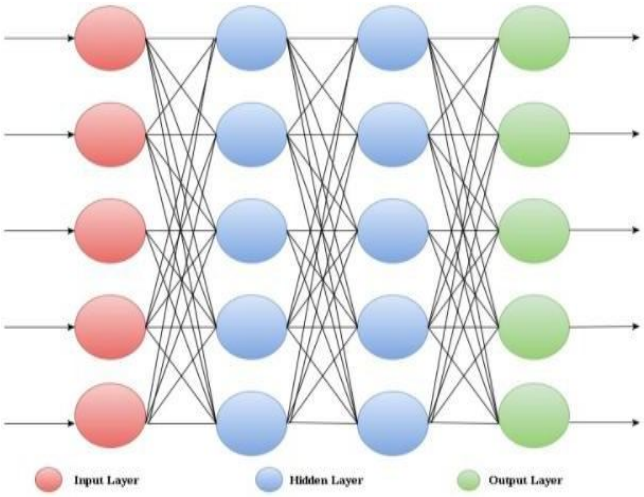


Figure 3: Neural Networks - Architetcure

Two different types of topologies used while developing a neural network, which include*s* – Feed Forward and Feeds backward, which considers a feedback loop to propagate the error at the output nodes, which occur when the actual output is different from the computed output. A sample Feedforward and feed backward ANN, shown in Figure 4 and Figure 5.

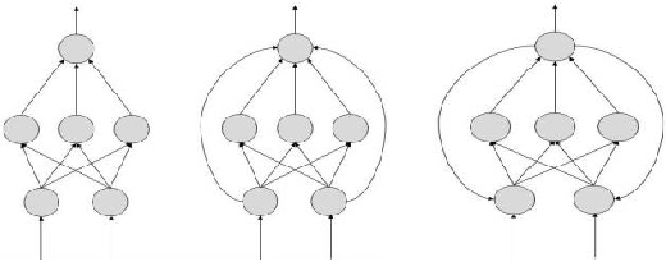


Figure 4 : Feed Forward ANN **Figure 5**: Feedbacks ANN

4.2 Neural network equalizations of a processing program

Neural networks are being used heavily for different purposes, including for software Testing. The input variables of a test case fed as inputs to the neural network. The test results are the outputs of a neural network. The hidden nodes are symbolic representation of processing that takes place within a program.

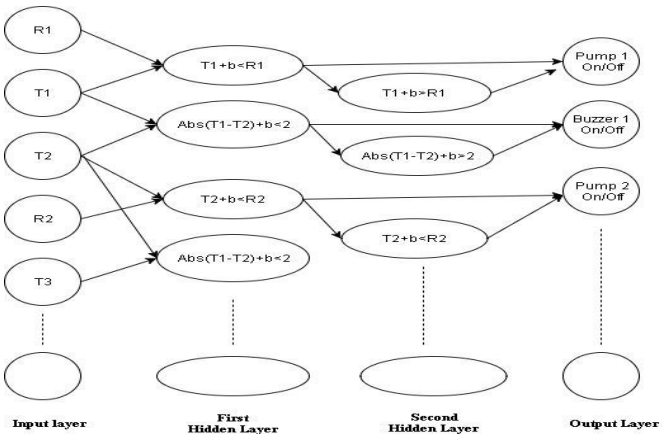
A test case meant for testing a specific path existing within a program. A test path represented as a sequence of processing nodes, which are the nodes situated in the hidden layers of the neural networks. The processing nodes situated in the adjacent hidden layers form a specific sequence. The processing at each of the nodes represented as summation of income inputs multiplied by the corresponding weights, and then some bios added. The resultant value is then transformed to get the test output value. Thus any internal processing coded into a program can be represented as a path in the neural network.

In this paper, an efficient algorithm is developed using which a neural network designed that represents a Testing model. The model thus used for undertaking Testing by feeding the data of test variables as input to the neural network and then immediately getting the expected output. The test case subjected to the program under teste and the production obtained from the program compared with the production received neural network. When the outputs are same, one can conclude that the neural network learned correctly. If there exists a difference between the program output and NN output then the difference is propagated in the feed backward direction, and the weights are adjusted accordingly. Several test cases tried before one can declare that the NN fully matured. Table 2 shows the steps for the algorithm**.**

Algorithm

Table 2 Algorithmic steps for constructing an NN and generation of test cases

Step Number	Step Execution
1.	Study Testing specification drafted for an embedded system and list down the Input Variables, or if the source code is in hand, parses the code to find the Input Variables.
2.	Order the variables based on the way they appear in a program
3.	Considering the list of ordered input variables, generate input variable pairs
4.	While parsing the program, find the conditions that exist in between Input variables, and generate the matrix based on those conditions
5.	Process the conditional matrix and remove redundant rows existing in the conditional matrix. The redundant rows are those that have input variables that are transitive.
6.	Process the conditional matrix to determine the total number of nodes, nodes exiting in the input layer and nodes existing in output layer and hidden layers existing in between output and Input layers
7.	Generate and Enumerate the formulas that connect the processing nodes situated in Hidden layers based on the number of incoming edges and outgoing edges. The relationships between the variables form the computation expressions
8.	With all the details about the inputs, outputs, connectivity among the nodes, develop neural networks and store the same as connected data structures that stores the Initial weights of the edges connecting the outgoing nodes
9.	Implement a recursive process that assumes an activation function and determines weights and bios such that the outputs generated matches with the expected output values. Different activation functions trained, and the function that create least variations selected such that model is full learned
10.	Enumerate all the paths contained in the neural network learned. Each way containing specific input variables and the output variables forms the test case — the data values calculated for the intermittent variables based on the expressions. The test data is the values assigned to the variables contained in a path within the neural network. The test data keeps varying based on the data assigned to the input variables, and when the inputs subjected to the neural networks, the expected outputs generated. All the generated test cases stored in an array
11.	The test cases derived out of the neural network subjected to the actual software, and test results obtained. The software produced test results compared with the test results obtained through neural networks. If the outputs are the same then one can conclude that the test succeeded or else test failed leading to the correction of the code related to the test path.



4.3 Step by step execution of Algorithm using prototype

Figure 6: Generated neural Network

4.4 model

The algorithm applied to the prototype project described above. The experimental results obtained while using the algorithm step by step presented below.

Step-1:

Determine the input variables, either parsing the sources code or tracing the functional / Testing specification of the application concerned, and in this case, it is the prototype project. The technical requirements of the Prototype project shown in Table-1. Table 3 shows the list of input variables traced out through the analysis of functional specification or processing of the source code.

Table 3: Variables traced in the Input domain

S. No	Input Variable	Description
1.	II1	Temp-1 data value
2.	II2	Temp-2 data value
3.	II3	Temp-3 data value
4.	II4	Temp-4 data value
5.	RR1	Value of Ref-1
6.	RR2	Value of Ref-2
7.	RR3	Value of Ref-3
8.	RR4	Value of Ref-4

Step-2
Process all the input variable pairs, as shown in Table 4.

Step-3

Develop a matrix that stores the conditions existing among the varaiables. The conditions that exist among the Input variables shown in Table 5.

Step-4:

Remove the redundant rows existing in the Conditional matric. Table 6 shows the pruned conditional matrix.

Step-5:

Process the conditional matrix and find the number of nodes in input, output, and hidden and all to see the number of hidden layers. One can note from the conditional matrix e 8 Input, 8 outputs and Inputs, two hidden layers, 8 nodes in each hidden layer are in existence. Two hidden layers exist to model two specific conditions.

Step-6:

Formulas to be implemented within the processing nodes that are situated in the hidden layer are determined.
Step-6(a):

The computational expressions processed @ hidden nodes within hidden layer-1 shown in equation 2. These expressions are related to processing the buzzer outputs

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

In the above equation, Ti is the temperature values (TT1, TT2…Tn), and wwi is the weights (w1, w2... wn).

Step-6(b):

The computational expressions processed @ hidden nodes within hidden layer-2 shown in equation 3. These expressions are related to processing the pumps outputs

$$\sum \text{(TTi*wwj+b) >(Refi*wwk) for i=1,j=1,k=1 to n with increment of i++,j=j+2,k=k+2} \quad (3)$$

TTi are the temperature values (TT1, TT2…Tn) and wwi are the weights (w1, w2…. wn), and Refi is the reference temperatures (Ref1, Ref2…. Refn). According to the Porotype project, if Abs (TT1 - TT2) + b > 2, then corresponding buzzers are to be in ON state; otherwise, the buzzers put at OFF state. Similarly, if the TT1 > Ref1, then the corresponding pump is in ON state; otherwise, set in OFF state.

Step-7:

A recursive process is implemented to arraive at the values of Edge weigts, and Bios and also determine most suitable transformation function. Intiatially the bios value is fixed at -1 and the weights fixed at value 1.
Step-8:

Based on the design parameters computed a nueral network is constructed which is shown in Figure 6.

Step-9

Test cases are generated by tracing the paths contained in the Nueral network. The paths enumerated with the help of constructed nuerl netork are shown in Table 7.

Step 10:

Any number of test cases can be generated considering each path by picking values from the domin of the Input variables. Some of the test cases selected by selecting the values for the input varaibles are shown in Table 8.

Step 11

The test cases generated, as shown in Table 6 subjected to the firmware and the output generated by the firmware shown in **Table 9.**
One can check whether the output produced by NN and the Firmware are sihmialr or different. If different the test concluded as failed.

4.5 Experimental Results

A Prototype system developed using 4-sensors, 4-pumps, and 3- buzzers tested and test cases generated using the algorithm are shown in Table 10. TT1, TT2...Tn represents 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 size generated by AETG, IPO, and NNBS for different criteria shown in Table 11. By inspection, the results show that the NNBS has produced optimal results when compared to other techniques like AETG, IPO, etc.

5 CONCLUSION AND FUTURE ENHANCEMENTS

Embedded system tested to find the system working different perspectives that include Input domain, output domain, multi-input, multi-output, and Input-Output domain. The input-output domain reflects the end-to-end testing of the embedded systems. Numeral networks also used when end-end processing depicted effectively.

Combinatorial methods have been provided to be quite useful in generating a few test cases that can help to complete Testing in the least time and cost. Considering of neural networks and combinatorial methods helps is creating the few test cases used for Testing using the model as well as the software. One can extend the model for considering 3-way or n-way combinations of the input variables.

In this paper, a novel way of constructing the neural network is presented based on the relationships that exist among the input variables to model the processing nodes situated in the hidden layers

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. https://doi.org/10.1109/TEST.1995.529902

[2] Cohen DM., Dalal SR., Method and system for automatically generating efficient test cases for systems having interacting elements, 1996 Patent.

[3] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G. C. PaTon (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. https://doi.org/10.1109/32.605761

[4] Yu Lei and K.C. Tai (1998), "In-parameter-order: a test generation strategy for pairwise Testing," Proceedings of Third IEEE International High-Assurance Systems Engineering Symposium, 1998, pp. 254-261. https://doi.org/10.1109/HASE.1998.731623

[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., & Colbourn, C. J. (2007), "The Density Algorithm for Pairwise Interaction Testing," Journal of Software: Testing, Verification, and Reliability. https://doi.org/10.1002/stvr.365

[8] Kewen Li and Zhixia Yang (2008), "Generating Method of Pairwise Covering Test Data Based on ACO," ET 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 backpropagation 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), "Predict- ing Testing Effort using Artificial Neural Network," Proceedings of the World Congress on Engineering and Computer Science(WCECS).

[11] R. Kuhn, Yu Lei, Raghu Kacker, "Practical Combinatorial Testing: beyond Pairwise, IEEE Computer Society - IT Professional, Vol.10, No. 3, 2018 https://doi.org/10.1109/MITP.2008.54

[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. https://doi.org/10.1109/IRI.2009.5211598

[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. https://doi.org/10.1109/MC.2009.253

[16] Xiang Chen, Qing Gu, Jingxian Qi, Daoxu Chen, Applying Particle Swarm Optimization to Pairwise Testing, 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, Sastry KR Jammalamadaka, Bala Krishna Kamesh Duvvuri, Reddy LSS, 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, 2011.

[19] Chandra Prakash V., Sastry JKR., S. Nalliboena, BKK. Duvvuri, LSS. Reddy, 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, 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. https://doi.org/10.1007/978-3-642-21219-2_32

[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. https://doi.org/10.1145/2593833.2593839

[25] H. L. Zakaria and K. Z. Zamli (2015), "Migrating Birds Optimization-based strategies for Pairwise 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 pairwise 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. https://doi.org/10.1145/2970276.2970335

[28]. J. K. R. Sastry, M. Lakshmi Prasad, Testing embedded system through opTimal mining technique (OMT) based on mulTi-input domain, International Journal of Electrical and Computer Engineering (IJECE) Vol. 9, No. 3, pp. 2141~2151, 2019

[29]. J Sasi Bhanu, JKR Sastry, M Lakshmi Prasad, Testing Embedded Systems from Multi-Output Domain perspective, International Journal of Recent Technology and Engineering (IJRTE), Volume-8, Issue-1, pp. 3106-3113, 2019

[30]. K Chaitanya1, Dr. K Rajasekhra Rao2, Dr. JKR Sastry3, International Journal of Advanced Trends in Computer Science and Engineering, Volume 8, No.4, pp 1194-1227, 2019 https://doi.org/10.30534/ijatcse/2019/30842019

[31]. M. Lakshmi Prasad, A. Raja Sekhar Reddy, J.K.R. Sastry, GAPSO: Optimal Test Set Generator for Pairwise Testing, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-8 Issue-6, pp. 2346-2350, 2019

[32]. M. Lakshmi Prasad, Dr.J.K.R. Sastry, A Graph-Based Strategy (GBS) for Generating Test Cases Meant for Testing Embedded Systems Using Combinatorial Approaches, Jour of Adv Research in Dynamical & Control Systems, Vol. 10, pp. 314-324, 2018

[33]. Chaitanya Kilaru, Dr. JKR Sastry, Dr. K Raja Sekhara Rao, Testing distributed embedded systems through logic Analyzer, International Journal of Engineering & Technology, 7 (2.7) (2018) 297-302 https://doi.org/10.14419/ijet.v7i2.7.10601

[34]. Lakshmi Prasad Mudarakola1, J. K.R. Sastry, V. Chandra Prakash, Testing embedded systems using test cases generated through combinatorial

techniques, International Journal of Engineering & Technology, 7 (2.7) (2018) 146-158
https://doi.org/10.14419/ijet.v7i2.7.10282

[35]. Dr. J Sasi Bhanu, M. Lakshmi Prasad, Dr. J. K. R. Sastry, A Combinatorial Particle Swarm Optimization (PSO) Technique for Testing an Embedded System, Jour of Adv Research in Dynamical & Control Systems, Vol. 10, 07-Special Issue, pp. (321-336), 2018

[36]. M. Lakshmi Prasad1, Dr. JKR Sastry, Building Test Cases by Particle Swarm Optimization (PSO) For Multi-Output Domain Embedded Systems Using Combinatorial Techniques, Jour of Adv Research in Dynamical & Control Systems, Vol. 10, pp. 1221-1229, 2018

[37]. Dr. J Sasi Bhanu, M. Lakshmi Prasad, Dr. J. K. R. Sastry, Combinatorial Neural Network-Based a Testing of an Embedded System, Jour of Adv Research in Dynamical & Control Systems, Vol. 10, pp. 605-611, 2018

[38]. Dr. J Sasi Bhanu, M. Lakshmi Prasad, Dr. J. K. R. Sastry, Testing Embedded System through OpTimal Combinatorial Mining Technique, Jour of Adv Research in Dynamical & Control Systems, Vol. 10, pp. (337-354), 2018

[39]. Dr. J Sasi Bhanu, M. Lakshmi Prasad, Dr. J. K. R. Sastry, Testing Embedded Systems Using - A Graph-Based Combinatorial Method (GBCM), Jour of Adv Research in Dynamical & Control Systems, Vol. 10, pp. (355-375), 2018

[40]. J Sasi Bhanu, Y. Venkata Raghavarao and JKR Sastry, Testing through In-Circuit Emulators, the RS485 based Distributed Embedded System, Journal of Engineering and Applied Sciences 13 (8): pp. 1947-1962, 2018

[41]. *Dr. JKR Sastry, K Chaitanya,* An Effective Model for Testing Distributed Embedded Systems using Scaffolding Method, PONTE-International Journal of sciences and research, Volume 78, issue 8, pp. 2-22, 2017
https://doi.org/10.21506/j.ponte.2017.8.1

[42]. Dr. JKR Sastry**,** K Chaitanya, Dr. DBK Kamesh, An Efficient Method for Testing Distributed Embedded Systems using In-circuit Emulators, PONTE-InternaTional Journal of sciences and research, Volume 73, issue 7, pp. 390-422, 2017
https://doi.org/10.21506/j.ponte.2017.7.59

[43]. Dr. Sastry JKR, K Chaitanya, Dr. K. Rajasekhara Rao, Dr. DBK Kamesh, Testing distributed embedded systems through instruction set simulators**,** PONTE-International Journal of sciences and research, Volume 73, issue 7, pp. 353-382, 2017
https://doi.org/10.21506/j.ponte.2017.7.57

[44]. K. Chaitanya1, Sastry JKR, K. N. Sravani, D. Pavani Ramya, and K. Rajasekhara Rao, Testing distributed embedded systems using assert macros, ARPN Journal of Engineering and Applied Sciences, VOL. 12, NO. 9, pp. 3011-3022, 2017,

[45]. ChandraPrakash V, Sastry JKR, Sravani G, Manasa USL, Khyathi A, Harini A, Testing software through genetic algorithms – A survey, Journal of Advanced Research in Dynamical and Control Systems Vol. 9, 1607-1623 2017

Table 1: Prototype Model – requirement specification

S. No.	Functional requirements
Req.1	The threshold temperatures(Ref-1 and Ref-2) maintained within the reactor tubes are transmitted from HOST (PC) and stored within the RAM of Microcontroller
Req.2	The temperature (Temp-1) within the first reactor tube must be sensed within every 1o Microseconds and displayed on LCD and sent to PC which is called the HOST
Req.3	Pump-1 is made to be in on state by changing the state of the Relay-1 to which the Pump-1 connected when Temp-1 > Ref-1 or else Pump-1 is switched off
Req.4	The temperature (Temp-2) within the second reactor tube must be sensed within every 1o Microseconds and displayed on LCD and sent to PC which is called the HOST
Req.5	Pump-1 is made to be in on state by changing the state of the Relay-1 to which the Pump-1 connected when Temp-1 > Ref-1 or else Pump-1 is switched off
Req.6	If the temperature difference between Temp-1 and Temp-2 > 2 Degrees, then buzzer must be triggered, and else the buzzer id de-triggered.

Table 4: Generated Input Pairs

Pair num.	**Input pair**	**Pair num.**	**Input pair**	**Pair number**	**Input pair**
1	(11, RR1),	2	(II1, RR2)	3	(II1, RR3)
4	(II2,RR1)	5	(II3,RR2)	6	(II2,RR3)
7	(II3,RR1)	8	(II3,RR2)	9	(II3,RR3)
10	(II1,II1)	11	(II1, II2)	12	(II1, II3)
13	(II3,II1)	14	(II3, II2)	15	(II3, II3)
15	(II3,II4)	17	(II4, II1)	18	(II4, II2)
19	(RR1, RR1)	20	(RR1,RR2)	21	(RR1, RR3)
22	(RR2, RR1)	23	(RR2, RR2)	24	(RR2, RR3)
25	(RR3, RR1)	26	(RR3, RR2)	27	(RR3, RR3)

Table 5: Relationships among the Input variables

S. No	**Input Variable pair**	**Conditions**		**Output**	
		1st Condition	**2nd Condition**	**Output-1**	**Output-2**
1.	(II1, RR1)	(II1 < RR1)	(II1 > RR1)	PUMP-1-OFF	PUMP-1-ON
2.	(II1, RR2)	-	-	-	-
3.	(II1, RR3)	-	-	-	-
4.	*(II2, RR1)*	-	-	-	-
5.	*(II2, RR2)*	(II2 < RR2)	(II2 > RR2)	PUMP-2-OFF	PUMP-2-ON
6.	*(II2, RR3)*	-	-	-	-
7.	*(II3, RR1)*	-	-	-	-
8.	*(II3, RR2)*	-	-	-	-
9.	*(II3, RR3)*	(II3 < RR3)	(II3 > RR3)	PUMP-3-OFF	PUMP-3-ON
10.	*(II1, II2)*	Abs (II1 - II2) < 2	Abs (II1 - II2) > 2	Buzzer-1	Buzzer-1
11.	*(II1, II3)*	-	-	-	-
12.	*(II2, II1)*	Abs (II2 - II1) < 2	Abs (II1 - II2) > 2	Buzzer-1	Buzzer-1
13.	*(II2, II3)*	Abs (II2 - II3) < 2	Abs (II1-II2) > 2	Buzzer-2	Buzzer-2
14.	*(II3, II1)*	-	-	-	-
15.	*(II3, II2)*	Abs (II3 - II2) < 2	Abs (II1 - II2) > 2	Buzzer-2	Buzzer-2

Table 6: Pruned Conditional Matrix

S.No	Input variable pair	Conditions		Output	
		1st Condition	2nd Condition	1st Output	2nd Output
1.	*(II1, RR1)*	(II1 < RR1)	(II1 > RR1)	PUMP-1-OFF	PUMP-1-ON
2.	*(II2, RR2)*	(II2 < RR2)	(II2 > RR2)	PUMP-2-OFF	PUMP-2-ON
3.	*(II3, RR3)*	(II3 < RR3)	(II3 > RR3)	PUMP-3-OFF	PUM-P3-ON
4.	*(II1, II2)*	Abs (II1 - II2) < 2	Abs (II1 - II2) > 2	1st Buzzer OFF	1st Buzzer ON
5.	*(II2, II1)*	Abs (II2 - II1) < 2	Abs (II1 - II2) > 2	1st Buzzer OFF	1st Buzzer ON
6.	*(II2, II3)*	Abs (II2 - II3) < 2	Abs (II1 - II2 > 2	1st Buzzer OFF	2nd Buzzer ON
7.	*(II3, II2)*	Abs (II3 - II2) < 2	Abs (I 1- II2) > 2	1st Buzzer OFF	2nd Buzzer ON

Table 7: Paths Contained Within the Neural Networks

Path Number	Input variable-1	Input Variable-2	Output Variable	Output Variable- value
1.	TT1	RR1	PUMP-1	ON
2.	TT1	RR1	PUMP-1	OFF
3.	TT2	RR2	PUMP-2	ON
4.	TT2	RR2	PUMP-2	OFF
5.	TT1	TT2	BUZZER-1	ON
6.	TT1	TT2	BUZZER-1	OFF

Table 8: Generated Test cases through test paths in the Neural Network

Test case Num.	TT1	TT2	RR1	RR2	Output
1.	28	-	31	-	PUMP-1-STATUS = OFF
2.	32	-	31	-	PUMP-1-STATUS = ON
3.	-	28	-	32	PUMP-2-STATUS = OFF
4.	-	33	-	32	PUMP-2-STATUS = ON
5.	28	29	-	-	BUZZER-1-STATS = OFF
6.	28	31	-	-	BUZZER-1-STATS = OFF

Table 9: Firmware Test Results

Test case Num.	TT1	TT2	RR1	RR2	Output from NN	Output from Firmware
1.	28	-	31		PUMP-1-STATUS = OFF	PUMP-1-STATUS = OFF
2.	32	-	31		PUMP-1-STATU = OFF	PUMP-1-STATU = ON
3.	-	28	-	31	PUMP-2-STATUS = OFF	PUMP-2-STATUS = OFF
4.	-	32	-	31	PUMP-2-STATUS = ON	PUMP-2-STATUS = ON
5.	28	29	-	-	BUZZER-1-STATUS = OFF	BUZZER-1-STATUS = OFF
6.	28	31	-	-	BUZZER-1-STATUS = ON	BUZZER-1-STATUS = ON

Table 10: Test suite generated from Input Domain for the Prototype application using NNBS

Test case No.	Input Vector				Expected Output	
	Temp 1 (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 11: 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	04	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)