2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)

Test Point Insertion Using Artificial Neural Networks

Yang Sun, Spencer Millican
Department of Electrical and Computer Engineering
341 War Eagle Way, Auburn, AL 36849
Tel: 334/844-1873 Fax: 334/844-1809
yzs0057@auburn.edu, millican@auburn.edu

***Abstract*-A method of data collecting, training, and using artificial neural networks (ANNs) for evaluating test point (TP) quality for TP insertion (TPI) is presented in this study. The TPI method analyzes a digital circuit and determines where to insert TPs to improve fault coverage under pseudo-random stimulus, but in contrast to conventional TPI algorithms using heuristically-calculated testability measures, the proposed method uses an ANN trained through fault simulation to evaluate a TP's quality. The time of feature extraction is demonstrated to be significantly faster compared to heuristic-based TP evaluation, and the impact of inserted TPs is shown to provide superior stuck-at fault coverage compared to conventional heuristic-based testability analysis.**

Keywords: Design for test, test point insertion, built-in self-test, artificial neural networks

I. INTRODUCTION

Pseudo-random digital circuit testing is an established industry practice due to its simplicity and significant fault coverage. Pseudo-random testing (often named "random testing") is the process of generating stimulus of a variable, yet predictable, nature and observing the response, often with on-chip stimulus and observation hardware. This test paradigm has been frequently studied and applied in industry due to its low-cost of implementation, significant stuck-at fault coverage, the ability to apply tests "at-speed", and the ability to apply tests "in-the-field" after a device is delivered to a customer.

When applied to modern circuits, pseudo-random tests can fail to excite and observe "random pattern resistant" (RPR) faults without circuit-specific considerations. RPR faults manifest in logic with many inputs when few input combinations can excite select logic paths. A representative example of such a fault is the output of a 32-bit wide AND gate stuck-at-1, which requires 32 logic-1 values to be applied to the input of the AND gate. Detecting RPR faults with random stimulus requires either **1)** modifying the pattern generator to create less "random" stimulus or **2)** modifying the circuit to make RPR faults more easily tested with random stimulus. This later method, known as test point (TP) insertion (TPI), is frequently used in industry due to its ability to be implemented on post-synthesis circuit netlists with minimal effort from circuit designers.

Using TPs to increase random pattern effectiveness is well established in literature, but TPI methods still strive to improve their computational performance. The TPI problem was originally proposed in [1], and later studies improved upon it by using fault simulation to classify RPR faults and to propose TP locations [2] by using automatic test pattern generation (ATPG) [3], by using gradient optimizing techniques [4], and by incorporating many other nuances and using a variety of algorithms. All of these methods are less-than-optimal, which is necessary because optimal TP placement is a known computationally-infeasible problem [5]. Therefore, it is imperative to continue exploring TPI methods which select higher-quality TPs with less computational resources, since reduced execution time translates to increased TPI efforts, which in turn translates to higher-quality TPs being inserted into a circuit in a given amount time.

Artificial neural networks (ANNs) are computing paradigms inspired by biological neural networks which present opportunities to overcome solution quality and computational barriers imposed by heuristic algorithms. ANNs use solved example problems and training algorithms to create a problem-solving algorithm, and ANNs have been used to solve previously unsolvable problems like handwriting analysis [6]. The advantage of ANNs is they do not require problem-specific heuristic algorithms to be explored and can find input-solution correlations which engineers may fail to incorporate into their heuristic algorithms.

Given the desire to decrease the large computation time of conventional TPI methods, this study proposes using ANNs to select quality TP locations. This new method collects training data from training circuits, trains an ANN, and then uses this ANN to predict the quality of a TP inserted on a given circuit location. This ANN will be trained using the fault-simulated effect of inserting TPs on arbitrary training locations, with the goal of this ANN being to provide accurate TP evaluations with less computational effort. This study explores the input features of the ANN and how to extract training data from circuits in a scalable manner.

The remainder of this article is organized as follows: Section II overviews TPs and circuit testability calculations. The training and use of the ANN is described in Section III. Experimental results are given and discussed in Section IV and Section V. Section VI concludes this article and provides future research directions.

II. TEST POINTS AND TESTABILITY MEASURES

TPs are circuit modifications which change circuit logic during test but do not change the circuit function when disabled. TPs are categorized into two types: *control TPs* and *observe TPs*. Control TPs change circuit *controllability*, which is the probability a line is logic-1 when random stimulus is applied. This is typically accomplished by inserting OR gates (see Figure 1(a)) for *control-1 TPs* or AND gates (see Figure 1(b)) for *control-0 TPs* and an extra "test enable" signal. The goal of these gates is to increase the probability of exciting faults in the circuit and to make faults easier to observe by creating propagation paths to circuit outputs [5]. *Observe TPs* change

978-1-7281-3391-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ISVLSI.2019.00054

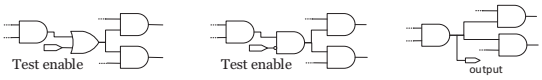


Figure 1: Illustrated here are the logic-level implementations of control-0 (a), control-1 (b) and observe (c) TPs.

circuit *observability* by inserting fan-outs to circuit outputs or memory (as shown in Figure 1(c)), which makes the line (and fault effects on a line) easily observed.

Choosing high-quality TP locations is computationally intensive, but many methods have successfully used circuit probability estimations, e.g. COP [7] and other analogous algorithms, as a basis to predict the impact of TPs on fault coverage. For each line in a circuit from inputs to outputs, COP calculates the probability each line in a circuit will be logic-1, or "CC", which is calculated in an inputs-to-outputs order using the input controllabilities of a gate and the gate type. The observability of a line, "CO", estimates the probability that a logic value on a line will be observed at a circuit output, which is calculated in an outputs-to-inputs order using the output observability of a driven gate and the input controllability of other lines driving the gate. With controllability and observability values of a line, the detection probability, "d_f", of a stuck-at fault "f" on a line "n" can be predicted: $d_{f,sa0} = CC_n \cdot CO_n$ for stuck-at 0 faults and $d_{f,sa1} = (1 - CC_n) \cdot CO_n$ for stuck-at 1 faults [4]. Presuming all circuit inputs "I" have an equal probability of being logic-0 and logic-1 (i.e., $\forall n \in I: CC_n = 0.5$) and all outputs "$O$" are directly observable (i.e., $\forall n \in O: CO_n = 1$), the predicted fault coverage " FC_p " amongst all faults "F" can be predicted. It must be noted that this fault coverage is not the exact fault coverage of the circuit as it does not consider the random nature of stimulus and it is prone to mis-calculations from reconvergent fan-outs[7].

$$FC_p = \frac{1}{|F|} \sum_{\forall f \in F} d_f$$

TPI algorithms have used COP [7] (and other analogous methods [8]) to heuristically predict the impact TPs have on fault coverage and to iteratively insert the most effective TP. Typically, the fault coverage "FC_p" before any TPs are inserted is calculated, then, for all candidate TPs, "$\forall t \in T$", the fault coverage of the circuit with the TP, "$FC_{p,t}$", is calculated. The TP which increases the fault coverage the most, $t' = \arg\max_{\forall t \in T} FC_{p,t}$, is found. If no TP increases fault coverage, i.e. if $\forall t \in T: \Delta FC_{p,t} = FC_{p,t} - FC_p \leq 0$, then TPI is halted. Otherwise, the TP t' is inserted, t' is removed from T, and the process repeats. This continues until **1)** no TPs are predicted to increase fault coverage, **2)** the number of TPs inserted has reached its designated limit, **3)** the predicted fault coverage has reached a satisfactory level, or **4)** a computation time limit is reached.

Although "list-based heuristics" which predict the impact of TPs have been demonstrated to be effective at increasing fault coverage under random stimulus, such methods can be **1)** prone to inaccuracies and **2)** computationally intensive. COP (and other analogous methods) cannot predict precise circuit controllability and observability measures (and thus testability measures) in the presence of reconvergent fan-outs [7], and thus TP placement using such calculation methods may be less than ideal. Also, although the time to heuristically predict a TP's impact on fault coverage is significantly faster than a precise calculation (which requires fault simulation), predicting the impact of every candidate TP can be computationally burdensome. To overcome computational resource limits, a sub-set of TPs may be evaluated or fewer TPs will be inserted, but these options will lead to lower quality TPs or fewer TPs being inserted.

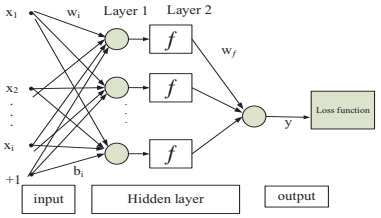


Figure 2: An ANN consists of one or more input features, one or more hidden layers, and one or more outputs. The function of the ANN is defined by the "weights" (w) and "baises" (b) of neuron outputs, which are adjusted during ANN training.

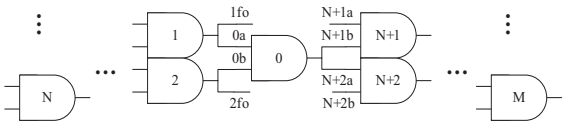
III. A Neural Network Applied to TPI

The ANN created by this study performs testability analysis on a circuit before and after a TP is inserted and finds the change in fault coverage, "$\Delta FC_{p,t}$", without performing fault simulation. The goal of this ANN is to provide accurate $\Delta FC_{p,t}$ values of a TP in a computationally expedient manner.

A. An introduction to ANNs

ANNs are powerful machine learning mechanisms which mimic biological neural networks (i.e., a brain), and ANNs have been repeatedly demonstrated to be effective at solving difficult problems. ANNs can solve previously computationally infeasible problems and have been used in various fields. For circuit testing, ANNs have been used for fault diagnostics (analog [9] and digital [10]) and as a model for circuits in digital test generation [11]. Although ANNs have been used to model circuits for test generation (thereby solving problems through "training" as opposed to using a trained ANN), to the authors' knowledge ANNs have never been trained and then used for test generation, random or otherwise.

ANNs are composed of input layers, hidden layers, and output layers, as illustrated in Figure 2. Each layer $l \in L$ contains $|N_l|$ neurons, with the number of neurons in the input (first) and output (last) layer being equal to the number of input "features" and "outputs" of the ANN, respectively. Neurons are connected with "dendrites" (the graph edges in Figure 2), and each dendrite has a weight "w_i" and a bias "b_i" assigned to it. Each neuron has an "activation function", f (a.k.a., a "transfer function"), which determines its output based on the sum of incoming signal values multiplied by dendrite weights and added biases, i.e., $x_n = f(\sum_{\forall i \in N_n} x_i w_i + b_i)$ where N_n are all neurons which are inputs to the neuron n. Evaluating an ANN for a vector of input features is done using a simple depth-order evaluation of neuron outputs.



NN Input Vector

$CC_0, CC_{1fo}, CC_{0a}, CC_{0b}, CC_{2fo}, CC_1, CC_2, \ldots, CC_{Nb}, CC_{N+1a}, CC_{N+1b}, \ldots, CC_{N+M-1}, CC_{N+M},$
$CO_0, CO_{1fo}, CO_{0a}, CO_{0b}, CO_{2fo}, CO_1, CO_2, \ldots, CO_{Nb}, CO_{N+1a}, CO_{N+1b}, \ldots, CO_{N+M-1}, CO_{N+M},$
$Gate\_0, Gate\_1, Gate\_2, \ldots, Gate\_N, Gate\_N+1, Gate\_N+2, \ldots, Gate\_N+M$

Figure 3:The ANN input features are the controllabilities, observabilities, and gate types before and after the TP location.

The key to ANNs' effectiveness is the automatic generation of dendrite weights and biases using training data and training algorithms. An ANN with randomly-assigned weights and biases is all but useless, and manually finding weights and biases which solve a given problem is tedious if not infeasible. Thankfully, many methods exist for computationally finding a set of dendrite weights and biases which match input vectors to known output solutions. These methods first create an ANN with random weights and biases (although studies debate the impact of initial values [12]) and then adjust the weights and biases to minimize the difference between the ANN output and known-solved outputs for given input examples (i.e., the "loss function"). Although the optimal dendrite weight and bias values may not be found without a computationally-infeasible exhaustive search (due to local optimization minima and discrete search step sizes), training can be repeated across several initial conditions and different parameters to increase ANN accuracy against available training problems.

Many parameters can affect an ANN's performance, and these parameters have trade-offs. The size and configuration of an ANN, in terms of the number of neurons, number of layers, and the connectivity between layers, directly limits the best possible ANN performance with simpler structures limiting performance. However, complex structures can take significantly longer to train and require more (unavailable) data to train. Different choices in activation functions can likewise effect performance: choices include linear, binary step, rectified linear unit (ReLU) [13], and many other activation functions [14]. The effectiveness of these activation functions, as well as ANN structure configurations, are best explored through trial-and-error, as will be demonstrated in Section IV.

B. ANN input format

The input features to the TP-evaluating ANN are circuit CC values, CO values, and gate types. CC and CO are the same values used by other conventional TPI algorithms [8], but the ANN will use only pre-TP insertion values. Using these values in such a manner presents several potential benefits. First, using only pre-TP activation values forgoes the need to re-calculate CC and CO values for every TP and can decrease TP evaluating time. Second, an ANN can find relations between values during training which heuristic algorithms may not take advantage of. For instance, the known issue of CC and CO values not considering fan-outs is ignored by other TPI algorithms, but a trained ANN can find and act on such nuances automatically.

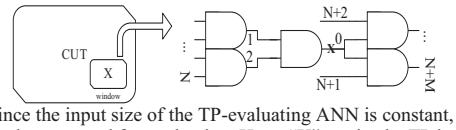


Figure 4: Since the input size of the TP-evaluating ANN is constant, sub-circuits must be extracted for evaluation. Here, "X" marks the TP location.

Gate types of nodes are represented using a one-hot encoding with each gate type having a separate encoding (e.g., "AND" = 0001, "OR" = 0010, etc.). Note that if more gate types or standard cells are used in a circuit, longer one-hot strings and more complex ANNs will need to be trained.

The input vector to the ANN are the previously discussed values around a candidate TP location in a vectorized format, as is illustrated in Figure 3. The first value is the CC of the candidate TP location. This is followed by the CC of gate inputs (and fan-outs) feeding the TP location, then the CC of gate inputs (and fan-outs) feeding these gates, etc., in a breadth-first levelized order. This is repeated until the values of N gates are collected, to which the limit N is discussed in the next paragraph. This process is repeated starting at the candidate TP location moving forward in the circuit until the values of M gates are collected. CC values are followed by CO values in the same order. Finally, gate types of each gate in the circuit are recorded in the same order with their one-hot encoding.

Since ANNs must be of a defined size and circuits come in many sizes, the ANN input will be a sub-circuit centered around a candidate TP's location, as is illustrated in . The ideal size of an analyzed sub-circuit is equal to or greater than the size of the entire circuit, but this will require the ANN to be unduly large for practical circuits. The sub-circuit size can be expressed in terms of the number of circuit levels before and after the TP location, which in turn can be translated into the number of elements "N, M" to capture. If the circuit to analyze is smaller than the ANN sub-circuit size, CC values, CO values, and gate types will be replaced with "default" values: non-existent inputs/fan-outs in N will have 50% logic-1 controllability and identical observability as the line they drive; non-existent outputs/fan-ins in M will have 100% observability and identical controllability as the line they are driven by; gate types will be encoded as a "no-hot" (all zeros). Using sub-circuits presents a potential detriment to the ANN compared to heuristics which include the entire circuit in their calculation, but experimental results will explore if using sub-circuits, combined with the ANN's training, overcomes this limitation.

Before the ANN is trained or used, circuits are converted into a consistent fan-in/fan-out structure. This is done because ANNs require inputs features to be of a constant order which denotes a circuit structure of 2-input, 2-fan-out gates, as is illustrated in Figure 4. Any given circuit can be converted to this structure by replacing gates with more than 2 inputs with functionally equivalent copies and by adding buffers to fan-outs, as is illustrated in Figure 5. For gates with one input (i.e. buffers and inverters), the CC and CO of the non-existent input line is replaced by a "default" value described in the previous paragraph. The same "default" replacement is performed for gates with no fan-outs.

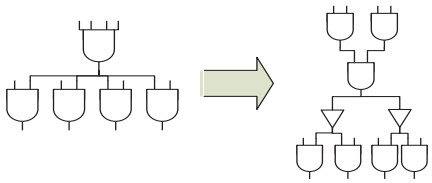


Figure 6: To extract features from a circuit in a consistent manner, all gates with more than two fan-ins and fan-outs will be converted before features are extracted.

C. *Training data generation*

The data generation process includes generating input features and corresponding outputs required to train the ANN. First, arbitrary locations are chosen in training circuits and sub-circuits are extracted from the vicinity of these locations. Second, CC and CO values (which require a single-pass calculation per training circuit to generate) and gates types (AND, NOR, etc.) in the sub-circuit are recorded.

To generate the training output (the change in circuit fault coverage after inserting a TP), procedures are implemented to reduce fault simulation time. Simulating all $2^{|I_S|}$ input patterns (where $|I_S|$ is the number of inputs of the sub-circuit) for a sub-circuit takes significantly less time compared to simulating 10K (or other common pseudo-random test lengths) random patterns for fault simulation for practical values of $|I_S|$, but even under many random vectors, all $2^{|I_S|}$ input patterns may not be applied. This is because the CC, CO values of inputs can prevent all patterns being applied, even when the number of patterns applied is significantly greater than $2^{|I_S|}$. The probability of generating an input pattern based on CC and CO can be calculated as $p = \prod_{\forall i \in I} CC_i'$, where CC_i' is $1 - CC_i$ if a 0-value is needed. The probability of generating a pattern in V vectors is $P_i = 1 - (1-p)^V$. Likewise, if a fault is observed on an output "o", the probability of the fault being observed on the pin in V vectors is $P_o = 1 - (1 - CO_o)^V$. By calculating these probabilities, every $2^{|I_S|}$ input vector is simulated at most once with probability P_i, and if the vector is applied, simulated faults will be considered detectd with probability P_o. To account for andom variation, P_i is calculated several times and the average value of P_i is used.

D. *ANN outputs and training*

The output of the ANN is the impact on fault coverage "ΔFC_t" a TP has on a circuit when inserted and activated. These values are found through fault simulation of a circuit before and after a TP is inserted. This is a relatively computationally intensive process to perform, but every time it is performed more training data will be generated and the accuracy of the ANN will increase. To avoid the ANN being too specialized to one type of circuit, fault-simulated TP impacts should be calculated for a diverse set of circuits.

To make the ANN training process simpler, three separate, smaller ANNs will be trained representing the three TP types: control-0, control-1, and observe TPs. The alternative to this is to have a single ANN with an extra "TP type" input, but this would make the complexity of the ANN unnecessarily large. When evaluating a TP's quality, the corresponding trained ANN will be used.

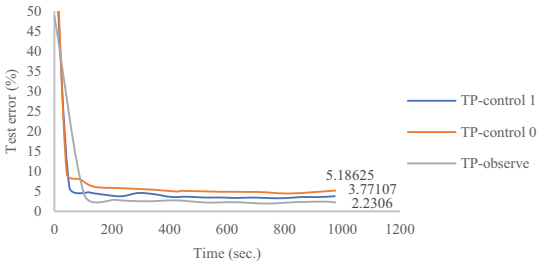


Figure 5: The impact on ANN training time and ANN accuracy is noteworthy. ANNs should be trained until accuracy "saturates" and no further significant benefit to training is gained.

E. *ANN use in a TPI algorithm.*

The use of a trained ANN in a TPI algorithm is analogous to the use of TP-evaluating heuristics in iterative TPI algorithms, as described in Section II. Until one of the four limits are met, the "best" TP among all candidate TPs (i.e., the one which has the highest $\Delta FC_{p,t}$) is inserted. Using the ANN, this is done by calculating all COP values in the circuit and then feeding these values and gate types into the ANN for every candidate TP. Since all TP share the same pre-insertion COP values, these COP values need only be calculated once among all candidate TPs, as opposed to re-calculating after each TP is inserted for existing heuristics.

IV. Experimental Results –Training

The benchmarks used in this study for ANN training are given in Table 1. These are from the ISCAS'85 [15] and ITC'99 [16] benchmarks, which represent a wide range of industry-representative circuits. The table provides the number of nets in each benchmark ("Num. nets") and the number of nets in the circuit after it is expanded ("Num. exp. nets") using the method presented in Section III.B. The number of training TPs and sub-circuits extracted is given ("Num. samples"). From each sample/sub-circuit, the impact of a control-0, control-1, and observe TP is fault simulated using the method presented in Section III.C .

The first issue explored through ANN training is the impact of different ANN structures. As was discussed in Section III.A, many different parameters influence the quality of an ANN, including ANN size (the number of neurons and the number of layers), ANN connectivity, training time, and neuron activation functions. These parameters were thoroughly explored through several parameter sweeps to find the configuration which decreases the average error of the ANN. The data showing the progression to the final chosen parameters is too cumbersome to be shown here, but several trends can be illustrated. Figure 6 shows the three separate ANNs' accuracy as more training time is allocated, which shows that training effort is "saturated" after some point. All ANNs in this study are trained beyond this saturation point. Figure 7 shows the accuracy of different ANNs for varying number of neurons, which shows more neurons may not produce a higher quality ANN. The final parameters chosen were two hidden layers with 128 neurons in the first layer and a single neuron in the second layer. A ReLU activation function and the Adam Optimization method [17] was used for training the ANN.

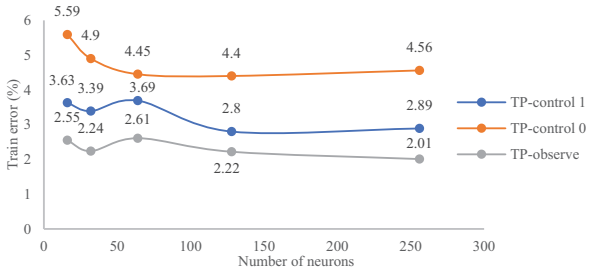


Figure 7: It has been observed that the number of neurons used to make the ANN has an impact on the ANN accuracy, but increasing the number of neurons does not necessarily yield positive results.

The second issue explored is how the size of training data influences the quality of the ANN. Figure 8 plots the ANNs' error with respect to training data size. This shows that small training data sizes creates an inferior ANN, but large data sizes will increase ANN training time with minimal returns on ANN quality. Different sizes of training data were collected and used to train the ANNs: 0.1%, 0.5%, 0.9%, and 1.5% of all TP locations in all training circuits. From this, the ANN with the lowest error is chosen: the total training samples/sub-circuits is 11,561, and the data collecting time was 33.6 minutes, and the ANN training time is 16.3 minutes. This 50-minute overhead will be considered with evaluating the computational efficiency of ANN-based TPI.

V. Experimental Results – TPI

The results of fault simulation on benchmark circuits are shown in Table 2. First, this table shows the final fault coverage for three variations of benchmark circuits: with no TPs, after ANN-based TPI is performed, and after COP-based TPI [8] is performed, with the COP-based TPI using the same information used for ANN training (i.e., neither the ANN or conventional TPI have an information advantage). These three circuit variations were fault simulated with 100, 1,000, and 10,000 vectors to confirm the effect of TPs are consistent under different stimulus conditions. For both TPI methods, the number of TPs inserted is listed under "num. TPs", which is limited to 1% of the total number of nets (predicted fault coverage limits or negative TP impact limits were never reached for any benchmark). Second, the table lists the time required to perform TPI for both ANN-based TPI and COP-based TPI [8]. To account for random stimulus variation, the average fault coverage is recorded amongst multiple trials: 100 trials for 100 vectors, 50 trials for 1,000 vectors, and 10 trials for 10,000 vectors.

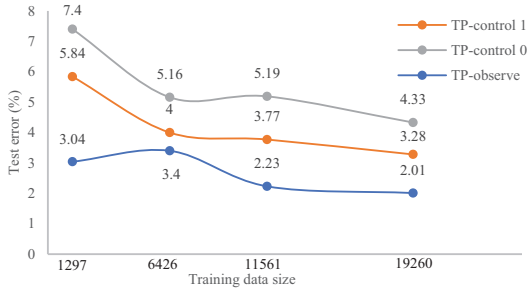


Figure 8: The impact on ANN training data size and ANN accuracy is noteworthy. ANN accuracy increases with more training data, but accuracy returns diminish at the expensive of increased training time.

Table 1: Training Benchmark Circuits

Benchmark	Num. nets	Num. exp. nets	Num. samples
c17	11	11	2
c499	243	425	6
c1355	587	881	13
c1238	543	1016	14
c2670	1426	1969	25
c6288	2448	3376	52
c7552	3719	5482	72
b02	27	46	2
b04	729	1221	16
b06	50	93	2
b08	179	304	4
b10	200	342	5
b12	1070	1900	26
b14	10044	17719	242
b15	8852	16920	232
b17	32229	59818	817
b18	114598	201844	2758
b19	231290	405924	5549
b20	20204	35993	491
b21	20549	36916	503
b22	29929	53564	730

Table 2 shows the fault coverage created from TPs inserted by the ANN-based TPI algorithm is decisively superior compared to conventional TPI. This is better illustrated Figure 9 which gives the relative fault coverage obtained throughe ANN-based TPI compared to COP-based TPI. These results reveal several findings. First, the fault coverage of ANN-based TPI is frequently higher, with the only benchmark providing lower fault coverage being *b13*. Second, it is noteworthy that extreme differences in fault coverage favor ANN-based TPI, as is demonstrated by the benchmark *b05*.

The second observation from Table 2 is the time to perform TPI is significantly less for ANN-based TPI compared to COP-based TPI, including when ANN training and data collecting time is proportionally distributed amongst circuits according to circuit size. This is better illustrated in Figure 10 which gives the execution time of ANN-based TPI compared to COP-based TPI. This figure clearly shows the ANN-based TPI performs faster than COP-based TPI across all benchmark circuits, with on average TPI being done in one-sixth the time. This result is understandable given the ANN-based TPI does not need to recalculate CC and CO values for every candidate TP and instead evaluates the ANN for its input vector, which is a relatively fast process for a small sub-circuit. When training time (approximately 50 minutes) is distributed proportionally amongst all benchmarks according to size (e.g., smaller circuits add a smaller fraction of the 50 minutes and vice versa), the impact on TPI time becomes negligible for larger circuits and the total TPI time is still decreased across all benchmarks.

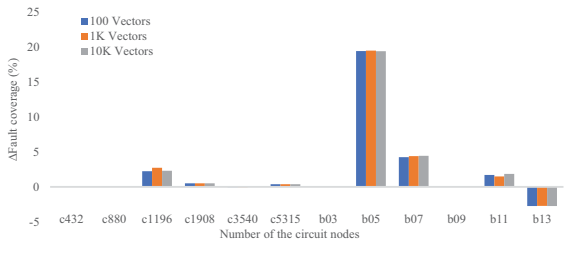


Figure 9: When comparing the fault coverage of the proposed ANN-based TPI method and traditional TPI methods, most benchmarks show significant improvement in fault coverage.

Table 2: Experimental Results

Benchmark			c432	c880	c1196	c1908	c3540	c5315	b01	b03	b05	b07	b09	b11	b13
Num. TPs			1	3	5	8	16	23	0	1	9	3	1	7	2
Num. nets			196	443	561	913	1719	2485	47	156	962	433	169	764	352
Num. exp. nets			310	682	978	1315	2693	4396	80	268	1783	706	280	1271	543
Processor memory (MB)			4	11	21.7	154.5	170	227.1		3.6	42.6	9.2	9.9	32.5	5.9
	Num. Vectors														
Fault coverage (%)	No TPs	100	92.37	90.31	66.99	75.86	80.5	94.3	100	98.12	66.63	88.71	79.86	80.24	89.94
		1K	98.58	97.26	87.14	94.6	94.47	99.18	100	99.98	75.5	92.76	84.6	92.06	94.52
		10K	98.84	99.71	92.97	99.73	96.38	99.41	100	100	80.04	97.55	98.56	95.38	95.78
	ANN TPI	100	98.82	99.86	94.23	99.69	96.4	99.44		99.99	83	97.52	99.57	95.91	95.75
		1K	98.83	99.84	94.65	99.83	96.38	99.44		100	82.95	97.68	99.02	95.66	95.75
		10K	98.84	100	94.35	99.91	96.47	99.44		100	83.03	97.75	99.91	96.07	95.79
	COP TPI	100	98.82	99.85	92	99.19	96.45	99.06		99.99	63.62	93.28	99.53	94.2	98.48
		1K	98.83	99.82	91.92	99.32	96.43	99.06		100	63.5	93.3	98.96	94.18	98.47
		10K	98.84	100	92.06	99.41	96.47	99.06		100	63.66	93.32	99.85	94.21	98.52
Execution time (min.)	ANN TPI	100	1.14	7.32	30.94	128.72	967.51	1568.82		0.38	131.62	10.79	0.58	72.9	2
		1K	1.09	7.17	30.43	124.62	894.43	1488.73		0.11	122.51	9.97	0.55	68.59	1.8
		10K	1.06	8.31	31.65	123.63	930.01	1508.25		0.31	122.19	10.08	0.58	69.67	1.8
	ANN TPI including training time	100	2.08	9.46	34.19	133.12	976.59	1583.21		1.18	137.62	13.04	1.44	77.13	3.65
		1K	2.03	9.31	33.68	129.02	903.51	1503.12		0.91	128.51	12.22	1.41	72.82	3.45
		10K	2.00	10.45	34.90	128.03	939.09	1522.64		1.11	128.19	12.33	1.44	73.9	3.45
	COP TPI	100	2.27	23.14	75.5	323.87	7321.71	7199.8		1.25	4004.53	27.12	1.62	204.02	10.06
		1K	2.36	23.31	78.39	323.37	5302.2	7015.34		1.31	482.72	25.9	1.56	195.05	10.02
		10K	2.17	23.77	76.44	329.86	7314.74	7003.95		1.2	3578.09	26.53	1.56	195.57	9.8

VI. Conclusions & Future Directions

This study has demonstrated the effectiveness of applying ANNs to TPI. This study has shown that ANNs can effectively predict the impact TPs will have on the stuck-at fault coverage for pseudo-random stimulus compared to conventional TP-evaluating heuristics. It has also been demonstrated that ANNs can insert TPs into a circuit at significantly increased speeds, thereby allowing design efforts to be focused elsewhere or allowing additional effort to be spent on increasing TP quality. Many future research directions are left unexplored, and it is the authors' intention to further apply the utility of ANNs to circuit testing. The results shown in this study suggest the performance of ANN-based TPI can be further improved to make their impact consistently positive, especially against other TPI methods which consider additional information. Possible methods of improvement include the use of newer ANN structures and including additional features as ANN inputs. ANNs can also be applied to changing the nature of random stimulus (i.e., biased/weighted random testing), as well as testing fault models representative of modern technologies, i.e., delay faults.

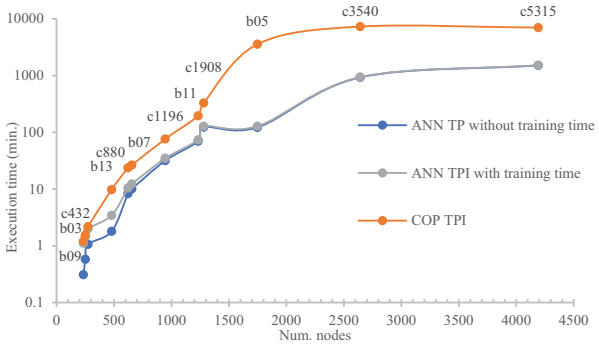


Figure 10: The proposed ANN-based TPI method provides a significant reduction in TPI time even when data collecting and training time is considered.

References

[1] J. P. Hayes and A. D. Friedman, "Test Point Placement to Simplify Fault Detection," *IEEE Trans. Comput.*, vol. C-23, no. 7, pp. 727–735, 1974.

[2] A. Briers and Totton, K., "Random pattern testability by fast fault simulation," in *Proc. IEEE International Test Conference*, 1986, pp. 274–281.

[3] M. J. Geuzebroek, J. T. Van Der Linden, and a. J. Van De Goor, "Test point insertion for compact test sets," *Proc. Int. Test Conf. 2000 (IEEE Cat. No.00CH37159)*, pp. 292–301, 2000.

[4] M. J. Geuzebroek, J. T. Van Der Linden, and a. J. Van De Goor, "Test point insertion that facilitates ATPG in reducing test time and data volume," *Proceedings. Int. Test Conf.*, 2002.

[5] N. A. Touba and E. J. McCluskey, "Test point insertion based on path tracing," *VLSI Test Symp. 1996., Proc. 14th*, pp. 2–8, 1996.

[6] J. Pradeep, E. Srinivasan, and S. Himavathi, "Diagonal based feature extraction for handwritten character recognition system using neural network," *ICECT 2011 - 2011 3rd Int. Conf. Electron. Comput. Technol.*, vol. 4, pp. 364–368, 2011.

[7] F. Brglez, "On testability analysis of combinational networks," *Proc. - IEEE Int. Symp. Circuits Syst.*, vol. 1, Jan. 1984.

[8] H.-C. Tsai, C.-J. Lin, S. Bhawmik, and K.-T. Cheng, "A hybrid algorithm for test point selection for scan-based BIST," *Proc. 34th Annu. Conf. Des. Autom. Conf. - DAC '97*, pp. 478–483, 1997.

[9] M. Aminian and F. Aminian, "Neural-network based analog-circuit fault diagnosis using wavelet transform as preprocessor," *IEEE Trans. Circuits Syst. II Analog Digit. Signal Process.*, vol. 47, no. 2, pp. 151–156, 2000.

[10] H. Tatsumi, Y. Murai, and S. Tokumasu, "Logic circuit diagnosis by using neural networks," in *Proc. IEEE International Symposium on Multiple-Valued Logic*, 2001, pp. 345–350.

[11] S. T. Chakradhar, M. L. Bushnell, and V. D. Agrawal, "Toward massively parallel automatic test generation," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 9, no. 9, pp. 981–994, Sep. 1990.

[12] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," *1990 IJCNN Int. Jt. Conf. Neural Networks*, pp. 21–26 vol.3, 1990.

[13] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, 2013, vol. 30, no. 1, p. 3.

[14] B. Karlik, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *Int. J. Artif. Intell. Expert Syst.*, vol. 1, no. 4, pp. 111–122, 2015.

[15] F. Brglez and H. Fujiwara, "A neural netlist of 10 combinational benchmark designs and a special translator in fortran," in *Proc. of International Symposium on Circuits and Systems*, 1985, p. 669.

[16] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Des. Test Comput.*, vol. 17, no. 3, pp. 44–53, 2000.

[17] D. P. Kingma and J. L. Ba, "ADAM: a method for stochastic optimization," pp. 1–15, 2015.