Machine Intelligence for Efficient Test Pattern Generation

Soham Roy, Spencer K. Millican, and Vishwani D. Agrawal
Department of Electrical and Computer Engineering
Auburn University, Auburn, AL 36849-5201
{*szr0075, millican, agrawvd*}*@auburn.edu*

2020 IEEE International Test Conference (ITC) | 978-1-7281-9113-3/20/$31.00 ©2020 IEEE | DOI: 10.1109/ITC44778.2020.9325250

***Abstract*—This study examines machine intelligence's (MI) ability to enhance automatic test pattern generation (ATPG) by reducing backtracks. In lieu of a conventional heuristic to decide backtracing directions, this study uses an artificial neural network (ANN) trained through PODEM on hard-to-detect faults. Training data contains topological data, testability measures, and backtracking history, and when trained on this data, the ANN guides backtracing in directions unlikely to backtrack. When trained with a single feature (e.g., COP), ATPG performance is comparable to conventional PODEM, and using multiple features further reduces backtracks and ATPG CPU time.**

I. Introduction

Automatic test pattern generators (ATPGs) are NP-hard algorithms [1]–[3], thus ATPGs use heuristics to reduce test generation time. Creating effective heuristics for all situations is difficult, but advances in machine intelligence (MI) can create effective heuristics with minimal programmer effort: MI algorithms automatically program themselves using past experience when a designer's decisions cannot be easily programmed. Several studies applying MI to circuit testing (see Section II-B) improved algorithm outcomes and reduced CPU time, thus one can foresee the same advantages when MI is applied to ATPG.

This study replaces conventional backtracing heuristics in ATPG with artificial neural networks (ANNs) and observes their impact on ATPG CPU time. This article presents procedures for collecting training data, training an ANN, and integrating the trained ANN into an ATPG algorithm. To evaluate the ANN's utility, training data contains features used by conventional ATPG heuristics, and the effect of using various feature sets is examined. The ANN is used in the PODEM [4] ATPG, but the ANN can be applied to any ATPG algorithm that traces through a circuit. The specific contributions of this study include:

- A procedure to collect training data for backtrace-guiding ANNs.
- An exploration of ANN training with varying ANN complexity.
- Comparison of CPU time and other performance parameters of ANN-based ATPG against conventional heuristics.

Section II introduces relevant literature on ATPG algorithms and applications of MI to test problems. Section III shows how to train an ANN to guide backtracing in ATPG. Section IV explores training of ANNs on several sets of training data. Section V evaluates the performance of ANN-based PODEM against PODEM using conventional heuristics. Section VI suggests future research directions and concludes the article.

II. Prior Work

A. ATPG algorithms and role of heuristics

ATPG algorithms search for input vectors to detect faults. For a combinational circuit (or a full-scan circuit), the search space consists of $2^{\#PI}$ vectors, where $\#PI$ is the number of primary inputs (PIs) (which can include scan latches as pseudo-primary inputs). Roth's D-ALG [5] first conceptualized ATPG by defining the D algebra and giving a complete search algorithm, and Goel's PODEM [4] improved the search efficiency by restricting search decisions to only PIs.

Backtracing identifies PIs to assign to meet ATPG objectives (e.g., exciting or propagating faults). All modern ATPG algorithms have search space of $2^{\#PI}$, but using backtracing heuristics to strategically select PIs can quickly find a test in the search space. This study's ANN-based heuristic, "MAR" (named after the authors), is one such heuristic.

B. Machine intelligence in test

Artificial neural networks (ANNs) are MI architectures modeled after the human brain. Using ANNs trained through supervised learning requires **1)** selecting an ANN structure (fully-connected, convolutional, etc.), **2)** generating training data by solving sample problems, and **3)** training the ANN with the training data. After this, the ANN can solve problems similar to training data.

Many VLSI test studies used ANNs for circuit modeling and in algorithms. Digital circuits and the ATPG problem have been modeled as ANNs with bidirectional links where neurons represent the states of signals [6]: the network energy function of the circuit is minimized to find a test, but this minimization depends on many variables (all signals) and the function has many local minima, which makes searching for a test difficult. Recent studies on digital circuits used ANN for diagnosing scan chains [7], classifying fault models [8], testability analysis [9], and test-point insertion [10], [11]. ANNs were also been proposed to classify the results of analog and RF tests [12].

III. Machine Intelligence (MI) applied to ATPG

This study's ANN guides backtracing by prioritizing the available choices, as opposed to the conventional guidance provided by some circuit levels [4] or COP [13]. The feed-forward ANN [14] consists of inputs, a single hidden layer

978-1-7281-9113-3/20/$31.00 ©2020 IEEE

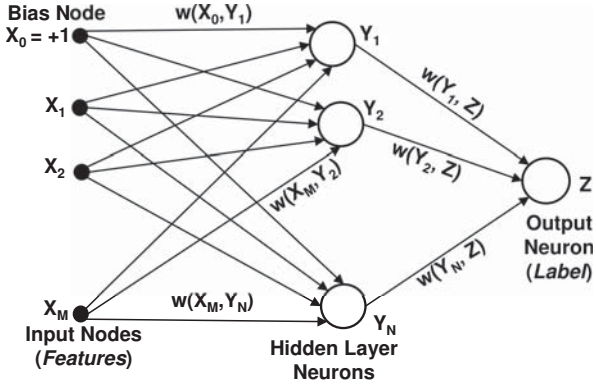

Fig. 1. The artificial neural network (ANN) in this work consists of inputs (features), hidden layer neurons, and an output neuron (label), connected through weighted edges. The weights, found through training, determine the function of the ANN.

of neurons, and an output neuron, as shown in Figure 1. The state of a neuron (i.e., it's output) is a value in the range $[0, 1]$ as is calculated using an *activation function*, f (in this study, the sigmoid function). Inputs (i.e., *features*), $X_0, X_1, \cdots, X_M$, are also driven by values in the range $[0, 1]$. X_0 is a bias input fixed at 1.0. Hidden layer neurons, $Y_1, Y_2, \cdots, Y_N$, perform calculations using their inputs, as does the output neuron, Z (i.e., the *label*). This article denotes the value of an input or a neuron in lower case, i.e., the output values of X_i, Y_i and Z are x_i, y_i and z, respectively. All neurons are connected by "dendrites": each dendrite from A to B is assigned a weight, $w(A, B)$, which is a signed floating point number in the range of $[-1, 1]$. For a given input, $\{x_0, x_1, \cdots, x_M\}$, each y_j is computed as follows, and z is a similar computation based on y_j's.

$$y_j = f(\sum_{i=1}^{M} x_i \times w(X_i, Y_j)) \quad (1)$$

Training patterns contain the values of input features and the expected output, and training finds edge weights to satisfy input-output relationships in the training data with minimum error. Finding edge weights for small ANNs can be done manually, but this is impractical for moderate to large ANNs. Instead, training algorithms compute weights that map input features onto expected output labels and minimize the output error over training patterns. These algorithms start with random weights, and over several iterations, the output label is computed across the training patterns to find the *average error*, i.e., the average difference between the desired and computed label. This error is reduced by iteratively retraining with different "hyper-parameters" (e.g., different sizes of the hidden layer, learning rates, activation functions, optimizers, etc.). Find the optimal weights by searching all possible weights is expensive, but iterative searches can find useful ANNs (see Section IV).

A. ANN input features and output label

For a circuit line, the input features of the ANN contain the following.

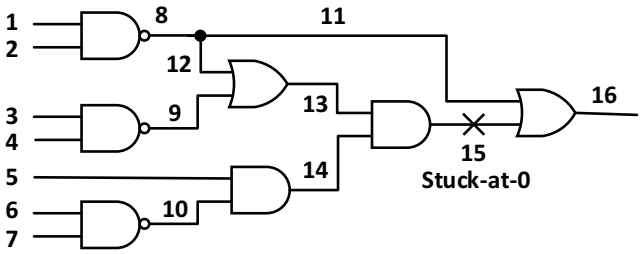

Fig. 2. Training patterns resulting from PODEM ATPG while generating the test $110X10X$ for line 15 stuck-at-0 fault in Figure 3.

- The type of gate driven by the line represented as one-hot encoded format, e.g., AND = 000000001, NAND = 000000010, OR = 000000100, etc. If necessary, the number of code bits can be expanded.
- The COP controllability, CC (i.e., probability of line being logic-1), and observability, CO (i.e., probability that line value is observed at PO). A one-time computation through the circuit finds approximated probabilities for all lines [13], and recalculation is not required during ATPG.
- The circuit level of line being traced, i.e., shortest distance to any PI from the line. This value is normalized in the range [0,1] by the maximum depth of the circuit.

During backtrace, the ANN returns the probability backtracing on a given line will result in a test (i.e., not be undone by a backtrack). When backtracing through a gate with multiple inputs, the ANN is evaluated at each circuit input, and the input which is most likely to result in a test is chosen. This is akin to COP-based or level-based easy-hard heuristic [15] that selects the line with the largest/smallest value depending on the desired characteristics.

B. ANN training data generation

Training data is obtained from successful (test found) and unsuccessful (backtracked) backtraces from ATPG trials. All backtraces to a PI assignment that generate a test for the target fault are labeled as "success" (1) and backtraces that are undone by backtracks are labeled as "failure" (0). Training data is generated using a random tracing heuristic (i.e., no set of rules is followed while backtracing). During this ATPG, the history of backtraces is recorded: when a backtrack is performed, backtraces that lead to undone PI assignments are labeled as "failures" ($z = 0$); when the ATPG finds a test, all remaining backtraces are labeled as "success" ($z = 1$).

As an illustrative example, the following steps produced the training data of Figure 3 to detect line 15 stuck-at-0 fault in the circuit of Figure 2:

- To excite the fault, an objective of "1" on line 15 is set. Backtracing through 14-5 assigns "1" to PI 5. Logic simulation shows that the objective is yet to be met.
- Another backtrace through 14-10-6 assigns "0" to PI 6, but still the objective is not met.
- Backtracing through 13-12-8-1 assigns "0" to PI 1. Simulation verifies that the objective of "1" on line 15 is met, but a "1" on line 11 blocks the fault effect from being

	Input Features												Output label
	Gate type									COP(CC)	COP(CO)	Distance	Success/Failure
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	z
Line 5	0	0	0	0	0	0	0	0	1	0.500	0.012	0.1	1
Line 14	0	0	0	0	0	0	0	0	1	0.375	0.016	0.3	1
Line 6	0	0	0	0	0	0	0	1	0	0.500	0.004	0.1	1
Line 10	0	0	0	0	0	0	0	0	1	0.750	0.008	0.3	1
Line 14	0	0	0	0	0	0	0	0	1	0.375	0.016	0.3	1
Line 1	0	0	0	0	0	0	0	1	0	0.500	0.489	0.1	1
Line 8	0	0	0	0	0	0	1	0	0	0.750	0.977	0.3	1
Line 12	0	0	0	0	0	0	1	0	0	0.750	0.023	0.4	1
Line 13	0	0	0	0	0	0	0	0	1	0.063	0.094	0.5	1
Line 1	0	0	0	0	0	0	0	1	0	0.500	0.489	0.1	0
Line 8	0	0	0	0	0	0	1	0	0	0.750	0.977	0.3	0
Line 12	0	0	0	0	0	0	1	0	0	0.750	0.023	0.4	0
Line 13	0	0	0	0	0	0	0	0	1	0.063	0.094	0.5	0
Line 3	0	0	0	0	0	0	0	1	0	0.500	0.012	0.1	1
Line 9	0	0	0	0	0	0	1	0	0	0.750	0.023	0.3	1
Line 13	0	0	0	0	0	0	0	0	1	0.063	0.094	0.5	1
Line 2	0	0	0	0	0	0	0	1	0	0.500	0.489	0.1	1
Line 8	0	0	0	0	0	0	1	0	0	0.750	0.977	0.3	1

Fig. 3. ANN training patterns derived from a ATPG trial for line 15 stuck-at-0 fault in Fig. 2. Each entry corresponds to a backtracing decision made: $z = 1$ if it resulted in a test or $z = 0$ if required a backtrack.

propagated to the PO. Therefore, a backtrack assigns the alternative value "1" to PI 1. Thus, the choices used in the previous backtrace through 13-12-8-1 are "failure" patterns with $z = 0$.

- After the backtrack, line 15 becomes "X", or unknown. To achieve the objective of 1 on line 15, backtracing through 13-9-3 assigns "0" to PI 3. On simulation, the objective of "1" on line 15 is met. The state of line 15 is denoted with D, where D means fault-free state is 1 and faulty state 0.
- To propagate line 15 value D to the PO, line 11 is now given an objective value "0". A backtracing through 8-2 assigns PI 2 to "1" and a test $110X10X$ is found. Therefore, all backtracing history without a label are assigned "success" labels with $z = 1$.

IV. Experimental Results - Training

To generate training data, PODEM is run on 100 "hard-to-detect" faults of benchmark circuits c6288, b05, and c3540: this study calls these "training" circuits, while others are "evaluation" circuits. Because easy-to-detect faults may be detected without backtracks, they might not produce useful training data, therefore hard-to-detect faults were selected using COP [13] testabilitity values: detection probabilities are calculated as $\mathrm{CC} \times \mathrm{CO}$ and $(1 - \mathrm{CC}) \times \mathrm{CO}$ for stuck-at 0 and stuck-at 1 faults, respectively, and the 100 faults with the lowest detection probability from these circuits were used for ANN training data generation.

Experiments were performed on industry-standard workstations with Intel i7-8700 processors and 8 GB RAM. All programs were written in C++ and compiled using the MSVC++14.15 compiler with maximum optimization parameters. This software implements the PODEM algorithm [4], and it is programmed to allow any one of the four heuristics (random, MAR, distance-based, or COP-based) for backtracing. The Adam optimizer [16] is used for ANN training.

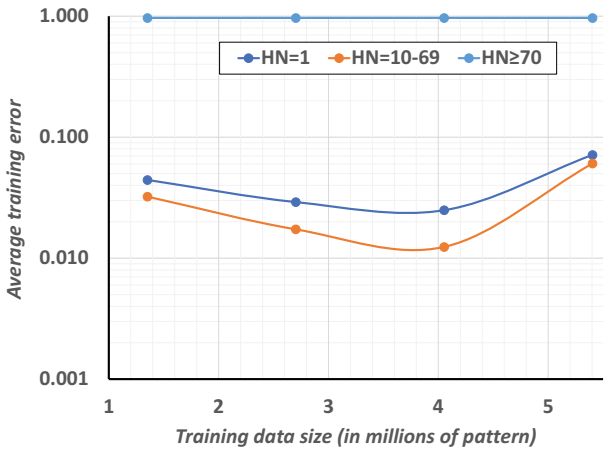

Fig. 4. Adding more training data decreases ANN error, but only to a certain point. The point which minimized error in this study was 3,730,724 patterns.

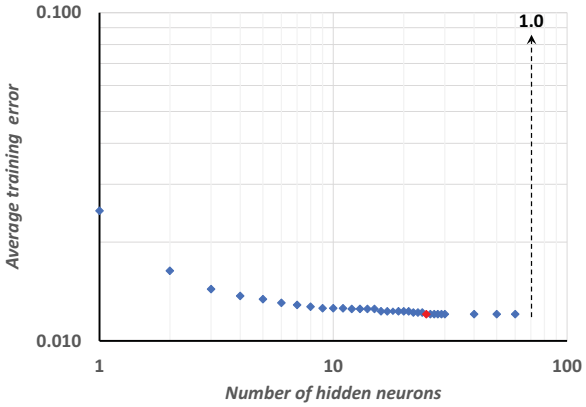

Fig. 5. As more hidden neurons are added to the ANN, error drops (ANN accuracy improves), leveling off at 25 hidden neurons, only to increase again beyond 70 neurons.

Figure 4 illustrates the impact of training data volume on ANN accuracy (i.e., average training error). As more training patterns are used, the accuracy of the trained ANN increases until a certain point, indicating that too much training data can negatively impact the accuracy. Following this observation, a set of 3,730,724 training patterns is used.

Figure 5 illustrates the impact of the number of hidden layer neurons on ANN accuracy when 3,730,724 training patterns are used. The training error is minimized at 25 neurons. Additionally, the error dramatically increases for more than 70 hidden neurons. From this, 25 neurons is used for this study's ANN.

The 3,730,724 training patterns were collected in 4.8 minutes, and ANN training required 5.1 minutes. Because training data generation and ANN training are one-time investments, they are not considered relevant costs while assessing the computational effectiveness of ANN-based ATPG.

V. Experimental Results - ATPG

A. Evaluating ANN Input Features

Before evaluating a trained ANN against conventional backtracing heuristics, this study examines the effect of us-

TABLE I
EFFECT OF INPUT FEATURES ON TOTAL BACKTRACKS IN ANN-GUIDED PODEM FOR 100 HARD-TO-DETECT FAULTS.

Circuit name	PODEM with random heuristic	PODEM with machine intelligence (MI), ANN trained with features listed in subheading							
		Untrained	Gate type	Dist.	COP	Dist.+COP	Gate type+Dist.	Gate type+COP	Gate type+COP+Dist.
c6288	12,157	10,831	10,414	12,013	11,929	10,334	8,481	10,612	**5,062**
b04	46,061	43,053	46,061	46,061	46,061	45,683	46,061	46,061	**16,973**
c432	84,080	80,725	73,352	81,041	73,440	76,956	71,918	81,365	**24,898**
b08	164	164	164	164	164	164	164	164	**118**
b03	27	27	27	50	27	50	14	47	**3**
b01	1	1	1	1	1	1	1	1	**0**

TABLE II
BACKTRACKS FOR 100 HARD-TO-DETECT FAULTS BY PODEM GUIDED BY CONVENTIONAL HEURISTICS AND THE TRAINED ANN.

Circuit name	Distance heuristic			COP heuristic			MAR (ANN trained for Gate type, COP, Dist.)		
	CPU time (ms)	#backtraces	#backtracks	CPU time (ms)	#backtraces	#backtracks	CPU time (ms)	#backtraces	#backtracks
c6288	81,915	19,478	17,914	52,547	13,633	119,74	**35,391**	**6,950**	**5,062**
b04	45,577	24,151	22,631	32,687	19,207	17,581	**39,656**	**18,555**	**16,973**
c432	21,416	42,290	40,979	40,010	87,131	85,041	**17,714**	**26,940**	**24,898**
b08	2,655	1,414	210	1,651	2,487	1,306	**562**	**1,327**	**118**
b03	262	603	38	397	662	42	**222**	**606**	**3**
b01	120	408	1	101	389	1	**266**	**413**	**0**

ing different sets of input features, i.e., gate types, COP testability measures [13], and shortest distances to PIs (see Section III-A). This experiment runs ATPG on the 100 hardest-to-detect faults in a subset of ISCAS'85 [17] and ITC'99 [18] benchmark circuits, namely, c6288, b04, c432, b08, b03, and b01: this restricted choice in faults and circuits is created by limited computational resources. The results random-guided PODEM and of ANN-guided PODEM trained with various combinations of input features (and with no training) are given in Table I.

The results of these ATPG runs show several trends. First, using an "untrained" ANN is comparable to random backtracing, which indicates training an ANN is an absolute requirement. Second, ANNs trained with a subset of features do improve backtracing quality compared to random backtracing, but there is no clear indication of one subset of features outperforming the others. Third, using all input features outperforms all other configurations, often substantially: this implies there is a way to combine features into a backtracing heuristic to obtain superior results, but that combination may be impossible without the assistance of MI.

B. ATPG performance for hard-to-detect faults

This second experiment shows the performs of the ANN with all input features compared to conventional COP-based and distance-based heuristics, when used in PODEM, on the same 100 hard-to-detect faults. Table II shows these results in terms of CPU time, number of backtraces, and number of backtracks. "MAR" shows substantial improvements for several benchmarks, and when it does not perform the best, its detriments are marginal.

C. ATPG performance for all faults

This third experiment performs ATPG on all checkpoint stuck-at faults in 21 evaluation circuits and 3 training circuits from the ISCAS'85 [17] and ITC'99 [18] benchmarks. Because PODEM takes exorbitant time to run exhaustively, some faults may be aborted. A suitable per-fault time limit is used to produce similar fault coverage with all heuristics.

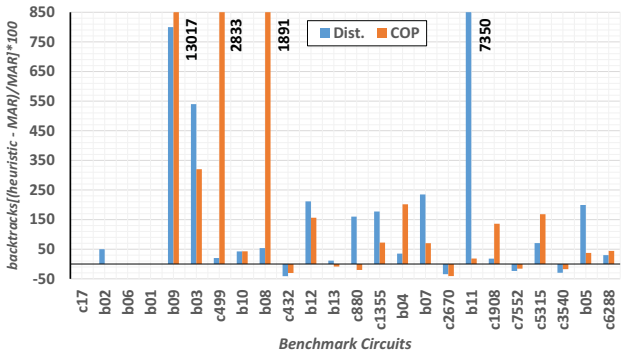

Fig. 6. When comparing the increase in backtracks when conventional heuristics are used for backtracing compared to an ANN, it is clear that using an ANN provides consistent and often substantial benefits. Positive bars indicate fewer backtraces by MAR over conventional heuristic.

Figures 6, 7, and 8 plot the backtracks, backtraces, and CPU times of PODEM using the distance (Dist.) heuristic, the COP heuristic, and ANN-guided heuristic (MAR), respectively. Results are shown as percentage increase compared to the MAR heuristic, i.e., positive results favor MAR. Circuits are ordered by increasing depth.

From these figures, one can conclude that ANN-based backtracing consistently decreases backtracks, often substantially, but one can potentially see a drawback of ANN-based backtracing worth addressing. First, a reduction in backtracks did not consistently translate to a reduction in backtraces: given that fewer backtraces can also reduce ATPG time (i.e., by finding a test in fewer PI assignments), training the ANN to reduce backtraces may be beneficial. Second, the impact on CPU time for MAR in Figure 8, although frequently positive, is not as positive as the impact on backtracks, which implies that CPU time of evaluating the ANN is significant.

VI. CONCLUSION AND FUTURE DIRECTIONS

This study explores the effectiveness of ANN heuristics in ATPG. Results show using ANN-based backtracing, "MAR", most often reduced backtracks and CPU time compared to other conventional heuristics, which suggests ANNs can assist

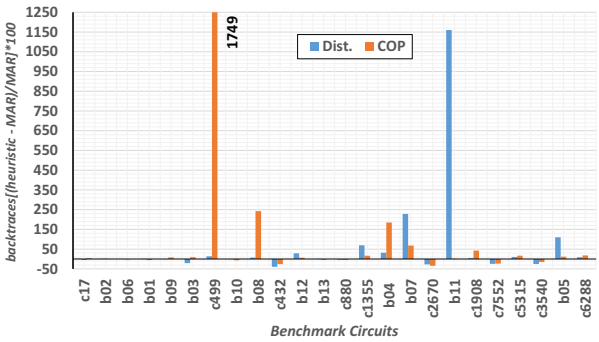

Fig. 7. Total backtraces for checkpoint faults in PODEM ATPG with conventional (distance or COP) heuristic and with ANN guidance (MAR). Positive bars indicate fewer backtraces by MAR over conventional heuristic.

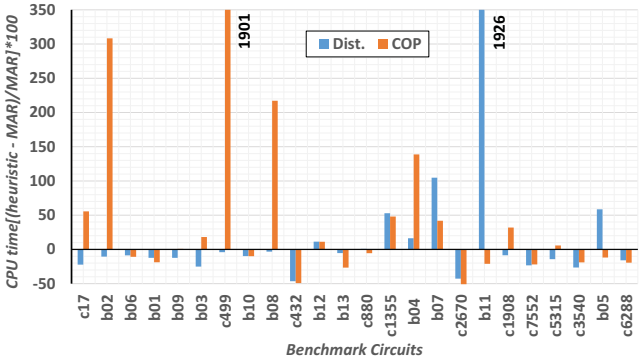

Fig. 8. Total CPU time for checkpoint faults by PODEM ATPGs guided by conventional (distance or COP) heuristics and ANN (MAR). Positive bars indicate lower CPU time by MAR compared to heuristic.

complex EDA problems like ATPG with few drawbacks. A major benefit of the presented MI approach is the ability to combine the benefits of multiple heuristics, which may be difficult otherwise.

Although the ANN-based ATPG is promising, negative results encourage future research directions. First, the training data selection was arbitrary, and it appears selecting "100 hard-to-detect" faults did not provide quality training data: this is indicated by ATPG results on c3540, which were detrimental despite being a training circuit; an algorithm that strategically selects training data may benefit ATPG results, or the ANN may train itself during ATPG (i.e., use recursive learning in lieu of supervised learning) to form a new "evolving" ATPG and create a new market of ever-improving EDA tools. Second, although backtracks were consistently decreased, decreases in backtracks did not directly translate into decreases in CPU time: this may be due to CPU-intensive ANN evaluation, and using novel ANN evaluation techniques may remedy this. Third, the proposed ANN features were rudimentary and limited in scope: expanding the ANN features (like SCOAP [19] and re-convergent fan-out information) or evaluating larger sub-circuits may decrease backtracks further, although this must be balanced with increased CPU time from complex ANN structures. Fourth, given ATPG algorithms beyond PODEM [4], like FAN [20] and SOCRATES [21], further decrease CPU time by adding ATPG subroutines but still require backtracing, "MAR" may improve these algorithms as well.

REFERENCES

[1] O. H. Ibarra and S. K. Sahni, "Polynomially Complete Fault Detection Problems," *IEEE Trans. on Computers*, vol. C-24, pp. 242–249, 1975.

[2] H. Fujiwara and S. Toida, "The Complexity of Fault Detection Problems for Combinational Logic Circuits," *IEEE Transactions on Computers*, vol. 31, pp. 555–560, 1982.

[3] G. Seroussi and N. H. Bshouty, "Vector Sets for Exhaustive Testing of Logic Circuits," *IEEE Trans. Information Theory*, vol. 34, pp. 513–522, 1988.

[4] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, vol. C-30, pp. 215–222, 1981.
in Logic Circuits," *IEEE Transactions on Electronic Computers*, vol. EC-16, pp. 567–580, 1967.

[5] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures

[6] S. T. Chakradhar, V. D. Agrawal, and M. L. Bushnell, *Neural Models and Algorithms for Digital Testing*. Springer, 1991.

[7] M. Chern, S.-W. Lee, S.-Y. Huang, Y. Huang, G. Veda, K.-H. H. Tsai, and W.-T. Cheng, "Improving Scan Chain Diagnostic Accuracy Using Multi-Stage Artificial Neural Networks," in *Proceedings of the 24th Asia and South Pacific Design Autom. Conf. (ASP-DAC)*, 2019, pp. 341–346.

[8] L. R. Gómez and H.-J. Wunderlich, "A Neural-Network-Based Fault Classifier," in *Proc. IEEE 25th Asian Test Symp. (ATS)*, 2016, pp. 144–149.

[9] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu, "High Performance Graph Convolutional Networks with Applications in Testability Analysis," in *Proc. 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[10] Y. Sun and S. K. Millican, "Test Point Insertion Using Artificial Neural Networks," in *Proc. IEEE Computer Society Annual Symp. on VLSI (ISVLSI)*, 2019, pp. 253–258.

[11] S. K. Millican, Y. Sun, S. Roy, and V. D. Agrawal, "Applying Neural Networks to Delay Fault Testing: Test Point Insertion and Random Circuit Training," in *Proc. IEEE 28th Asian Test Symp. (ATS)*, 2019, pp. 13–18.

[12] D. Maliuk, H.-G. Stratigopoulos, H. Huang, and Y. Makris, "Analog Neural Network Design for RF Built-In Self-Test," in *Proc. International Test Conference (ITC)*, 2010, pp. 23.2.1–23.2.10.

[13] F. Brglez, "On Testability Analysis of Combinational Circuits," *Proc. International Symp. Circuits and Systems*, pp. 221–225, 1984.

[14] J. Zhang and A. J. Morris, "Selection of Proper Neural Network Sizes and Architectures - A Comparative Study," *Neural Networks*, vol. 11, no. 1, pp. 65–80, 1998.

[15] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*.

[16] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[17] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Targeted Translator in FORTRAN," *Proceedings of the IEEE Int. Symposium on Circuits and Systems (ISCAS)*, pp. 677–692, June 1985.

[18] F. Corno, M. S. Reorda, and G. Squillero, "RT-Level ITC'99 Benchmarks and First ATPG Results," *IEEE Design & Test of Computers*, vol. 17, pp. 44–53, Jul. 2000.

[19] L. Goldstein, "Controllability/Observability Analysis of Digital Circuits," *IEEE Trans. Circuits and Systems*, vol. 26, pp. 685–693, 1979.

[20] Fujiwara and Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Trans. on Computers*, vol. C-32, pp. 1137–1144, 1983.

[21] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, pp. 126–137, 1988.