Efficient and Effective Neural Networks for Automatic Test Pattern Generation

Lizi Zhang
University of Wisconsin-Madison
Madison, Wisconsin, USA
lzhang697@wisc.edu

Azadeh Davoodi
University of Wisconsin-Madison
Madison, USA
adavoodi@wisc.edu

Abstract

Automatic Test Pattern Generation (ATPG) algorithms such as FAN and PODEM heavily rely on a backtracing step to explore the search space. Conventional implementations often use a single metric such as a testability measure to guide backtracing. Recently, Neural Network (NN) models were proposed which combine multiple metrics to make a better backtrace decision. This paper identifies two fundamental, unresolved issues for effective and efficient use of NNs for ATPG: (1) portability of the NN model across different levels of a combinational circuit; (2) significant runtime overhead when using the NN model in backtrace decisions of each gate. To address these issues, a hybrid approach is proposed which builds and applies the NN model to only selected levels of the circuit. Guidelines to select the level and to train circuits are also discussed in this context. Also, a lookup technique is proposed to reuse the results of prior inferences at each gate to further accelerate the runtime.

CCS Concepts

• **Hardware → Test-pattern generation and fault simulation**.

Keywords

Automatic Test Pattern Generation, Machine Learning, Backtracing

ACM Reference Format:
Lizi Zhang and Azadeh Davoodi. 2024. Efficient and Effective Neural Networks for Automatic Test Pattern Generation. In *2024 ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD '24), September 9–11, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3670474.3685939

1 Introduction

Automatic Test Pattern Generation (ATPG) is an NP-hard problem [10]. Most recent ATPG algorithms are extensions of PODEM [2, 11] which search the space of assignments to Primary Inputs (PIs), or the flipflop content (if adopted for sequential circuits) to reach a reasonable solution quickly. The search space in PODEM-like algorithms is modeled as a decision tree with nodes representing a subset (or worst-case all) of the PIs. The decision tree is determined when trying to find a feasible assignment to activate a fault site and propagating the fault effect to a primary output (PO).

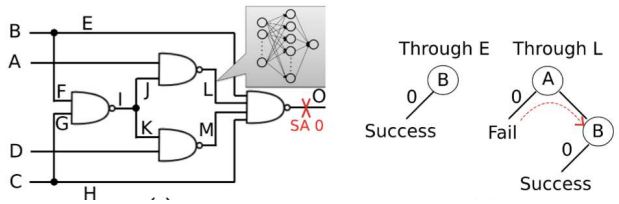


Figure 1: Two possible decision trees for test pattern generation for a stuck-at-0 (SA0) fault. Each tree corresponds to a different backtracing path based on fanin selection at the output gate. Neural networks help pick the best fanin.

A fundamental issue impacting the runtime of PODEM-like algorithms is a backtracing step to build a path from an internal signal line (such as the fault site) to a PI. It requires making backtrace decisions at individual gates to select the "most promising" fanin. Each backtrace path identifies a PI and its assignment, hence the order of backtrace paths impacts the order the PIs are listed in the decision tree. They also impact the number of back*tracks* before the algorithm can terminate, hence the tree size. A back*track* (different from a back*trace*) occurs whenever an infeasible PI assignment is found and requires expanding the tree to explore new directions in the circuit. Figure 1(a) shows an example circuit. Two decision trees are shown in (b) corresponding to different backtrace paths based on fanin selection at the output gate. In the left one, no backtracks happen by selecting signal E which assigns $B = 0$. However, in the right tree, signal L is initially selected which causes a backtrack. The above example shows that the core part impacting the runtime of a PODEM-like algorithm is correctly identifying the fanin to backtrace at a single gate to avoid backtrack when reaching the PI.

Different heuristic metrics may be used to make a backtracing decision, e.g., selecting the fanin with the best testability measure such as COP [1], SCOAP [7], and CAMELOT [12]. Recently, a neural network (NN) model was used to combine different metrics such as testability measure and information about the gate type as input features, with the goal to improve backtracing at a gate [14–16]. The NN model was integrated with PODEM to make inference calls to the NN for each backtrace decision. The runtime benefits were when the number of backtracks was significantly reduced to compensate for the overhead of inferences. Later on, unsupervised learning techniques were proposed [16, 17] which first applied Principal Component Analysis (PCA) to deal with correlated input features. The top two principal components were used as input features to create a smaller NN model. Overhead was associated due to performing PCA before each inference. Moreover, PCA was not compatible with the 'one-hot' format needed to communicate the gate type information. As a result, the one-hot gate type vector which carried important information for making a backtracing decision was dropped as a feature in the unsupervised approaches.

MLCAD '24, September 9–11, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0699-8/24/09
https://doi.org/10.1145/3670474.3685939

In this paper, besides the above-listed areas of improvement, we identify two fundamental issues to make NN-guided backtracing effective and efficient for ATPG. These two are: (1) portability of a NN model across different levels of a combinational circuit; (2) significant runtime overhead of NN inference calls at each gate throughout the course of the algorithm. First, we show there exists significant variations in the values of some input features (i.e., testability measures) based on the level of a gate in the circuit. We show building a single NN model across all levels cannot improve ATPG due to this variation. There is also no runtime advantage to use NN when the level is small or large. We then propose a hybrid approach to train and apply the NN model at only select levels of the circuit. We propose guidelines on how a level can be selected. We also introduce metrics to decide which circuits may be selected to extract data for training purposes. We also propose a lookup technique to speed up the runtime of NN-guided backtracing at each gate. This is based our observation that majority of the inferences at each gate are already-computed values which can be reused. Overall, the summary of our contributions are listed below:

- We identify fundamental issues with NNs to make the backtracing step of ATPG efficient and effective.
- We propose hybrid use of NNs at select levels, and discuss effective selection of training data to address the issues.
- We also show a significant amount of NN computations can be reused at a gate which allows skipping many inferences.
- Our NN model is flexible in receiving different types of input features such as gate type information in the one-hot format.

In our experiments, we integrate our ideas with the FAN algorithm implemented by the open source ATALANTA tool [3]. We show significant improvements in the number of backtracks, with same or better fault coverage and/or ATPG effectiveness, compared to [3] and a NN-guided backtracing using supervised learning.

2 Related Work

One of the earliest ATPG algorithms is Roth's D-Algorithm [4] which defines the D algebra. It is a complete algorithm with the search space size of 2^N where N is number of all signal nodes in the circuits. Later, PODEM [11] improved it by searching assignments for only the PI nodes. It decreased the size of search space to $2^{\#PI}$ and incorporated use of heuristic measures to guide its backtracing. Next, FAN [2] was proposed which compared to PODEM, it detected conflicts much faster and reduced the cost of backtracks.

Early work since 90s showed attempts to learn from data to improve ATPG. The work SOCRATES [9] used static and dynamic learning to accelerate ATPG. The works [6, 8] were among the first to use information from previously detected faults by introducing an E-frontier. The first explicit application of NNs in ATPG was [18] which used a bi-directional binary NN.

Recent works have used NNs to improve backtracing in PODEM [14–17] by effectively combining multiple heuristic metrics [5, 13]. Both supervised learning [14, 15] and unsupervised learning [16, 17] were applied. The work [14] used a two-layer NN with a single output to combine multiple heuristic metrics including gate type, circuit level, COP controllability and observability. The NN model was called for any (not-yet implied) fanin of a gate, for each gate on a backtraced path. The output of the NN represented probability of successful backtrace if a particular fanin of a gate was selected. The fanin with highest probability was selected for backtracing. The NN model was called for every fanin of a gate in every backtraced path in the circuit. Training data was collected from different levels of a circuit in a mixed fashion.

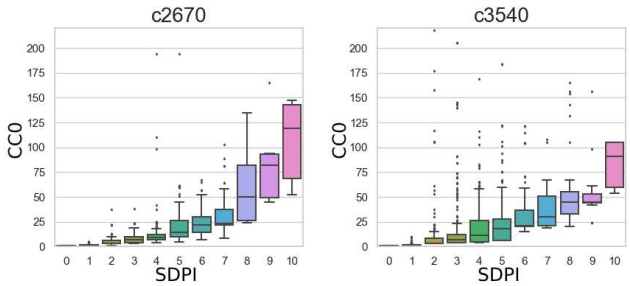


Figure 2: Statistics of SCOAP Combinational Controllability 0 (CC0) as a function of the Shortest Distance to the PIs (SDPI).

3 Effective Neural Network-Guided Backtracing

3.1 Our Observations

The NN-based backtracing techniques described earlier suffer from a number of challenges which we have identified and listed below:

Variations in signal line features in the same circuit: Signal line features such as SCOAP testability and COP are important inputs to the NN model. However, they can significantly vary as a function of distance to PIs with some features not having any guaranteed range. For example, combinational controllability (CC0 and CC1) are much smaller when a line is close to the PIs (i.e., more controllable) and greater near the POs. This means the knowledge derived from data near the PIs cannot be directly transferred to circuit lines near POs. Figure 2 shows an example of the ranges of CC0 as a function of Shortest Distance to any PI (SDPI) for two circuits. As can be seen, for the same circuit, there is significant variation in the values of CC0. Training a NN by directly using these values across all levels won't be effective.

Variations in signal line features across circuits: The previous issue also exists from one circuit to another even when considering the same circuit level. This is primarily due to the distinct characteristics exhibited by individual circuits. For example, as seen in Figure 2, for SDPI=10, the highest CC0 is about 150 in c2670 while it is about 100 in c3540. We note the variation seen at the same level across different circuits is significantly lower than variations across different levels even in the same circuit.

Not all signal lines can benefit from NN models: Cost of a backtrack is low for a line which connects to a small number of PIs (i.e., close to the PIs). The search space for these lines is so small that even considering an exhaustive approach, the cost is still lower than the cost of computing the output of a simple NN model. Moreover, the lines near PIs have very similar feature values, e.g., similar distances to PIs, controllability measures. The high resemblance of data make it hard for NNs to learn and be useful in such scenarios. For the lines near PO, there is not much benefit to use NN to decide backtrace at a gate because they have a huge search space and all fanins of a gate are likely to lead to a successful backtrace. Using NN-guided backtracing is not beneficial in these cases.

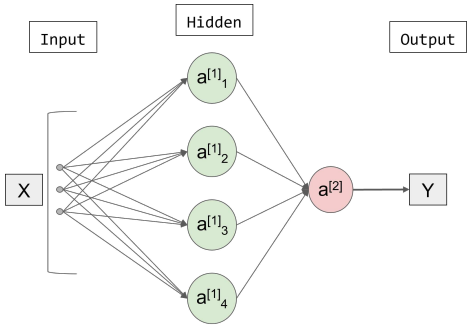


Figure 3: Single-layer NN used for integration with ATPG.

3.2 Proposed Techniques

To develop an ATPG algorithm which effectively uses neural networks, it is important to acknowledge that while sophisticated NN models can result in better performances, they may not be worth the runtime overhead of additional inferences.

Based on the presented observations, in this work, a single hidden-layer NN model is found to be most effective for ATPG. Shown in Figure 3, the model consists of input features, a hidden layer, and an output neuron. The input features are denoted as $X \in \mathbb{R}^{d_1}$ and the hidden layer neurons are $A \in \mathbb{R}^{d_2}$, where d_1 is the number of input features and d_2 is the number of neurons in the hidden layer. The output Y is a real value between 0 and 1. The output works as a measurement of how likely this line may backtrace successfully. The forward propagation is expressed as:

$$A = f(W_1^\top X + b_1) \tag{1}$$

$$Y = f(W_2^\top A + b_2) \tag{2}$$

where $W_1 \in \mathbb{R}^{d_1 \times d_2}$ and $W_2 \in \mathbb{R}^{d_2 \times 1}$ are the weight matrices, and $b_1 \in \mathbb{R}^{d_2}$, $b_2 \in \mathbb{R}$ are the biases, f is the activation function implemented as the sigmoid function.

3.2.1 Hybrid Implementation and Acceleration. In our hybrid implementation, the NN model is trained on data generated for specific levels and is only applied to the same corresponding levels of a circuit. For the remaining levels, the default metrics in the ATPG tool are used to decide the fanin during a backtrace. This addresses the issues discussed earlier: Since the model is trained on and applied to the same level, the features are always comparable to each other.

Figure 4 shows how a circuit level may be identified for hybrid implementation. The plot shows average percentage reduction in number of backtracks and in runtime as a function of circuit level (SDPI). This is compared to a non-hybrid version of ATPG when NN is not used at all. For each SDPI, a separate NN was trained using circuit data for only that level. The trained NN is only used for that level. As can be seen SDPI=6 results in the most reduction in runtime which is used in our experiments. At this level the backtrace can always benefit from the NN model such that the computation cost of NN is quiet often smaller than the cost of backtracks.

To further accelerate NN-guided backtracing, we observed that some circuit lines may be backtraced many times during the course of the algorithm (as the PI assignments change or due to signal implications). Therefore, a lookup table is established to record the NN model outputs for these lines to avoid redundant computations.

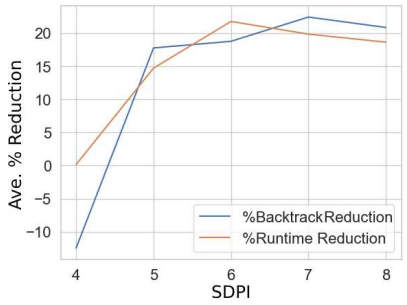


Figure 4: Average reduction in backtrack and in runtime as a function of circuit level (SDPI). We observe SDPI=6 results in the most overall reduction in runtime.

3.2.2 Input Features and Normalization. In this work, we adopt a broad range of input features for NN training. Whenever possible, data corresponding to a feature is normalized to ensure the gradient descent is much more stable and converges faster. Our considered input features are listed below:

- Shortest and Longest Distance to the PIs denoted by SDPI and LDPI, respectively. The LDPI feature is normalized to [0,1] for all gates with same SDPI, as explained earlier.
- SCOAP controllability measures, i.e., Combinational Controllability 0/1 (CC0 and CC1), which represent the difficulty of setting a line to 0 or 1, respectively. These measures contain valuable information in the paths containing the considered lines to the PIs. The SCOAP contrallability measures are also normalized between [0,1] for all gates with the same SDPI.
- COP controllability for a line is the probability of the line being 1 if the PIs are uniformly and randomly set to 0 or 1. This feature provides another approximation of how hard it is to observe a specific signal on a line. They are already in the [0,1] range so normalization is not needed.
- Encoded gate type. The gate type that connects to a line has a great impact on the backtracing process because it may directly impact the values that should be taken by the inputs. A gate type is encoded as one-hot vector. For example, "AND" gate is encoded as 0000001 and "OR" gate is encoded as 0000010. In our work, we consider 7 gate types.

Example: Consider the circuit in Figure 1. The features corresponding to line L are (LDPI, SDPI, CC0, CC1, COP) = (2, 1, 4, 2, 0.625). The gate type is "NAND", hence the one-hot vector 0100000.

Data Collection Process: To generate training data, a default ATPG algorithm (e.g., ATALANTA tool [3] in this work) was applied to several circuits. Both successful and failed backtracing instances were recorded. Meanwhile, the features of lines were also recorded during backtracing. When backtracing is completed (which could be either successful or fail), the same label is assigned to all lines on the same backtracing path. Specifically, if the backtracing results in PI assignment without any conflicts, it is considered successful and the related lines are labeled '1'. Otherwise, it is a fail and backtrack is needed so all lines on the backtracing path are labeled '0'.

It is very possible that a line may be recorded many times during the course of the ATPG algorithm run. The frequency of the labels for the same line represents the likelihood of successful backtracing if this line is chosen.

Example: Consider line O in the same example circuit. Table below shows the generated data (prior to normalization) for a successful backtrace to detect stuck-at-0 fault at this line.

Line	COP	LDPI	SDPI	CC0	CC1	Type	Label
O	0.902	3	1	7	2	0100000	1
L	0.625	2	1	4	2	0100000	1
J	0.750	1	1	3	2	0100000	1
F	0.500	0	0	1	1	0100000	1
B	0.500	0	0	1	1	0100000	1

3.3 Circuit Selection for Training

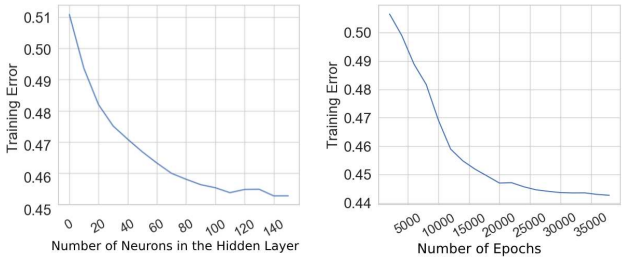
In practice, given a collection of circuits, a subset should be selected for training with the rest to be used for testing. To decide which circuits are suitable for training, the number of samples and the ratio between positively and negatively -labeled data are important. Positive and negative samples have 1 and 0 labels, respectively. If the number of training data points is small, the model will not function properly. Also, if the training data is highly imbalanced (as far as ratio of positive to negative samples), the model tends to have great bias. Table below shows data statistics that are collected from different circuits during the ATPG process.

	Signal Lines with SDPI=6				
Circuit	#Samples	#Pos	#Neg	Ratio	#Samples/Ratio
c1908	318	242	76	3.18	99.87
c1355	259	208	51	4.08	63.50
c2670	366	256	110	2.33	157.27
c3540	191	152	39	3.90	49.01
c5315	269	258	11	23.45	11.47
c6288	84	62	22	2.82	29.81
c7552	907	786	121	6.50	139.63

For each circuit, columns 2, 3, 4, 5 correspond to number of samples, positive samples, negative samples, and ratio of larger to smaller samples. The smallest value that the ratio can take is 1 when the number of positive and negative samples are equal to each other. This is ideal for training to have equal number of positive and negative samples. Column 6 reports total number of samples divided by the ratio in column 5. Column 6 may be used as a metric to decide if a circuit is a good candidate to select for training the NN. The higher number indicates a better candidate due to higher number of samples and/or sample ratio closer to the minimum (equal positive and negative samples). In this work, we used data from c1908 and c2670 for NN training. These two had the highest value in column 6 (except c7552 which we reserved for testing due to its larger size).

3.4 NN Design for Better Training

The main design decision for the NN is number of neurons in the hidden layer. We decide this number by looking at how the training error is impacted. As shown in Figure 5(a), models trained tend to have lower training error as the number of neurons in the hidden layer increases. But the improvement becomes trivial when the number of neurons is beyond 60, accompanied by an increase in the inference time. Therefore, in this work, the NN is designed to have a hidden layer with 60 neurons.



(a) (b)

Figure 5: Training error as function of (a) number of neurons in the hidden layer and (b) number of training epochs

In the figure, the training error is measured using the Binary Cross Entropy (BCE) which is defined as follows:

$$loss(X, y) = -[y \cdot log(f(X)) + (1 - y) \cdot log(1 - f(X))]$$

where X is the input feature, y is the label and $f(X)$ is model output.

Figure 5(b) shows the training error as a function of number of epochs. We set the number of epochs to 30000 to achieve the best training quality. Since we apply a pre-trained model, this number is not considered a significant overhead in runtime.

4 Simulation Results

To show the impact of our NN-guided ATPG, we compared the following approaches in our experiments.

1. **FAN**: We used the open-source ATALANATA [3] tool which implements FAN using LDPI (Longest Distance to any PI) as heuristic measure for fanin selection during backtracing.
2. **NN-Hybrid**: This is the hybrid NN-guided approach proposed in this work. Specifically, we implemented a hybrid version using FAN as the base with the changes discussed in Section 3: The NN model was applied only to lines with SDPI=6. The model was designed and trained as presented in Section 3.4. For the remaining lines, we used the default SCOAP controllability measures for backtracing.
3. **NN-All**: This approach similar to NN-Hybrid, except that the ML model is applied on *all* circuit lines. The main difference compared to NN-All is that the NN model was trained using data extracted across *all* levels. This approach is essentially our best effort implementation of [14, 15] based on available information. We note, all configuration setup is same between NN-All and NN-Hybrid approaches.

The following metrics were used for evaluation:

- **Fault coverage** defined as percentage of detected faults compared to total number of faults.
- **ATPG effectiveness** defined as number of detected faults and number of faults identified as untestable divided by total number of faults. Untestable faults (a.k.a redundant) are typically a small percentage of total number of faults in practice but may take a significant portion of the ATPG runtime because they often require building a complete decision tree (exhausting all test patterns) to conclude a fault is untestable.
- **Number of backtracks** reported both as absolute and as a percentage improvement relative to the FAN case.
- **Runtime** reported in seconds.

Table 1: Comparison for the 'all-faults case' when stuck at 0/1 faults are injected on *all* lines. Parameter K is the backtrack limit.

		K=1000			K=10000			K=25000		
		FAN	**NN-All**	**NN-Hyb**	**FAN**	**NN-All**	**NN-Hyb**	**FAN**	**NN-All**	**NN-Hyb**
c1908	FaultCov.	99.52	99.52	99.52	99.52	99.52	99.52	99.52	99.52	99.52
	ATPG-Eff.	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	#BTracks	1124	1037	550	4818	1037	550	4818	1037	550
	%BTracks	0.00	7.74	51.07	0.00	78.48	88.58	0.00	78.48	88.58
	Runtime	0.08	0.05	0.03	0.25	0.05	0.03	0.23	0.05	0.02
c2670	FaultCov.	95.74	95.74	95.74	95.74	95.74	95.74	95.74	95.74	95.74
	ATPG-Eff.	99.02	99.02	99.02	99.16	99.16	99.16	99.16	99.16	99.31
	#BTracks	31643	32743	31648	266551	273924	252629	611551	618924	518561
	%BTracks	0.00	-3.48	-0.02	0.00	-2.77	5.22	0.00	-1.21	15.21
	Runtime	0.58	0.60	0.62	4.32	4.43	4.28	9.03	9.63	7.40
c3540	FaultCov.	96.00	96.00	96.00	96.00	96.00	96.00	96.00	96.00	96.00
	ATPG-Eff.	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	#BTracks	181	232	176	181	232	176	181	232	176
	%BTracks	0.00	-28.18	2.76	0.00	-28.18	2.76	0.00	-28.18	2.76
	Runtime	0.10	0.12	0.10	0.10	0.10	0.12	0.08	0.13	0.12
c5315	FaultCov.	98.90	98.84	98.90	98.90	98.84	98.90	98.90	98.84	98.90
	ATPG-Eff.	100.00	99.94	100.00	100.00	99.94	100.00	100.00	99.94	100.00
	#BTracks	116	3235	87	116	30235	87	116	75235	87
	%BTracks	0.00	-2688.79	25.00	0.00	-25964.66	25.00	0.00	-64757.76	25.00
	Runtime	0.20	0.25	0.15	0.15	0.92	0.17	0.17	2.05	0.18
c6288	FaultCov.	99.38	99.17	99.56	99.41	99.41	99.56	99.41	99.41	99.56
	ATPG-Eff.	99.82	99.61	100.00	99.85	99.85	100.00	99.85	99.85	100.00
	#BTracks	15563	40835	6405	122384	197953	40774	302384	452212	85774
	%BTracks	0.00	-162.39	58.84	0.00	-61.75	66.68	0.00	-49.55	71.63
	Runtime	0.58	1.23	0.38	4.10	4.87	1.58	10.00	10.65	3.27
c7552	FaultCov.	98.16	98.07	98.17	98.16	98.07	98.17	98.16	98.07	98.17
	ATPG-Eff.	99.18	99.09	99.19	99.18	99.09	99.19	99.18	99.09	99.19
	#BTracks	99330	112005	79508	990330	1102005	781508	2475330	2752005	1951508
	%BTracks	0.00	-12.76	19.96	0.00	-11.28	21.09	0.00	-11.18	21.16
	Runtime	2.58	3.25	2.15	26.28	29.02	16.57	60.50	73.55	40.08
	Ave. Imp. #BTracks		-481.31%	26.27%		-4331.69%	34.89%		-10794.90%	37.39%
	Ave. Imp. Runtime		-5.69%	21.75%		-77.19%	26.27%		-190.66%	26.89%

4.1 Comparison in the All-Faults Case

Here, we consider the case when stuck-at-0 and stuck-at-1 faults are injected on all lines of each circuit. The three approaches were applied on the two training circuits (c1908, c2670) and four testing circuits (c3540, c5315, c6288, c7552), from the ISCAS85 suite, similar to [14, 15]. The approaches were also compared for different backtrack limits (denoted by K). Test pattern generation procedure terminates for each fault when the backtrack limit is reached. The results are shown in Table 1. Average improvements in number of backtracks and in runtime are reported for NN-All and NN-Hyb relative to FAN. We make the following observations:

- In terms of number of backtracks, NN-Hyb on-average has reduction of 26.27%, 34.89%, and 37.39% with increase in K. Higher K allows more time to detect a fault which makes NN-guided backtracing to have more improvement.
- Reduction in runtime is a direct consequence of reduction in number of backtracks. On-average reduction in runtimes are 21.75%, 26.27%, and 26.89% with increase in K.
- Fault coverage is always the same or even better (for c6288 and c7522) in NN-Hyb compared to the other approaches.
- ATPG effectiveness is always same or better. Specifically, for c6288, NN-Hyb achieves a fault coverge of 100% indicating all detectable and redundant faults are identified.
- The NN-All approach has worse performance. (The number of backtracks are only reduced in NN-All for c1908 compared to FAN.) Our NN-All is identical to NN-Hyb (so they are optimized extensively and in the same way) except that the neural network in NN-All is trained with data extracted across all levels.

4.2 Comparison for Hard-to-Detect Faults

In this experiment, we consider hard-to-detect (H2D) faults. To identify these, we first ran FAN with a list of all possible stuck-at-faults (0 and 1 stuck-at faults for each signal line). Next, FAN generates a list of aborted faults which are the ones it was not able to generate a test pattern with its default backtrack limit. These are also known as aborted faults. They may be detectable if a higher backtrack limit is given or they may be inherently untestable. In this experiment, we only used the circuits which had more than 10 H2D faults (i.e., more than 10 aborted faults). We additionally experiment with a higher backtrack limit (K=50000).

For each circuit, we report the number of H2D faults (aborted faults generated by FAN with K=10). Next, after running each approach with a higher backtrack limit (K=25000 and 50000), we also report the number of redundant and number of aborted faults. The results are reported in Table 2. We make the following observations from the table:

Table 2: Comparison for hard-to-detect (H2D) faults

		K=25000		K=50000	
		FAN	NN-Hyb	FAN	NN-Hyb
c2670	#H2D Faults	31	31	31	31
	#Redu+#Abor	8+23	16+15	8+23	20+11
	ATPG-Eff.	25.81	51.61	25.81	64.52
	#BTracks	598971	517111	1173971	818979
	Runtime	9.15	7.72	17.78	13.32
c6288	#H2D Faults	24	24	24	24
	#Redu+#Abor	0+12	0+8	0+12	0+8
	ATPG-Eff.	50.00	66.67	50.00	66.67
	#BTracks	302344	253872	602344	503872
	Runtime	9.98	3.83	20.03	7.85
c7552	#H2D Faults	68	68	68	68
	#Redu+#Abor	6+62	6+62	6+62	6+62
	ATPG-Eff.	8.82	8.82	8.82	8.82
	#BTracks	1550170	1550546	3100170	3100546
	Runtime	35.38	29.22	71.90	58.27
	Ave. Imp. ATPG-Eff		**44.43%**		**61.11%**
	Ave. Imp. #BTracks		**9.89%**		**15.52%**
	Ave. Imp. Runtime		**31.55%**		**34.95%**

- NN-Hyb performs significantly better than FAN as far as resolving the number of H2D faults. First, it performs better in terms of detecting a higher number of redundant/untestable faults. For example, in c2670 and for K=25000, out of 31 H2D faults, the number of aborted faults were reduced from 23 (in FAN) to 15 (in NN-Hyb). The number of faults identified as redundant/untestable was increased from 8 (in FAN) to 16 (in NN-Hyb). Therefore, NN-Hyb identified more faults as redunant and finished with fewer aborted faults. This behavior holds consistently and for both values of K.
- NN-Hyb has same or higher ATPG effectiveness for the same value of K. This means NN-Hyb is able to detect same or higher number of H2D faults (given that it identifies higher faults as redundant and has fewer aborted faults).
- Finally, in terms of runtime and number of backtracks, NN-Hyb is significantly better. For example, the average improvement in runtime over FAN is 31.55%and 34.95% for K=25000 and 50000, respectively. The average improvements in ATPG effectiveness were 44.43% and 61.11% with increase in K.

Note, in c7552, 62 of the H2D faults remain aborted for both FAN and NN-Hyb. Further experiments (even with backtrack limit of 10 million) suggest that these faults are highly unlikely to be detected. However, there is significant reduction in runtime in NN-Hyb compared to FAN for each K.

4.3 Impact of the Lookup Approach on Runtime

In our last experiment, we disable the lookup-based acceleration in NN-Hyb and compare the runtime with the case when it is enabled. The results are reported in Table 3. Recall, the lookup-based acceleration records NN inferences for each signal and only computes inferences if they have not been computed before. As can be seen, on average 30.33% improvement in runtime can be achieved. The dynamically-generated lookup table effectively eliminates the need for forward propagation computations of the NN model in the test pattern generation process, greatly reducing the computational efforts.

Table 3: Runtime of NN-Hyb with and without acceleration

	NN-Hyb (K=10000)	
	w/o lookup	w/ lookup (%Impr.)
c1908	0.067	0.033 (51%)
c2670	13.350	4.283 (68%)
c3540	0.117	0.117 (0%)
c5315	0.200	0.167 (17%)
c6288	1.867	1.583 (15%)
c7552	23.867	16.567 (31%)
Ave. Imp.		**30.33%**

5 Conclusions

In this work we identified the issues with existing NN-guided ATPG algorithms. We proposed a hybrid procedure for applying NN to select circuit levels, with guidelines for level selection, and training of the NN. We also proposed a lookup technique to reuse the existing computations in a NN for higher speedups. Our simulation results conducted for two cases of "all circuit faults" and "hard-to-detect faults" demonstrated the effectiveness of our procedure in reducing the number of backtracks, with same or better ATPG metrics.

6 Acknowledgments

This work is supported by a grant from National Science Foundation under Award No. 2322713.

References

[1] F. Brglez. 1984. On Testability Analysis of Combinational Circuits. *IEEE International Symposium on Circuits and Systems* (1984), 221–225.
[2] Fujiwara and Shimono. 1983. On the Acceleration of Test Generation Algorithms. *IEEE Trans. Comput.* C-32, 12 (1983), 1137–1144.
[3] H. K. Lee and D. S. Ha. 1993. *On the Generation of Test Patterns for Combinational Circuits.* Technical Report No. 12-93. Department of Electrical Engineering, Virginia Polytechnic Institute and State University.
[4] J. P. Roth, W. G. Bouricius, and P. R. Schneider. 1967. Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits. *IEEE Transactions on Electronic Computers* EC-16, 5 (1967), 567–580.
[5] J. Patel and S. Patel. 1985. What Heuristics are Best for PODEM? *First International Workshop on VLSI Design* (1985), 1–20.
[6] K. T. Cheng. 1991. On Removing Redundancy in Sequential Circuits. *ACM/IEEE Design Automation Conference* (1991), 164–169.
[7] L. Goldstein. 1979. Controllability/Observability Analysis of Digital Circuits. *IEEE Transactions on Circuits and Systems* 26, 9 (1979), 685–693.
[8] M. L. Bushnell and J. Giraldi. 1997. A Functional Decomposition Method for Redundancy Identification and Test Generation. *Journal of Electronic Testing* 10, 3 (1997), 175–195.
[9] M.H. Schulz and E. Auth. 1989. Improved Deterministic Test Pattern Generation with Applications to Redundancy Identification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 8, 7 (1989), 811–816.
[10] O. H. Ibarra and S. K. Sahni. 1975. Polynomially Complete Fault Detection Problems. *IEEE Trans. Comput.* C-24, 3 (1975), 242–249.
[11] P. Goel. 1981. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Trans. Comput.* C-30, 3 (1981), 215–222.
[12] R.G. Bennetts, C.M. Maunder, and G. D. Robinson. 1981. CAMELOT: a Computer-Aided Measure for Logic Testability. *IEE Proceedings E - Computers and Digital Techniques* 128, 5 (1981), 177–189.
[13] S. Patel and J. Patel. 1986. Effectiveness of Heuristics Measures for Automatic Test Pattern Generation. *ACM/IEEE Design Automation Conference* (1986), 547–552.
[14] S. Roy, S. K. Millican, and V. D. Agrawal. 2020. Machine Intelligence for Efficient Test Pattern Generation. *IEEE International Test Conference* (2020), 1–5.
[15] S. Roy, S. K. Millican, and V. D. Agrawal. 2021. Training Neural Network for Machine Intelligence in Automatic Test Pattern Generator. *IEEE International Conference on VLSI Design and International Conference on Embedded Systems* (2021), 316–321.
[16] S. Roy, S. K. Millican, and V. D. Agrawal. 2021. Unsupervised Learning in Test Generation for Digital Integrated Circuits. *IEEE European Test Symposium* (2021), 1–4.
[17] S. Roy, S. K. Millican, and V. D. Agrawal. 2022. Multi-Heuristic Machine Intelligence Guidance in Automatic Test Pattern Generation. *IEEE Microelectronics Design & Test Symposium* (2022), 1–6.
[18] S. T. Chakradhar, V. D. Agrawal, and M. L. Bushnell. 1991. *Neural Models and Algorithms for Digital Testing.* Springer.

A Artifact Appendix

A.1 Abstract

This project proposes a hybrid approach for Automatic Test Pattern Generation (ATPG) by integrating machine learning with the FAN algorithm. It includes implementations of two approaches, namely, NN-Hyb and NN-All. These two apply neural network models at selective and at all circuit levels, respectively. Project files support the paper by providing the source code, circuit files, and scripts needed to reproduce key results, including comparisons for "all fault case", and "hard-to-detect faults". Comparison of runtime with and without a proposed acceleration scheme is also included. Minimal requirements include a C++ compiler and shell script execution capabilities.

A.2 Artifact check-list (meta-information)

- **Algorithm:** A novel Neural Network guided ATPG algorithm
- **Program:** Linux 5.15.153.1
- **Compilation:** g++ 11.4.0.
- **Data set:** ISCAS85 Benchmark circuits
- **Binary:** A binary file is provided.
- **Model:** NN-networks used in our paper are integrated in the source code.
- **Runtime environment:** Ubuntu 22.04.2 LTS.
- **Metrics:** Fault Coverage, ATPG Effectiveness, number of backtracks and runtime
- **Output:** Numerical results in Tables I, II and III
- **How much time is needed to prepare workflow (approximately)?:** 1 hour
- **How much time is needed to complete experiments (approximately)?:** 30 minutes
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:**: MIT license
- **Workflow framework used?:** No, but scripts are provided.
- **Archived (provide DOI)?:** 10.5061/dryad.m0cfxppcm

A.3 Description

A.3.1 How to access. All necessary source code, circuit files, and scripts to reproduce key results are available on the GitHub page: https://github.com/lzzh97/NN-for-ATPG. Clone the repository and follow the instructions in the README file.

A.3.2 Hardware dependencies. Standard computers without any specific hardware dependencies.

A.3.3 Software dependencies. C++ compiler and shell script execution capabilities.

A.4 Installation

In the paper we have compared the following 3 approaches:

(1) FAN: base approach for ATPG used for comparison.

(2) NN-Hyb: our approach based on applying neural network (NN) model during backtrace at select circuit levels during ATPG.

(3) NN-All: alternative approach using our NN model, but at all levels of the circuit.

To get an executable for (1), please download source code from https://github.com/hsluoyz/Atalanta and compile. (We did not include it because it is already published by another group.)

For (2) and (3), please compile (using the make command) from the NN-Hyb and NN-All directories, respectively.

A.5 Experiment workflow

We recommend using the provided scripts to reproduce the results, which can be easily executed by the command `bash [path to script]`. The scripts in the "script" directory generate the results reported in Tables I, II, and III:

(1) **All Faults Case** (Table I): `AllFault_FAN.sh`, `AllFault_NN-All.sh`, `AllFault_NN-Hyb.sh`. Table I reports fault coverage, ATPG effectiveness, number of backtracks and runtime. These quantities are reported by our program for NN-All and NN-Hyb.

For FAN's results, the tool does not directly report ATPG effectiveness. Compute it using this formula: (#redundant_faults + #detected_faults)/#total_faults.

(2) **Hard-to-Detect Faults** (Table II): `H2D_FAN.sh`, `H2D_NN-Hyb.sh`. The results show the performances of FAN approach and NN-Hyb approach on Hard-to-Detect faults with default backtracking limits 25,000 and 50,000.

(3) **Acceleration with Lookup Approach** (Table III): `Acceleration_NN-Hyb.sh`. This script generates results for NN-Hyb with and without acceleration.

A.6 Evaluation and expected results

For scripts given in Section A.5 (excluding FAN) should print out results of following format:

```
*********************************************
Name of the circuit           : "c2670"
Fault coverage                : 0.000 %
Total number of backtrackings : 517111
ATPG Effectiveness            : 51.613 %
Total time                    : 7.550 Secs
```

The results, except for total time, should closely match our report. The variation in total time is typically less than 10%.

A.7 Experiment customization

Once compiled, use the command from each directory. Users are allowed to apply our approach to a customized dataset with different parameters by using command line

```
cd ./NN_Hyb # directory for desired approach
./atalanta -Options [path to circuit file]
```

The options needed to generate similar results of the paper are listed below:

-w: Disable lookup table (enabled otherwise if -w not specified)

-b n: Maximum number of backtracking (default: -b 10)

-f fn: Faults are read from the file fn. This options is only used to generate the results in Table II in the paper. Otherwise, if a fault file is not specified, the input is assumed to be all possible faults.

A.8 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- http://cTuning.org/ae/submission-20201122.html
- https://github.com/ml-eda/artifact-evaluation/