Toward Zero Backtracks in Test Pattern Search Algorithms with Machine Learning

by

Soham Roy

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
August 7, 2021

Keywords: Automatic test pattern generation (ATPG), Artificial intelligence (AI), Artificial neural network (ANN), Digital testing, Heuristics in algorithms, Principal component analysis (PCA)

Approved by

Spencer K. Millican, Chair, Assistant Professor of Electrical and Computer Engineering
Vishwani D. Agrawal, Co-chair, Professor Emeritus of Electrical and Computer Engineering
Adit D. Singh, Godbold chair, Professor of Electrical and Computer Engineering
Victor P. Nelson, Professor Emeritus of Electrical and Computer Engineering
Sanjeev Baskiyar, Professor of Computer Science and Software Engineering

Abstract

A digital circuit with n primary input lines has $N = 2^n$ possible input vectors and a test vector that detects a fault in the circuit may be among those 2^n n-bit combinations. A pure random test generator can generate these test vectors, but this is inefficient due to its random generation of n-bit combinations. Various testing algorithms were developed and implemented over six decades to overcome this inefficiency. Classic algorithms like D algorithm, PODEM, and FAN laid the foundations over which other algorithms were built upon to improve the search time for test vectors.

Because the search for tests for hard-to-detect faults in a circuit has exponential complexity, test generation, whether performed randomly or algorithmically, is computationally expensive. Backtrack is one of the essential activities in ATPG algorithms that directly impacts search time. In colloquial terms, backtrack means the algorithm took a bad decisions when determining which circuit inputs should be set to achieve an objective to find a test vector. Contemporary algorithms use various circuit topological information and testability measures as heuristics to reduce backtracks and improve search time. Patel and associate concluded from their experiments that rather than using a single testability measure with a high backtrack limit, it is more efficient to use multiple testability measures successively with lower backtrack limits. However, the use of multiple testability measures successively as ATPG heuristic still remains quite expensive in test generation time.

To address unmanageable time complexity, engineers often rely on human "hunches" and heuristics learned through experience. Training machines to adopt these human skills is known as machine learning (ML) or machine intelligence (MI). This dissertation examines MI for its ability to enhance automatic test pattern generation (ATPG) by the combining

circuit topological information and testability measures as a novel heuristic to reduce backtracks. Instead of a conventional heuristic to guide backtracing directions, this work uses MI algorithms. The guidance can come from unclassified data in which we find patterns – known as unsupervised learning – or it can come from a database of training problems with desired outcomes, known as supervised learning. The ML framework applied to ISCAS'85 and ITC'99 benchmark circuits showed significant improvements in ATPG performance as reduced backtracks and computation time. Initial experiments found a significant decrease in computation time and backtracks with basic MI structures and training.

In this research, a PODEM ATPG program is implemented with ML-based guidance for backtraces. Initially, basic trained-ANN guidance is found to reduce backtracks and CPU time over any single heuristic guidance. Then, an optimally-trained-ANN guidance enhances the ATPG performance. Next, principal component analysis (PCA) combines several heuristics to train the ANN. The PCA-trained-ANN guidance produced the best ATPG performance.

To the divine power of this universe

For being beside me invisibly and help me to evolve consciously

In the loving memory of my grandparents: (Dadu, Thamdadu, Dida, Thamma)

For their blessings, love, and faith

To my beloved parents: (Ma and Baba)

For their constant love, affection, and support

Acknowledgments

Through the words of ***The Beatles*** "The long and winding road ...", I could see the entire two years and 11 months of doctoral life as a journey. Many people came, became a part, and left my side, but few people stood by me along with my ups and downs of life. I am truly honored to be surrounded by great friends, motivating advisors, an incredible group, and a family that filled my heart with umpteen joy and happiness. In the late evening of the snowy winter of Dresden, Germany, 2018–I wrote an email to Prof. Vishwani D. Agrawal regarding my Ph.D. at Auburn University, USA. Later, I got an interview email from Prof. Spencer K. Millican that turned out to be positive. On 9th August 2018, I landed in the United States and started my research journey with dreams, aspirations, passion, and insanely hard work. I thank the Electrical and Computer Engineering (ECE) department (Prof. Mark Nelms and Prof. Stuart Wentworth) for supporting my graduate study by providing me a Teaching Assistantship (TA), without which I would not have been able to complete my Ph.D. work successfully. I am immensely thankful to Prof. Ujjwal Guin and Prof. John Y. Hung for allowing me to assist as TA in their respective (ELEC 4200 and ELEC 3040/3050) laboratories. I am also thankful to Prof. Sanjeev Baskiyar for becoming a University Reader of my dissertation.

I want to bestow my regards to all my committee members. Without their support, my journey till now and in the future would not have been possible. It is my honor and pleasure to have Prof. Spencer K. Millican and Prof. Vishwani D. Agrawal as my advisors who helped me grow technically, spiritually, and professionally. During my initial days here at Auburn, I got the opportunity to nurture my skills and have a deeper understanding of VLSI design and test through our untimely discussions with Prof. Adit D. Singh. My research path demanded learning Automatic Test Equipment (ATE). Prof. Victor P. Nelson was too

generous in providing the knowledge transfer added with a hands-on experience. Without which it would have been difficult for me to be an expert in ATE in that brief period.

I thank Prof. Spencer K. Millican for encouraging me to write C++ codes for ATPG tools. We had our disagreements, technical and casual discussions over a cup of coffee. Because of him, I got to know variants of coffee flavor. Prof. Millican's reviews were genuinely helping me evolve my English language, for which I shall be truly grateful. I was, and I will be surprised by how he thinks while coding ahead of time.

I thank Prof. Vishwani D. Agrawal for encouraging me to **THINK** technically and non-technically. His constant attitude of challenging me to push the limits and seek out bold, new ideas. His enthusiasm for research and learning is infectious, which is something I hope I have picked up and would like to embody in my entire life. I am a firm believer of **THINK->EXECUTE->PUBLISH**, and he supported my philosophy throughout my doctoral journey. I thank Prof. Prathima Agrawal for giving words of advice and encouragement as and when required.

I thank all my friends (Yang, Joshua, Pouyan, Ayokunle, Krishna, Ziqi, Yuqiao, and Wendong) who have supported me over the years and have provided me constant encouragement and moral support. Office staff in ECE, namely Mary Lloyd, Linda Allgood, Faith Fain, John Tennant, Linda Newton, and L. Autry May were accommodating throughout my stay in Auburn. I especially thank my best friend, Joshita Majumdar, without whom I could not have successfully done many thought experiments in my Ph.D. that led to the enlightenment of many facts in my research. She gave me suggestions and solved technical problems (both coding and writing) and was like a giant tree beside me who always supported and believed in my mission and vision.

For all that I have learned, everything that I have gained, whatever I have achieved - I owe my deepest thanks and express my gratitude to my parents and grandparents for their constant love, support, and encouragement. Without you, I could not have been so happily motivated.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial neural network
ATPG	Automatic test pattern generation
CNN	Convolutional neural networks
COP	Controllability and Observability Program
DT	Decision tree
DUT	Design under test
EST	Equivalent State Hashing
FAN	Fanout-oriented Test Generation
GRASP	Generic Search Algorithm for Satisfiability Problem
HS-Trees	Half-space trees
IC	Integrated circuit
LR	Linear regression
LSTM	Long short-term memory
MARS	Multivariate adaptive regression splines
MI	Machine intelligence
ML	Machine learning
mRMR	Minimum redundancy maximum relevance

MSE	Mean square error
OPTICS	Ordering points to identify the clustering structure
PAM	Partitioning around mediods
PC	Principal components
PCA	Principal component analysis
PCC	Pearson correlation coefficient
PI	Primary inputs
PO	Primary outputs
PODEM	Path-oriented decision making
RF	Random forest
RNN	Recurrent neural netowrks
s-a-0	Stuck-at-0
s-a-1	Stuck-at-1
SAF	Stuck-at fault
SAT	Satisfiability
SCOAP	Sandia Controllability/Observability Analysis Program
SOCRATES	Structure-oriented Cost-reducing Automatic Test Pattern Generation System
SOMs	Self-organizing maps
SVD	Singular value decomposition
SVM	Support vector machine

TEGUS	Test Generation Using Satisfiability
VLSI	Very large scale integration

Chapter 1

Introduction

Circuit testing is a critical part of the integrated circuit (IC) manufacturing process that prevents the release of defective circuits, and it plays a pivotal role in maintaining trade-offs between IC quality and manufacturing cost. An exemplary IC manufacturing test can avoid shipment of bad ICs to the customer, and a poor IC manufacturing test can deteriorate the quality and increase the manufacturing cost of the ICs. This increased IC manufacturing cost must then be recovered from increased manufacturing and shipment costs of good ICs [4].

The central principle of testing is applying stimuli to manufactured ICs to excite and detect defects created during silicon manufacturing. Binary patterns (or test vectors) are applied to the circuit's inputs, and then the response of the circuit is compared against the expected response. If the response matches, one may say the circuit is free of any defects. Therefore, the quality of testing can be measured by calculating the percentage of the modeled defects detected by the test vectors. Therefore, a high quality test makes less likely to ship a defective circuit.

The cost of testing circuits is a significant portion of IC manufacturing costs [5], and as transistor density continues to scale upwards, circuit test costs are increasing and efforts continue to keep these costs down. Testing costs are increasing for two reasons: test generation and test application. Test generation comprises computer programs that algorithmically generate tests for the device under test (DUT), and this cost is incurred only once when developing the circuit. In contrast, test application costs are repeated for each manufactured device, and the total cost is proportional to the number of circuits manufactured. The challenge of circuit test with regards to these two costs is to reduce costs while simultaneously **1)** preventing the release of defective circuits and **2)** not discarding good devices.

For the earliest generations of ICs, designers and test engineers resorted to functional testing that was not fully automated, yet it was useful for exhaustively testing small to medium-sized ICs. However, problems of functional testing became prolific when the logic functions become large and complex. Let us illustrate this problem with the testing of a ten-input AND function. Suppose an input pattern 0101010101 is applied and 0 is observed at the output of the function. One may conclude that gate under test is neither a NAND nor an OR function because it violates the function of NAND and OR functions. Again, an input pattern 1111111111 is applied to ensure the gate under test is not a NOR function because it violates the function of NOR. Still, one can not guarantee the given gate under test will function correctly as an AND gate for all $2^{10} = 1024$ possible input patterns. But, one can conclude that the gate with ten inputs under test is AND by checking each entry of the truth table, and although it is possible with ten inputs, such a test will be too long and impractical for an actual circuit with many input lines.

In 1959, Eldred [6], Galey *et al.* [7, 8], and several others established structural testing built around the stuck-at fault model and developed automatic test pattern generation (ATPG) algorithms to create test vectors. ATPG is a classic very large scale integration (VLSI) testing problem. Typically, the problem can be formulated as, "a fault is given, find a test." Since the first digital circuit was created, several testing methods were developed to test if the logic worked in an intended way. As digital circuits became complex following the trend of Moore's law [9], there was an impetus to research various ATPG algorithms that can find efficient test vectors.

Several noteworthy theoretical studies [10–12] show that test generation for combinational circuits belongs to the class of problems called NP-complete, suggesting that no test generation algorithm with a polynomial computation time complexity is likely to exist. One may have to try all possible circuit input vectors to find a test for a fault, but this is impractical for modern, large circuits, making ATPG difficult. In practice, the worst-case time complexity of test generation for a circuit is non-polynomial (exponential), and test

generation algorithms can achieve slower time growth by using *heuristic* search techniques or subroutines in ATPG. A sub-step in many ATPG algorithms is to trace backward (i.e., "backtrace") from an interior location and select a circuit's input to assign to remedy the exponential complexity of test generation algorithms. This choice may or may not lead to a test, and in the latter case, one may have to *backtrack* to undo the circuit's input assignment. To reduce the possibility of backtracking, heuristics based on designer's intuition choose tracing directions when backtracing.

Creating an effective heuristic for all situations in ATPG may be difficult, but recent advances in MI can create effective heuristics with minimal programmer effort. MI is a category of algorithms that are automatically programmed based on past experience. It is best applied to problems that cannot be solved by deterministic decisions or, more specifically, where the designer's deterministic decisions cannot be easily programmed. Recent studies applying MI to circuit testing [13] have yielded improved algorithm output quality and reduced algorithm CPU time, and I foresaw the same advantages for ATPG.

The novelty in this dissertation entails replacing conventional heuristics of an ATPG algorithm with new heuristics based on circuit topology and testability measures combined through MI. MI techniques – artificial neural network (ANN) and principal component analysis (PCA) – guide backtracing decisions in ATPG that traditionally relied on a human-dictated heuristic. Results show fewer backtracks, reduced CPU time, and potential for exploring and combining a greater variety of ATPG inputs.

This dissertation is divided into eight additional chapters. Chapter 2 introduces the standard terms of ATPG, the key concepts behind generating tests, including the fault sensitization and propagation process, and ATPG algorithms' evolution. Chapter 3 introduces the idea of MI, an introduction to ANNs and PCA, prior work in the application of MI to testing, and how MI is currently applied to ATPG.

Chapter 4 examines MI's ability to enhance ATPG by reducing backtracks. In lieu of a conventional heuristic to decide backtracing directions, this chapter uses an 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., Controllability and Observability Program (COP)), ATPG performance is comparable to conventional PODEM, and using multiple features further reduces backtracks and ATPG CPU time. This chapter establishes the feasibility of "basic trained-ANN guidance" capability for ATPG [2].

Chapter 5 presents a training method for ANNs for ATPG. Unlike in Chapter 4, ANNs combine any number of known multiple ATPG inputs, such as input-output distance (logic depths), gate type, and testability measures like COP, as heuristics and guide PODEM to find tests with reduced backtracks in reasonable CPU time, but under ad-hoc training methods, some circuits obtained degraded fault coverage; the unaddressed challenge was finding useful data that improved ATPG performance for such circuits. The proposed training method in this chapter recursively collected data on hard-to-detect faults and discarded data that did not improve ANN quality; the method both optimized ANN hyperparameters, and the resulting ANN reduced backtracks. This chapter develops the concept of "optimally-trained-ANN guidance" for ATPG [3].

Chapter 6 introduces unsupervised learning that can combined any number of known ATPG inputs, such as input-output distance (logic depths), and testability measures like COP and Sandia Controllability/Observability Analysis Program (SCOAP) values through PC analysis, and then the major PC can guide ATPG choices. Some ATPG inputs data were re-calculated and two major PCs were obtained. These PCs guided backtrace directions in a PODEM ATPG program, and for most circuits, the number of backtracks either matched the best of the three heuristics or was lowered. This chapter introduces the principal component analysis (PCA) to combine multiple heuristics for "PCA-guidance" in ATPG [14].

Chapter 7 introduces Chapter 5's ANN feature reduction methodology to improve the ANN complexity and guide decisions that otherwise rely on heuristics. The ANN analyzed

gate types, logic depths, fan-outs, and testability measures to choose backtracing directions and to make circuit input assignments. When treating these values as a multivariate statistics problem, extracting the PCs, and re-training the ANN with PC as features, the complexity of the ANN is reduced and ATPG efficiency is enhanced. This chapter combines the techniques investigated in the previous three chapters. Here, the ATPG relies on "PCA-trained-ANN guidance". The ANN is trained with sample ATPG data combined with several principal components derived from the heuristic data of training circuits. The result, not surprisingly, is the best achieved in this research so far [15].

Finally, Chapter 8 summarizes this dissertation and discusses open challenges yet to be addressed by ML in testing. Chapter 9 concludes this dissertation with possible future research directions that may explore the latest computing technologies–such as quantum algorithms–to apply to test generation problems.

Chapter 2

Automatic Test Pattern Generation – History and Challenges

This chapter introduces standard terms used in ATPG research, the critical concepts of ATPG, and ATPG algorithms' evolution over the past six decades. Specifically, this chapter informs the reader about the typical VLSI testing-related standard terms required to understand the ATPG process. The following section gives a synopsis of the fault detection processes in ATPG and the final sections explain the important ATPG algorithms, their problems, solutions to fix them, and detail on using similar solutions in other ATPG algorithms.

In the past, the design and test of digital ICs was classified as a tedious process, and a significant part of this process was test generation, due to its manual nature. Various ATPG algorithms played a critical role in amplifying the quality of manufactured ICs. These algorithms primarily focused on different ways to reduce the number of test patterns needed to test a digital circuit: these patterns are the combination of binary logical patterns assigned to circuit inputs needed to detect faults in the circuit, also known as *tests*. Computers use ATPG algorithms to find tests from all possible patterns by using various circuit-related information to determine the circuit inputs' values that form tests. These algorithms became more efficient and robust with more research, and with more ways to use the circuit information.

2.1 Standard Terms in ATPG

Before going into more detail, one must be familiar with the standard terms in ATPG used frequently throughout this dissertation.

- **0**: – A logic value 0 in both the faulty and fault-free circuits.

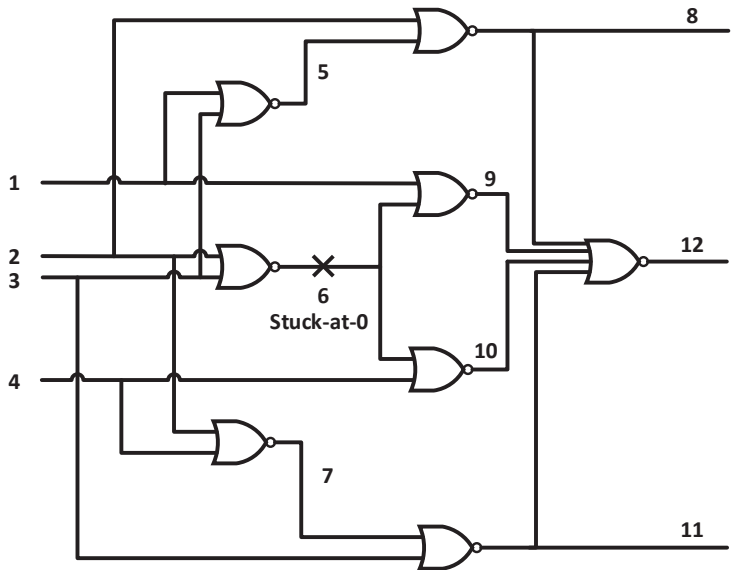

Figure 2.1: The Schneider circuit [1].

- **1**: – A logic value 1 in both the faulty and fault-free circuits.
- **Fault**: When a failure in a circuit causes a difference in its node's logical value.
- **D**: – A logic value 1 in the properly functioning circuit, but a logic value 0 in the faulty circuit.
- $\overline{\mathbf{D}}$: – A logic value 0 in the properly functioning circuit, a logic value 1 in the faulty circuit.
- **X**: – Unassigned or unknown value.
- **Schneider circuit**: A circuit with reconverging fanout named after the engineer Peter R. Schneider, as shown in Fig. 2.1. In 1967, J. P. Roth used this circuit to find tests for those failures that are not detected by other old sensitized path algorithms.
- **Circuit graph**: A representation of circuit (e.g. see Fig. 2.2) comprising of nodes and edges, as shown in Fig. 2.3.
- **Primary inputs (PIs)**: The externally accessible input pins of a circuit-under-test through which logical signals can be stimulated.

- **Primary outputs (POs)** : The externally accessible output pins of a circuit-under-test through which logical signals can be observed.
- **Goal**: The node of interest in a circuit which can hold a desired value.
- **Search space**: A set of all possible input test patterns that may be assigned to primary inputs of a circuit.
- **Search graph**: A tree structure (see Fig. 2.4) built using all possible choices for circuit input patterns.
- **Backtrace**: Moving a goal backward from a goal until a PI is reached and a logical value is assigned to the primary input.
- **Backtrack**: In the search graph of a circuit, when a conflict occurs because a goal cannot not be satisfied, and it must be resolved by alternatively assigning previously assigned PIs.
- **D-Frontier** [16]: The set of digital gates that have one or more stuck-at values (D or $\overline{D}$) on their inputs and an X (or unknown) value on their output.
- **Fanout**: A circuit node that drives inputs of multiple digital gates.
- **Fault coverage**: The number of faults detected by a set of test patterns, expressed in percentage of all faults.
- **Reconvergent fanout**: When branches of two or more fanout nodes reconverge as inputs to a gate. The gate is called the reconvergence point, while the fanout node is called the reconvergent fanout.
- **Free line**: A circuit node that has no reconvergent fanout nodes among its predecessors (i.e., any path from gate inputs to any PI).

- **Headline**: A free line that drives a reconvergent fanout node. In other words, the root of a tree of free lines in the circuit.
- **Implication**: The process of determining the values implied by already assigned values in a circuit.
- **Implication Stack**: A data structure that keeps track of all the implied signals and also records assigned PIs that have possible alternative values [16].
- **Justification**: The process of assigning PI values to fulfill a goal; essentially similar to backtrace with conflict resolution.
- **X-path-check**: Verifying whether at least one D-frontier gate can have its D/U input reach a PO. Otherwise, the algorithm will backtrack.
- **Single-path sensitization**: Choosing/sensitizing a path from the origin of a fault to the circuit output by assigning values to gates' inputs along the path such that the effect of the fault must propagate to the output.
- **Single fault assumption**: One and only one fault is present in a circuit at a time.
- **Stuck-at fault**: A fault model in which a fault site has a permanent binary value (0/1) due to the presence of a fault.
- **Heuristics**: Any approach to solving a problem using experiences to develop solutions that may not be optimal.
- **Backward implications**: Determination of a logic gate's possible inputs for given desired output value.
- **Forward implications**: When a logic gate's inputs' are significantly labeled so that logic gate's output can be determined.

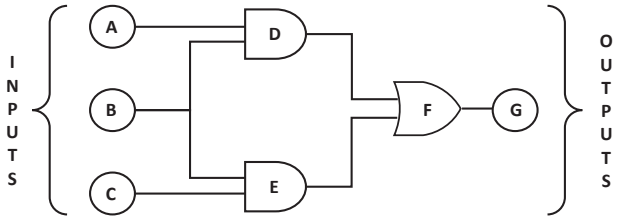

Figure 2.2: A combinational sample circuit comprises 3 PIs, 3 digital logic gates, and 1 PO.

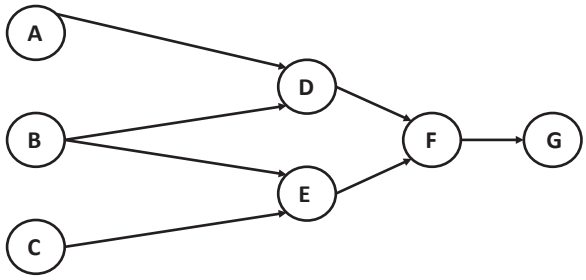

Figure 2.3: A circuit graph comprises nodes to represent PIs, digital logic gates, and PO in a circuit, shown in Fig. 2.2.

- **Branching operation**: An algorithm to determine which input variable will be set to what value (0 or 1) at each level of the binary decision tree.
- **Bounding operation**: An algorithm to avoid exploring large portions of a binary decision tree by restricting the search decision choices due to unnecessary exploration of the tree.

2.2 Critical Concepts of ATPG

A digital circuit model is used to illustrate the generation of circuit tests using fault sensitization and propagation methods. Consider the sample circuit as shown in Fig. 2.2, also illustrated by the graph in Fig. 2.3. The nodes in the graph represent digital gates, and graph edges represent interconnects/wires. The inputs and outputs to and from the circuit are represented by PIs and POs, respectively. PIs are the only places where test patterns can be applied, and POs are the only places where the effects of the test patterns can be observed.

Test pattern generation is the process of finding an input test pattern set that thoroughly tests the circuit. These test patterns are known as a *test set*, and the set causes all faulty circuits to respond differently from good ones at the POs. In other words, a failure (i.e., a difference between a node's expected value and a node's actual value) is present when at least one PO's logical value is different from its expected value. A test set must consist of

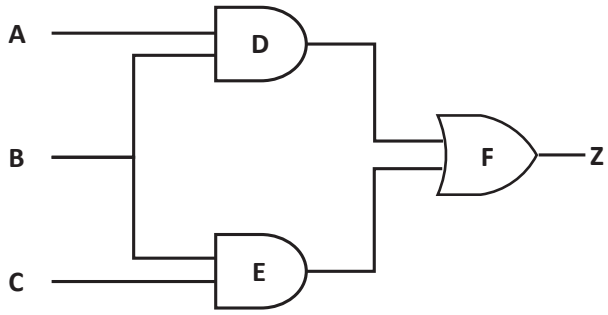

Figure 2.4: A good circuit.

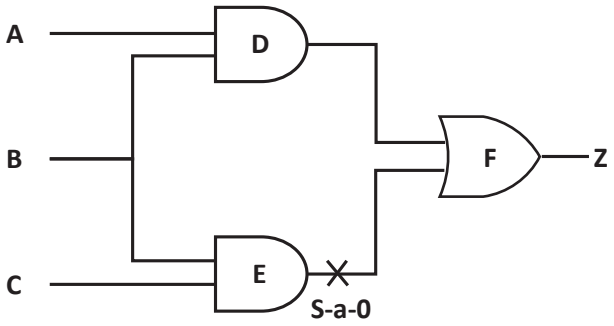

Figure 2.5: A faulty circuit.

binary patterns that generate difference at a fault location's logic value, also known as *fault sensitization*. This difference in fault location's logic value must also be propagated/revealed to one or more POs, also known as *fault propagation*. During sensitization and propagation, the internal signal assignments are justified by assigning binary logical values to PIs to generate the test set.

Test pattern generation time is the biggest challenge for larger circuits, and therefore testing introduces a few simple assumptions about the type and frequency of faults. Test generation assumes stuck-at faults (SAF) are the only failures present in a circuit. This type of fault considers itself a node in the circuit that permanently assumes 0 or 1. Depending upon the node's value, i.e., if it is 0, known as stuck-at 0 (s-a-0) and if it is 1, known as stuck-at 1 (s-a-1). It is assumed thay only one SAF is present in a faulty circuit. This single SAF in a circuit may be impractical in reality, but it has been useful in practice because detecting a large amount of single SAFs will eventually result in a test set that detects a high percentage of all defects [17,18]. Therefore, test pattern generation can be redefined as the task of generating a test set that detects all detectable single SAFs.

Consider an example of a good circuit, shown in Fig. 2.4, having no fault on the gate E's output, and the same circuit having a s-a-0 fault on the gate E's output, shown in Fig. 2.5, which is known as a faulty circuit. A test pattern can be found for the good circuit that produces 1 at gate E's output. In the faulty circuit, gate E's output is s-a-0, which is opposite to the good circuit value. To set 1 at the gate E's output, B and C must be set to 1 to

sensitize the fault since the faulty and fault-free circuits have different values at the fault location. The fault sensitization procedure can be summarized as:

- Set B to 1.
- Set C to 1.

The next step is to propagate and observe the difference produced by the fault sensitization process to the PO. Since the output of gate F is connected to a PO, the input from gate D must be set to 0 to allow the s-a-0 fault on the input of gate F to propagate through it. To produce a 0 at the output of gate D, one of its inputs must be set to 0, and since B is already set to 1, this means A must be set to 1. The fault propagation procedure can be summarized as:

- Set D to 0.
- Set A to 0.

2.3 ATPG and its Evolution

As the fault sensitization and the fault propagation processes are familiar, the next step is to automate them, and ATPG algorithm achieves this automation. ATPG is a set of tasks performed to generate test patterns to test a circuit [4]. The problem of ATPG complexity can vary from pseudo-random PI assignments to complicated searching techniques, but most algorithms have the following steps in common:

1. Pick a not-yet detected fault.
2. Develop a search graph for a circuit with the picked fault.
3. Search the search space until one of the following conditions is met:
   (a) a test is found.

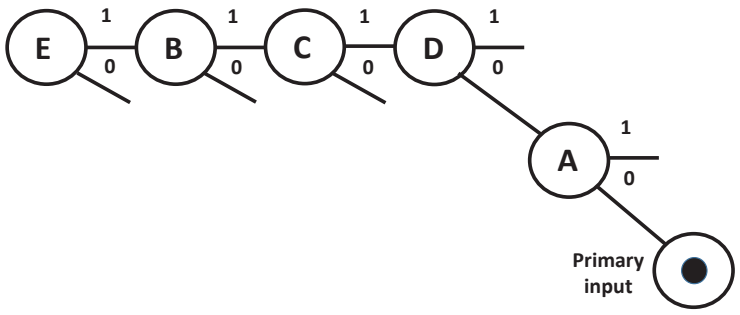

Figure 2.6: A search graph.

(b) the search space is exhausted.

If the algorithm stops at step 3(a), the test is obtained with the target fault and expected output. If the algorithm stops at step 3(b), then no test exists and the corresponding fault is *undetectable* or *redundant*.

In this dissertation, I have considered two ATPG algorithms for discussion: the D algorithm and Path-Oriented Decision Making (PODEM) algorithm. The representative algorithms differ in the following ways: 1) the size of the search space, 2) their search strategy, and 3) heuristics used to guide the search. The size of the search space and the search order are essential since they determine the upper bound on the search time and the impact on the search time, respectively. Therefore, the search strategy and heuristics used to guide the search are equally important to the search space size.

2.3.1 The D Algorithm

Roth's D algorithm [1] was the revolutionary algorithm that conceptualized ATPG by defining the five values in D algebra (see Section 2.1) and by giving a complete search algorithm (i.e., it finds all possible tests). Earlier work on ATPG by [19] failed to generate tests for the Schneider circuit [1] using the single-path sensitization process. Later, the work of [20] was successful and gave a textual background of the testing aspects of the D algorithm, while Roth gave a detailed mathematical analysis in his first paper [1] and also showed the generation of a test for the Schneider circuit [1]. The D algorithm assigns values

to nodes that depend on their locations in the search space, and the search order depends on the implementation since the strategy was ad hoc for searching order.

In the D algorithm, a fault cone is found by forward tracing the circuit topology from the fault location to the POs of a circuit. These POs form an inward cone towards the PIs that are responsible for activating and propagating the fault. The fault detection sequence is illustrated by taking the steps below:

1. The algorithm initializes all nodes in a circuit to X after constructing the search graph of the circuit.

2. A target fault is chosen and a value D or $\overline{D}$ is assigned to the fault location for a s-a-0 or s-a-1 fault, respectively.

3. To activate the fault, the algorithm finds a non-conflicting set of PI assignments, then one can say the fault is sensitized.

4. The activated fault is propagated to at least one of the POs so that the fault site's value i.e., D or $\overline{D}$ can be observed. To do this fault propagation, the algorithm preserves a list of gates also known as D-frontier and gates are chosen to drive the fault effect to one of the POs.

5. The implications of the above assigned values are propagated forward and backward in the circuit.

6. After this process, the algorithm checks to determine whether a test is generated. If so, the algorithm terminates, otherwise the pursuit of search continues until the algorithm assigns all probable values to every node in the search graph of the circuit.

If a conflict is encountered while assigning values at the circuit's nodes, one may have to backtrack [1]. A backtrack [16] is performed when either one of the two conditions occurs: 1) D-Frontier set becomes empty, implying that there is no way for the fault to propagate to any

PO, or 2) a conflict occurs, meaning that if the objective for a signal is 0 or 1, instead it is set to 1 or 0, respectively. A backtrack is performed by checking if there is any alternative value possible for the gate's inputs and, if so, assign an alternative value and push the assigned gate's input into the implication stack. If there are no more alternatives left, the assigned gate's input is popped off the implication stack and is assigned to X. If the search graph is exhausted, then the fault is redundant and no test exists.

D algorithm is a complete ATPG algorithm, i.e., it finds tests for all possible faults in a circuit. However, this algorithm has downsides. First, it is long, difficult to program, and has high algorithmic complexity (2^n where n is the number of nodes in a circuit) for large circuits as it does forward implications on all internal signals as a part of fault sensitization and propagation processes. Therefore, one may conclude ATPG is an NP-complete problem [10–12], as the search algorithm's complexity increases exponentially with circuit size. Second, it is inefficient for circuits containing XOR gates and re-convergent fanouts [21] as the D algorithm's search is too directionless.

2.3.2 PODEM

Goel observed that the search graph of D algorithm consists of every node in a combinational circuit, and improved the search efficiency by focusing on PIs since all nodes are a function of PI nodes [21]. The main steps of PODEM are as follows:

1. Find a fault cone followed by establishing a D-frontier in the same way as in the D algorithm.

2. Set a line value objective that establishes or advances the D-frontier toward a PO.

3. Backtrace from the objective followed by backward implications in a stack until a PI is reached and assigned a logic value.

4. Perform forward implication.

5. If there is conflict in meeting the objective (expect 1 but get 0, or vice-versa), the D-frontier becomes null, or X-path-check fails, then perform a backtrack on the assigned PIs.

6. Repeat until the D-Frontier reaches a PO, i.e., a test is found, or if backtracking finds that no test is possible, i.e., the fault is redundant.

The computational barrier of the D algorithm was solved by reducing the total search space of a circuit. Suppose there is already a set of PI assignments and another PI is assigned a value to cause conflict. To resolve this issue, one would have to try the complement of the current PI value and determine whether PI assignment will ever meet the goal. If the complementary value also conflicts, then the goal can never be achieved by the existing PI assignment, and one may need to backtrack. One can reduce the search space if such conflicts arise, since there are no other PI assignments left at this point. Overall, the search space is reduced from 2^n, where n is the total number of signals (gates and PI) in the circuit, to $2^{\#PI}$ where #PI signifies number of PIs.

PODEM allows the use of various heuristics to speed up the search for a test and checks to prevent unproductive searches through the following characteristics. First, the concept of X-path-check was introduced. D algorithm may try to find a test even when the entire D-frontier is blocked, but PODEM's X-path-check verifies that there is at least one D-frontier gate with access to a PO. Otherwise, it will backtrack. Second, PODEM originally proposed a heuristic that uses the distance between the PIs and signal sites to identify easy or hard to control inputs of logic gates while backtracing, as opposed to D algorithm which chose a random gate input. Likewise, several other heuristics based on circuit topology were proposed later.

2.3.3 Heuristic Guidance for ATPG

Combinational ATPG algorithms use *branch and bound* searches to find tests for a fault in a circuit by traversing the entire search graph. In a search graph (also known as binary

decision tree), each vertex i.e., a "branch" represents a line (represented by a graph node) in the circuit to binary logic value 0 or 1, which is known as branching. As the size of the circuit increases, the height and width of the search graph can also increase. Therefore, exploring the entire search graph may be impractical for larger circuits and creates trouble when making decisions on what to assign circuit values. The bound operation restricts the search decision space (i.e., length and height of the search graph) by applying suitable heuristics. Several other ATPG algorithms summarized in Section 2.3.4 have a search graph of size $2^{\#PI}$, but they attempt to find tests faster by inserting subroutines to filter the search graph or by using heuristics to more quickly find a test. It is this second aspect of ATPG that I focus on in this dissertation.

2.3.4 Other ATPG Algorithms

The Fanout-oriented Test Generation (FAN) algorithm [22] proposed improvements over PODEM by introducing additional heuristics: immediate implications of signal assignments, unique sensitization, headlines, and multiple backtraces to restrict the search space. Their main contribution is a breadth-first backtrace as opposed to depth-first strategy in PODEM.

Dominator ATPG programs [23] provided further improvement. A *dominator* is a signal through which a fault's effect must pass to reach a PO. Signals controlling dominators within the fault effect cone must be assigned non-controlling values to allow the fault's effect to propagate. These assignments are compulsory and can be determined by circuit topology without search.

The Structure-oriented Cost-reducing Automatic Test Pattern Generation System (SOC RATES) [24–26] program used static and dynamic learning to generate tests. Static learning, a form of preprocessing, assigns all signals with 0/1 and saves implications. The same procedure is used dynamically at each step in the search algorithm to find a test. Dynamically learned ATPG is costly but provides more scope to identify the implied signals that further help to find a test quickly.

Equivalent State Hashing (EST) [27–29] used a form of dynamic programming. This is the only ATPG algorithm that finds a test for a target fault by taking help from the tests of previously detected faults. EST introduced E-frontier, which signifies a sub-set between the circuit lines being already assigned and not assigned, i.e., X (also includes the D-Frontier). E-frontiers are generated at each decision step of ATPG and stored. ATPG continues by comparing the current E-frontiers and prior-learned E-frontiers by a circuit decomposition process. EST also uses multiple parallel backtraces of the ATPG, which eventually speeds up the process.

A recursive learning [30] program was introduced to improve the FAN algorithm by applying SOCRATES-style learning recursively to signals as a part of implications. It has an advantage over SOCRATES due to it's recursive (i.e., repetitive) nature. The test generation time for recursive learning may grow exponentially, but the memory grows linearly with recursion depth.

TRAN [31, 32] formulated ATPG as a Boolean satisfiability (SAT) problem. It uses implication graphs and transitive closure for faster signal assignments and fewer backtracks than other ATPG algorithms. Several other authors have also used satisfiability (SAT) in their ATPG algorithms [33–35]. Larrabee [33, 34] used path variables to find the solution efficiently. Other SAT-based ATPG programs include Generic Search Algorithm for Satisfiability Problem (GRASP) [36], Test Generation Using Satisfiability (TEGUS) [37], and those reported by Henftling *et al.* [38], Larrabee *et al.* [34], and Tafertshofer *et al.* [39].

Chapter 3

Machine Intelligence – General Theory and its Application in State-of-the-Art ATPG

This chapter discusses the general theory of the advance technology of MI (or ML), followed by a listing of a wide variety of computational models of MI. Afterward, I chose two specific models, i.e., ANNs and PCA, to use in my dissertation, thus I present a brief reason behind selecting these MI models in this chapter. The following sections talk briefly about the basic theory of ANNs and PCA, followed by a brief mention of the application of MI in VLSI testing. Finally, this chapter presents an illustration of how MI was used in ATPG a few decades back followed by an overview of the application of ANNs and PCA in my dissertation.

MI describes a broad category of algorithms that automatically program themselves through experience. This experience can come from unclassified data in which the program finds patterns, which is unsupervised learning; or the experience can come from a database of training problems with desired outcomes, which is supervised learning; or the experience can come from iterative re-evaluation and adjustment, which is reinforced learning.

Supervised learning models are trained with inputs where the desired outputs are known. Supervised learning uses patterns to predict labels on unlabeled data and is used in applications where the history of data predicts likely future events. A supervised learning algorithm receives inputs along with corresponding correct outputs; the algorithm learns by comparing its outputs against and correct outputs to find errors and modifies the learning model accordingly to minimize the errors. Popular learning models like support vector machines (SVMs), one-class SVMs and one-class neural networks, decision trees (DTs), random forests

(RFs), linear regressions (LRs), multivariate adaptive regression splines (MARS), logistic regressions, adaboosts, ANNs, convolutional neural networks (CNNs), autoencoders, recurrent neural networks (RNNs), long short-term memories (LSTMs), half-space trees (HS-Trees).

Unsupervised learning models are used when the training data has no history of correct outputs (labels). The goal of the learning algorithm is to explore the data and find some structure or patterns within it. Popular learning models include K-means clustering, partitioning around medoids (PAMs), ordering points to identify the clustering structure (OPTICS), PCA, minimum redundancy maximum relevance (mRMR), self-organizing maps (SOMs). These methods are typically used to segment text topics, classify items, and identify data outliers.

The crux of reinforcement learning is to capture the critical aspects of the real problem that involves a learning agent interacting with its environment to achieve a final goal. Reinforcement learning problems are closed-loop problems because the learning system's interaction with the environment decides its later inputs. However, the learner system is not told about the actions taken in any form of ML but instead must discover actions that give the most reward by trying them out. The characteristics such as being closed-loop effectively, no direct instructions as to what actions to take and their consequences, including reward signals to play out over extended periods,are the three most essential features of a reinforcement learning problem. In a nutshell, reinforcement learning problems pertain to what to do and map situations to corresponding actions to maximize a numerical reward signal.

In this dissertation, ANNs (the supervised learning model) and PCA (the unsupervised learning model) are the learning models that act as ATPG heuristics. ANNs have been demonstrated to be an effective and possible means to solve past unsolvable problems. With time, ANNs have become resilient with the availability of data. These days, ANNs are widely used to solve problems where one may not need an optimal answer, but existing methods to solve the problems are inaccurate or unmanageable to calculate. On the other

hand, PCA became popular as the storage of data become expensive. Data mining and pruning techniques are sought to meet the challenges of storage and computation. Despite the discovery of the PCA almost a century ago [40,41], its demand burgeoned when computer-based applications spread across multiple disciplines.

3.1 Artificial Neural Network (ANN)

In 1943, neuroscientists studied how to mimic biological neurons, and ten years later, psychologist Frank Rosenblatt further improved the idea by developing a single-layer neural network for supervised learning called the perceptron. The end result of this research is ANNs [42], which are MI architectures modeled after the human brain. One of the most common forms of ANN is the "feed forward neural network." It is a popular structure used for solving scientific and engineering problems in which a neural network starts calculating from inputs and works its way to outputs without any loops or other convolutions [43]. The basic process of training an ANN consists of selecting an ANN structure, generating training data by solving sample problems using a known method, and training the ANN with this data. After this, the ANN will be ready to handle problems of a similar nature. In this dissertation, I used an ANN consisting of an input layer, a single hidden neuron layer, and an output neuron in the [0,1] range. This is illustrated in Fig. 3.1. The values of the inputs, referred to as features, $X_0, X_1, \cdots, X_M$, were normalized to [0,1] to facilitate the ANN training. X_0 is fixed at 1.0 and is a neuron called the bias input. Bias inputs can shift the activation function by adding a constant to the inputs. This bias can be thought as analogous to the role of a constant in a linear function where constant value effectively transposes the line. Hidden neurons $Y_1, Y_2, \cdots, Y_N$ and an output neuron Z, called the "label", evaluate their outputs using respective inputs. The output value of a neuron is denoted as x_i, y_i or z. The directed edge from any neuron A to another neuron B, denoted by $w(A, B)$, carries a signed floating point value. The output of any neuron Y_j is calculated as

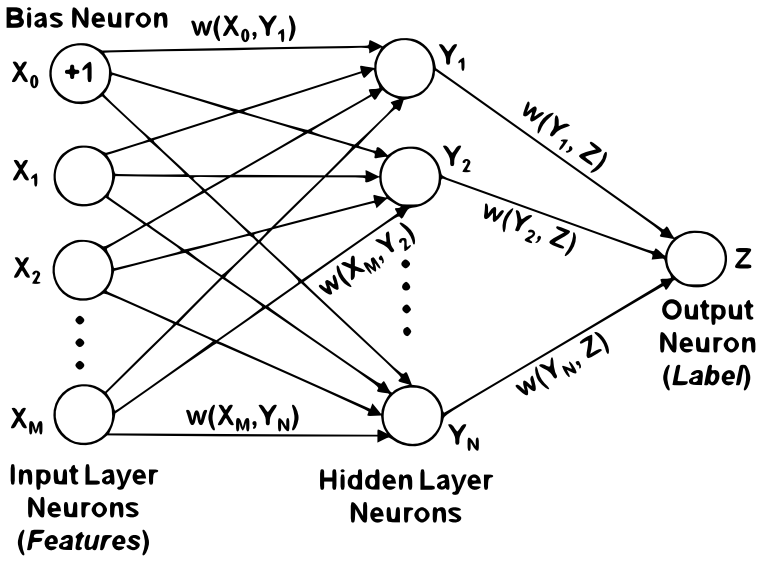

Figure 3.1: A traditional ANN consists of inputs (features), hidden layer neurons, and an output neuron (label), connected through weighted edges.

$$y_j = f(\sum_{i=1}^{M} x_i \times w(X_i, Y_j)) \tag{3.1}$$

where f is the activation function [44] for which a sigmoid function [45] (see equation 3.2) was used.

$$f(v) = 1 - \frac{1}{1 + e^{-v}} \tag{3.2}$$

A training data pattern consists of inputs (features) and expected output (label) values. During training, the output label is computed for the given input features and compared to the expected value of output. The square of the difference between the computed and expected values of output neurons averaged over all training patterns is the mean square error (MSE). Weights are adjusted in successive training "epochs" (i.e., one cycle through the entire training data set) to minimize this MSE. This error can be further minimized by tuning hyper parameters (i.e., any parameter in the ANN configuration that is not directly learnable by training) like the number of hidden layers, number of hidden neurons per hidden layers, learning rates, activation functions, etc. Adam [46] is a typical optimizer prominently

used for feed-forward ANN structures. Subsequent chapters will describe the process of training data generation and applications.

3.2 Principal Component Analysis (PCA)

PCA is a statistical technique that drastically reduces data dimensionality through new variables known as principal components (PCs). Each PC is a linear function of the original data that maximizes the variance of uncorrelated data while preserving statistical information. Evaluating PCs instead of original data narrows down decision making to an *eigenvalue/eigenvector* (also known as eigen decomposition method) or singular value decomposition (SVD) [47]. PCs can be chosen based on either a covariance matrix or correlation matrix, and the choices are independent of any pre-defined functions [47]. PCs represent the same amount of information as the original data, i.e., the original data can be restored from the PCs. Eigen decomposition method is a primitive method (used by [41]). It is fairly simple to understand the mathematical background, but it is complex to code in python. Conversely, SVD is an advanced technique, fairly simple to code, but it is complex to understand its detailed mathematics. SVD avoids potentially serious numerical issues of eigen decomposition methods and therefore is the preferred method. This dissertation uses SVD and the eigen decomposition method to obtain PCs based on a parameter known as *explained variance* [48] and eigen values [47], respectively, to gain the first-hand experience of both the methods.

3.3 Machine Intelligence (MI) Applied to Test

The work in [49–51] showed that MI has been quite popular in the field of research and development of IC testing and provides enough motivation to apply MI to solve the problem specific to this dissertation. These surveys studied ML-based analog and RF circuits testing [52–56], digital circuit testing [2, 3, 13–15, 57–81], memory testing [82–84], recent applications of ML to hardware security, IC counterfeiting, and devices based on emerging

technologies [85–105]. This dissertation discusses about specific ML models (ANN and PCA) used to guide ATPG algorithmic decisions to generate tests for digital circuits.

ANN was used to model a digital circuit where a bi-directional binary neuron represents the state of a signal [81]. In the study [81], the stable state of the network is the minimum energy state, and this energy state was modified with a fault. Thus, finding the minimum energy state gave a test in the form of the states of input neurons of the neural network. This application to ATPG requires either a physical neural network or a software model. In either case, the network energy function depends on many variables, and the function has many local minima, which makes the search for a test (i.e., finding the optimal criterion of the network energy function finds a test) for some faults is rather difficult. Another application of MI was related to the heuristic part of ATPG algorithms that use different heuristics, such as the distance between the PIs and outputs to signal sites, testability measures, voting of digital logic values on fanout stems depending on its branches, and other learning techniques using implication graphs, etc. to speed up the search process. In 1985, Patel and associate [106, 107] conducted experiments to study the effectiveness of different testability measures as heuristics for PODEM and developed a new strategy for test generation. They suggested that rather than using a single testability measure with a high backtrack limit, it is more efficient to use multiple testability measures successively with a lower backtrack limit.

In this dissertation, my approach is entirely different than previous works [106, 107]: I use ANNs and PCA as MI models, I rely on the conventional digital circuit model and I use a search algorithm that–given unlimited computing resources–guarantees a test in significantly reduced CPU time by reducing the number of backtracks. The ANNs and PCA allow me to combine multiple circuit topological information and testability measures, create a novel heuristic, and guide the search toward a test without a backtrack limit as a stopping criterion while avoiding unproductive decisions. To my knowledge, such an approach has never been

investigated. My research uses PODEM as an ATPG algorithm to explore a novel MI-based heuristic for finding the direction of backtrace from a desired objective. ANN-guided PODEM [2, 3] or PCA-trained-ANN guided PODEM [15] can learn from ATPG with any or no fixed set of rules to guide tracing backward and develop an intelligent system to use topological circuit information; this circuit information includes the type of a gate that drives a signal node on which a backtrace is to be performed, the existence of fanouts, and testability measures such as distance [21], COP [108], SCOAP [109]. PCA-guided PODEM [14] can combine any number of known circuit information to develop unsupervised learning-based test generation system. The measure of success of these experiments will be fewer backtracks and reduced CPU time compared to conventional heuristics-based PODEM ATPG.

Chapter 4

Machine Intelligence for Efficient Test Pattern Generation

This chapter proposes replacing conventional backtracing heuristics in ATPG with ANNs. I developed procedures for collecting training data, training an ANN, and integrating an ANN-based heuristic into the ATPG algorithm. My training data contained topological and functional features used by several existing ATPG heuristics and used them to train an ANN that then guided an ATPG to outperform any conventional heuristic. Although I restricted myself to the PODEM [21] ATPG algorithm to demonstrate the efficacy of the ANN-based heuristic, my technique can be applied to any ATPG algorithm that traces through a circuit. The specific contributions of this chapter are:

- A description of ATPG trials to obtain training data and then train an ANN.
- An evaluation of ANN-based ATPG's ability to reduce backtracks and a comparison against conventional heuristics.
- A comparison against the CPU time of ANN-based ATPG against conventional heuristics, which was reduced despite increased tracing complexity.

The remainder of this chapter is organized as follows. Section 4.1 discusses my approach to train an ANN to guide backtracing in ATPG. Section 4.2 gives experimental results on ANNs trained by several sets of training data, selecting one that produced the best PODEM ATPG result, and also evaluates the performance of the selected ANN-based PODEM on benchmark circuits against PODEM using conventional heuristics.

Major portions of this chapter are taken verbatim from my previously published research work [2].

4.1 Modus Operandi

I selected an ANN configuration as discussed in Section 3.1, generated training data by applying a PODEM program that did not use any specific heuristic to a set of circuit inputs, and trained the ANN. The training data contained patterns of input neuron values and the expected output values. Initial part of training determined the appropriate number of neurons for the ANN and the required number of training patterns that minimized the training error.

4.1.1 ANN input features and output label

For a circuit line, the input features of the ANN contained the following:

- 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 found approximated probabilities for all lines [108], and recalculation was not required during ATPG.
- The circuit level of line being traced, i.e., the shortest distance to any PI from the line. This value was normalized in the range [0,1] by the maximum depth of the circuit.

During backtrace, the ANN returned the probability of 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 was evaluated at each circuit input, and the input which was most likely to result in a test was chosen. This was akin to COP-based or level-based easy-hard heuristic [4] that selected the line with the largest/smallest value depending on the desired characteristics.

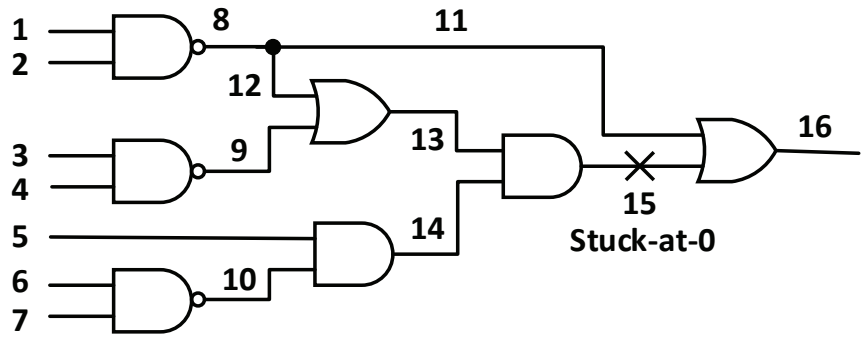

Figure 4.1: Training patterns resulting from PODEM ATPG while generating the test $110X10X$ for line 15 stuck-at-0 fault in Fig. 4.2.

4.1.2 ANN training data generation

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

As an illustrative example, the following steps produced the training data of Fig. 4.2 to detect the line 15 stuck-at-0 fault in the circuit of Fig. 4.1:

- To excite the fault, an objective of "1" on line 15 was set. Backtracing through 14-5 assigned "1" to PI 5. Logic simulation showed that the objective was yet to be met.
- Another backtrace through 14-10-6 assigned "0" to PI 6, but still the objective was not met.
- Backtracing through 13-12-8-1 assigned "0" to PI 1. Simulation verified that the objective of "1" on line 15 was met, but a "1" on line 11 blocked the fault effect from being propagated to the PO. Therefore, a backtrack assigned the alternative value

	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

Figure 4.2: ANN training patterns derived from a ATPG trial for line 15 stuck-at-0 fault in Fig. 4.1.

"1" to PI 1. Thus, the choices used in the previous backtrace through 13-12-8-1 were "failure" patterns with $z = 0$.

- After the backtrack, line 15 became "X", or unknown. To achieve the objective of 1 on line 15, backtracing through 13-9-3 assigned "0" to PI 3. On simulation, the objective of "1" on line 15 was met. The state of line 15 was denoted with D, where D means fault-free state as 1 and faulty state as 0.

- To propagate line 15 value D to the PO, line 11 was now given an objective value "0". A backtracing through 8-2 assigned PI 2 to "1" and a test $110X10X$ was found. Therefore, all backtracing history without a label were assigned "success" labels with $z = 1$.

4.2 Experimental Results

In this chapter, a workstation containing an Intel i7-8700 processor and 8 GB of RAM performed all experiments. Tools were implemented in C++ using the MSVC++ 14.15

compiler with maximum performance optimization, and all ANN activity was executed in Python. PODEM ATPG [21] was reproduced along with event-driven fault simulation [4] such that any heuristic (distance [21], or COP [108]) can be applied across ISCAS'85 [110] and ITC'99 [111] benchmark circuits without favoring a single heuristic. This chapter will surely polarize EDA vendor mindset to deploy MI in their ATPG software. However, EDA vendors hesitate to divulge their program source code, and it is impossible to conduct research-based experiments using the executable. Therefore, I prefer to run the experiments using the in-house EDA tools.

Training

To generate training data, PODEM was run on 100 "hard-to-detect" faults of benchmark circuits: c6288, b05, and c3540. This chapter calls these "training" circuits, while others are called "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 [108] testabilitity values: detection probabilities were calculated as $CC \times CO$ and $(1 - CC) \times CO$ 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.

Figure 4.3 illustrates the impact of training data volume on ANN accuracy (i.e., average training error). As more training patterns were used, the accuracy of the trained ANN increased until a certain point, which indicated that too much training data could negatively impact the accuracy. Following this observation, a set of 3,730,724 training patterns was used.

Figure 4.4 illustrates the impact of the number of hidden layer neurons on ANN accuracy when 3,730,724 training patterns were used. The training error was minimized at 25 neurons. Additionally, the error dramatically increased for more than 70 hidden neurons. From this, 25 neurons was used for this study's ANN.

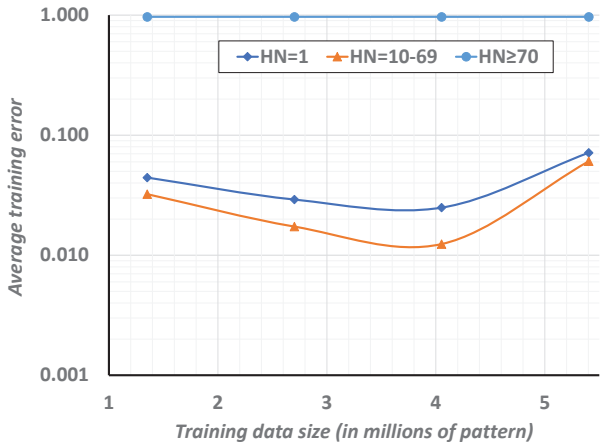

Figure 4.3: Adding more training data decreases ANN error, but only to a certain point. The point which minimized error in this study was 3,730,724 patterns.

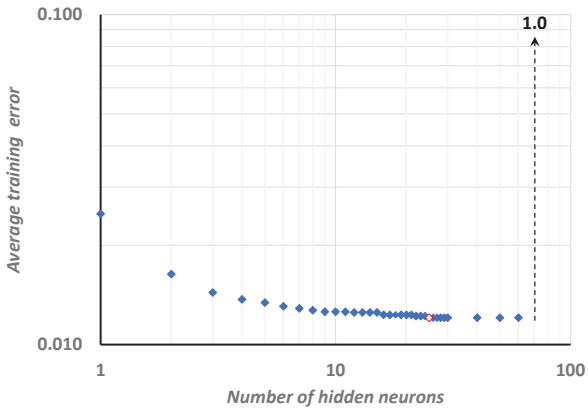

Figure 4.4: As more hidden neurons are added to the ANN, error drops (ANN accuracy improves), leveling off at 25 hidden neurons, only to increase again beyond 70 neurons.

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 were one-time investments, they were not considered relevant costs while assessing the computational effectiveness of ANN-based ATPG, whose time is significantly more than this training (data collection) time and is a one-time investment.

ATPG

Before evaluating a trained ANN against conventional backtracing heuristics, this section examines the effect of using different sets of input features, i.e., gate types, COP testability measures [108], and shortest distances to PIs (see Section 4.1.1). This experiment ran ATPG on the 100 hardest-to-detect faults in a subset of ISCAS'85 [110] and ITC'99 [111] benchmark circuits, namely, c6288, b04, c432, b08, b03, and b01: this restricted choice in faults and circuits was created by limited computational resources. The results of random-guided PODEM and of ANN-guided PODEM trained with various combinations of input features (and with no training) are given in Table 4.1.

Table 4.1: 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 4.2: 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**

4.2.1 Evaluating ANN Input Features

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

4.2.2 ATPG performance for hard-to-detect faults

This second experiment showed the performance 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 4.2 shows these results in terms of CPU time, the number of backtraces, and the number of backtracks. "MAR" (after the authors

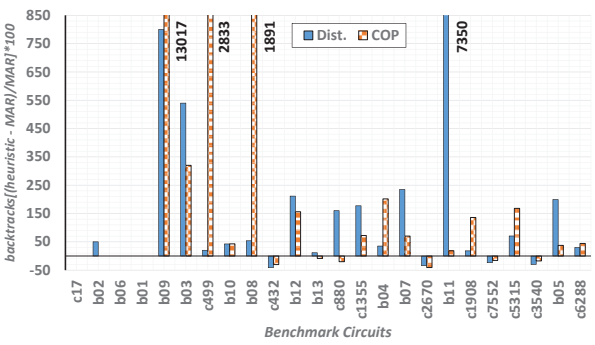

Figure 4.5: Percentage reduction in total backtracks by basic trained-ANN guidance (named MAR) with respect to the conventional (distance or COP) heuristic guidance. Backtracks are for all checkpoint single stuck-at faults tested by PODEM ATPG.

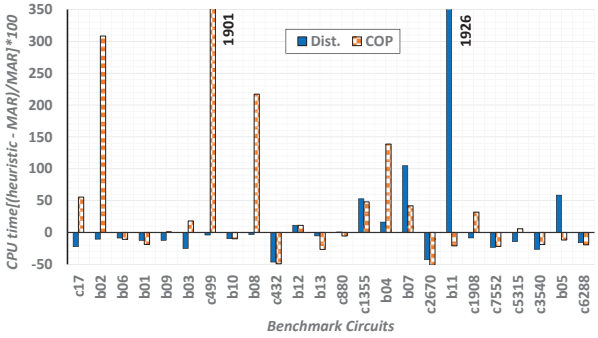

Figure 4.6: Percentage reduction in total CPU time by basic trained-ANN guidance (named MAR) with respect to the conventional (distance or COP) heuristic guidance. Backtracks are for all checkpoint single stuck-at faults tested by PODEM ATPG.

Millican-Agrawal-Roy) which is an ANN-guided heuristic, showed substantial improvements for several benchmarks, and when it did not perform the best, its detriments were marginal.

4.2.3 ATPG performance for all faults

This third experiment performed ATPG on all checkpoint stuck-at faults in 21 evaluation circuits and 3 training circuits from the ISCAS'85 [110] and ITC'99 [111] benchmarks. Because PODEM took exorbitant time to run exhaustively, some faults might be aborted. A suitable per-fault time limit was used to produce similar fault coverage with all heuristics.

Figures 4.5 and 4.6 plot the backtracks and CPU times of PODEM using the distance (Dist.) heuristic, the COP heuristic, and ANN-guided heuristic, respectively. Results are shown as percentage increase compared to the MAR heuristic ($[(heuristic - MAR)/MAR] * 100$), where $heuristic$ represents conventional heuristic and MAR represent ANN-guided 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 Fig. 4.6, although frequently positive, is not as positive as the impact on backtracks, which implies that CPU time of evaluating the ANN is significant.

In the subsequent discussion, the technique of this chapter will be referred to as "basic trained-ANN guidance".

Chapter 5

Training Neural Network for Machine Intelligence in Automatic Test Pattern Generator

Choosing the best path during backtrace is a vital problem in ATPG, and Chapter 4 showed that MI in the form of an ANN can replace any heuristic in PODEM [21] and speed up ATPG. However, some circuits did not exhibit any noticeable improvement in the backtracking performance. The "basic" ANN training of Chapter 4 was ad hoc and rudimentary, thus formulated training may elevate ANN-guided PODEM's performance. This chapter examines such a training method: the training recursively used ATPG data generated from hard-to-detect as well as easily tested faults, resolved conflicts in data patterns (e.g., when different ANN outputs come from similar inputs), and discarded data that did not improve the guidance of ANN. The specific contributions of this chapter are as follows:

- A technique to resolve conflicts among ANN training data. The main benefit of conflict resolution was improved ANN accuracy as evident from reductions in the MSE during training.

- A technique for selecting faults using a recursive training method that dynamically trained an "evolving" ANN as opposed to training with a pre-selected set of faults. ATPG data from a fault was retained only if the trained ANN guidance continued to further reduce backtracks compared to conventional heuristics (distance and COP). Both hard-to-detect and easy-to-test faults were sampled as opposed to only the former [2]. This prevented the training data from being overwhelmingly "failure oriented."

- An improved assessment of the quality of the ANN guidance by modifying the backtrack count procedure. I counted backtracks only for faults that were detected or

found redundant; earlier work [2] also included backtracks from aborted faults, which distorted the count.

The remainder of this chapter is organized as follows. Section 5.1 outlines the motivation leading to the present work, summarizes the objectives, and describes the proposed training techniques. Section 5.2 provides experimental exploration of the proposed training technique using benchmark circuits.

Major portions of this chapter are taken verbatim from my previously published research work [3].

5.1 Modus Operandi

Improvement in ATPG performance was reported [2] for many benchmark circuits, but several circuits, including some large benchmarks (c3540, c7552, c2670, and c432), did not perform well. I will discuss these results in the next section, since they serve as the motivation for the work in this chapter.

To motivate this work, I repeated previous work [2] with minor changes. Training patterns were generated using PODEM applied to 100 hard-to-detect faults in the three highest depth circuits: c6288, c3540, and b05. The trained ANN was used to guide a PODEM applied to the 100 hardest-to-detect faults in each of the previously used 24 benchmark circuits [2]. Figure 5.1 compares the total backtracks with those from conventional PODEM using distance [21] and COP [108] heuristics. One difference in my evaluation procedure is that I did not include backtracks from aborted faults. I referred to this ANN as the "basic trained-ANN" to distinguish it from the ANN developed in this chapter, identified as the "optimally-trained-ANN".

I made several observations from Fig. 5.1. Circuit size increased from left to right, and the first four small circuits and c2670 had no backtracks by any method. Also, for circuits b09, b10 and c2670, the ANN required no backtracks. Although ANN guidance satisfied the expectation of fewer backtracks than both conventional heuristics for several circuits, there

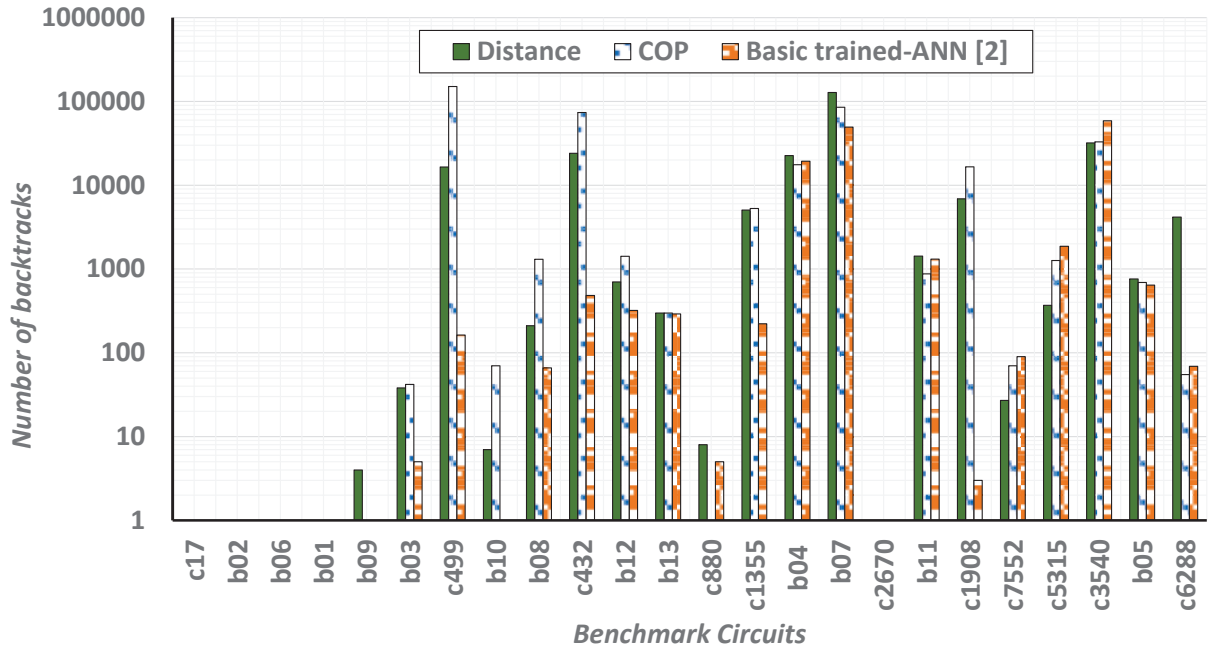

Figure 5.1: Backtracks in PODEM ATPG with various guidance mechanisms for 100 hard-to-detect checkpoint stuck-at faults (including detected and redundant).

were exceptions, especially among larger benchmarks. Notable among these were c7552, c5315 and c3540. For example, c3540 shows 32,008 (distance), 32,978 (COP), and 58,852 (ANN) backtracks used, which is similar to previous observations made in [2].

I believe remedying these outliers required addressing several aspects of the ANN training procedure. My investigation led to the following observations:

1. **Selection of training circuits.** In Chapter 4, the ANN was trained with ATPG data from high depth circuits: c6288, c3540 and b05. Since the Chapter 4's ANN-guided ATPG performance on c3540 did not show improvement, I retrained the ANN with just c3540, but still the ATPG performance of this circuit did not improve. Therefore, I retained the three high depth circuits for training.

2. **Resolution of conflicts among training data patterns**. These data patterns might lead to an increase in training error of the ANN and thus degraded its guiding ability. This is explained in Section 5.1.1.

3. **Selection of faults**. All faults do not provide useful training information, and pre-selection of faults for training may unnecessarily increase the training data volume without benefit. Section 5.1.2 gives a novel method of training the ANN considering three aspects, i.e., the training error of ANN must be kept low by adjusting the number of hidden neurons, training data from a fault must be accepted only if guidance from the resulting ANN reduces backtracks, and the faults used for training data generation should not be restricted to hard-to-detect faults. In fact, any fault provides useful training data as long as training with it reduces backtracks.

4. **Forgetfulness of the ANN**. The ANN may enter a zone called "catastrophic forgetting" [112] when it forgets information contained in a large training data volume. Such consideration in training improvement is worth exploring in the future.

In this chapter, I discuss new training strategies addressing the above observations (except item 4) to improve the backtracking performance of ANN-guided PODEM. In Chapter 4, the number of hidden neurons in the ANN was preselected, but a static ANN was incapable of absorbing increasing amounts of training data. The new training technique progressively added training data after resolving conflicts in data patterns to further reduce backtracks while discarding data that did not accomplish this objective, and the ANN evolved through adding hidden neurons to further reduce MSE during training.

5.1.1 Resolving conflicts in training data

Each training data pattern "i" consisted of an input vector $\{x_i\}$ of features and an output label z_i. The label and all features range in [0,1]. Two patterns, i and j, formed a conflicting pair if $\{x_i\} \equiv \{x_j\}$ and $z_i \neq z_i$. This was because the ANN could not be trained to produce different outputs for the same input. The presence of conflicting patterns in training data influenced the training as indicated by non-decreasing MSE during training.

To remedy this, I collected patterns with identical features into groups and then replaced each group by a single *representative pattern* with common features. Since these patterns

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
0	0	0	0	0	0	0	0	1	0.76	0.84	0.12	1
0	0	0	0	0	0	0	0	1	0.76	0.84	0.12	1
0	0	0	0	0	0	0	0	1	0.76	0.84	0.12	0
0	0	0	0	0	0	0	1	0	0.95	0.95	0.95	1
0	0	0	0	0	0	0	1	0	0.95	0.95	0.95	1
0	0	0	0	0	0	1	0	0	0.5	0.5	0.5	0
0	0	0	0	0	0	1	0	0	0.5	0.5	0.5	0
0	0	0	0	1	0	0	0	0	0.37	0.63	0.25	1
0	0	0	0	0	1	0	0	0	0.36	0.43	0.75	0
0	0	0	0	0	0	0	0	1	0.63	0.34	0.8	1

Figure 5.2: An example of ANN training data patterns with conflicts. Note the first three patterns with identical inputs (features) and conflicting outputs (labels).

were derived from actual ATPG runs, the label of a pattern was either 1 (indicating success) if it belonged in a backtrace leading to a test or 0 (indicating failure) if it resulted in a backtrack. I counted the number of 1 and 0 labels in the group, and the representative pattern was given the label $\frac{count(1)}{count(0)+count(1)}$.

A conflict is illustrated by the sample training patterns in Fig. 5.2, where the first three patterns have a conflict. This is resolved in Fig. 5.3, where the three patterns were replaced by a single representative pattern with a label $\frac{2}{2+1} = 0.67$.

I collected training data from conventional heuristic-based PODEM applied to the training circuits (c6288, c3540 and b05), resolved all conflicts among training data patterns, trained the ANN, and integrated the ANN guidance into PODEM. I observed that pre- and post-conflict resolution MSE were 3% and 1%, respectively. Additionally, training patterns were compacted in this manner allowed more faults to be included during training.

5.1.2 Recursive training and evolving ANN

In the Chapter 4, PODEM with random heuristic generated the ANN training data. This could be time-consuming, difficult, and non-repeatable. Experimental results of Fig. 5.1

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
0	0	0	0	0	0	0	0	1	0.76	0.84	0.12	0.67
0	0	0	0	0	0	0	1	0	0.95	0.95	0.95	1
0	0	0	0	0	0	0	1	0	0.95	0.95	0.95	1
0	0	0	0	0	0	1	0	0	0.5	0.5	0.5	0
0	0	0	0	0	0	1	0	0	0.5	0.5	0.5	0
0	0	0	0	1	0	0	0	0	0.37	0.63	0.25	1
0	0	0	0	0	1	0	0	0	0.36	0.43	0.75	0
0	0	0	0	0	0	0	0	1	0.63	0.34	0.8	1

Figure 5.3: Example ANN training data patterns after resolving conflicts. The first pattern here replaces the first three patterns of Fig. 5.4.

show that COP [108] heuristic can provide reasonable ATPG performance (i.e., fewer backtracks or in the same ballpark regime) for the training circuits. Therefore, PODEM guided by COP was used for training data generation in this chapter.

Since ANN training quality largely depends on its structure and complexity, this chapter examined the interrelation between the number of hidden neurons and MSE of the ANN to find a "sweet-spot" that guaranteed efficient ANN training. This "sweet-spot" was found during a new ANN training step by either adding hidden neurons or restoring the previous optimal hidden neurons with corresponding training data, making the ANN effective with minimal MSE. Training data from PODEM using COP was obtained from a small set of faults and applied recursively to train the ANN. This training continued to minimize MSE by adding small batches of faults to the training as long as they continued to improve the ANN quality. This made my ANN dynamic in terms of hidden neurons.

The algorithm to develop an "evolving" ANN-guided ATPG is illustrated in Fig. 5.4. I selected large depth circuits, c6288, b05, and c3540, from the ISCAS'85 and ITC'99 benchmarks [110,111]. The ANN structure was same as in the Chapter 3 (Fig. 3.1) and parameters were initialized as "hidden neurons (HN) = 10" and "MSE = 1.0". COP-based PODEM was applied to a set of 100 hard-to-detect faults, and the number of backtracks was recorded

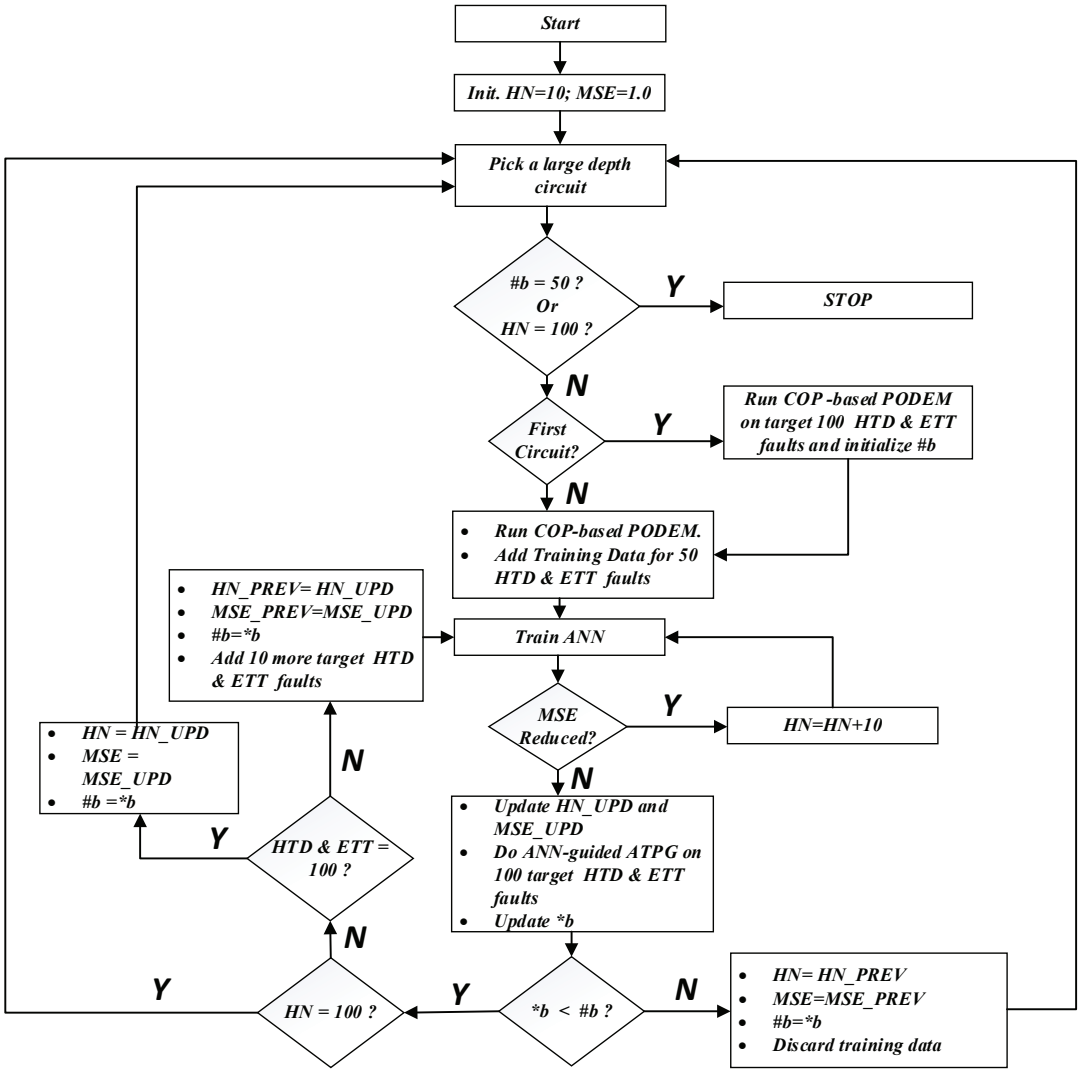

Figure 5.4: Flow chart of proposed training methodology, including sub-procedures, recursive training by conventional heuristic-based PODEM, followed by "evolving" ANN.

"#b" to be minimized through recursive training. The training started by using 50 hard-to-detect faults to generate a training database, followed by ANN training to check MSE and record the corresponding number of hidden neurons, simultaneously. This process continued until MSE started saturating, and the corresponding MSE "MSE_UPD" and the number of hidden neurons "HN_UPD" were recorded and the ANN was re-trained. A similar process was continued for 100 easy-to-test faults. To evaluate the training, the ANN-guided ATPG was applied to 100 hard-to-detect faults of the same training circuits. If the backtrack count was decreased, then 10 more hard-to-detect faults were added to the training of the ANN, else

the hidden neurons "HN_PREV" and MSE "MSE_PREV" were restored, the corresponding training patterns from training database were discarded, and another circuit was selected to re-iterate the same process until one of the following conditions was satisfied: 1) the number of backtracks was reduced to 50; 2) 100 hard-to-detect faults were used for training; or 3) the ANN contained 100 hidden neurons.

5.2 Experimental Results

In this chapter, a workstation containing an Intel i7-8700 processor and 8 GB of RAM performed all experiments. Tools were implemented in C++ using the MSVC++ 14.15 compiler with maximum performance optimization, and all ANN activity was executed in Python. PODEM ATPG [21] was reproduced along with event-driven fault simulation [4] such that any heuristic (distance [21], or COP [108]) can be applied across ISCAS'85 [110] and ITC'99 [111] benchmark circuits without favoring a single heuristic. This chapter will surely polarize EDA vendor mindset to deploy MI in their ATPG software. However, EDA vendors hesitate to divulge their program source code, and it is impossible to conduct research-based experiments using the executable. Therefore, I prefer to run the experiments using the in-house EDA tools.

Figure 5.5 shows how this chapter's optimally-trained-ANN performed on 100 hard-to-detect faults across benchmarks as compared to the basic trained-ANN [2]. In terms of backtracks, with few exceptions, the optimally-trained-ANN did the same or better in reducing backtracks compared to the basic trained-ANN [2], especially for larger circuits.

My next experiment used all (testable and redundant) faults to show the efficacy of guidance by the new ANN using the proposed training technique. Table 5.1 shows the computation time of ATPG "CPU Time (ms)", "Backtrace count", and "Backtrack count". Clearly, the new ANN performs better (with fewer backtracks and less ATPG computation time (see Fig. 5.7) than the basic trained-ANN [2], reaffirming the value of the proposed training technique of this chapter. The backtrack counts for all faults are shown in Fig. 5.6.

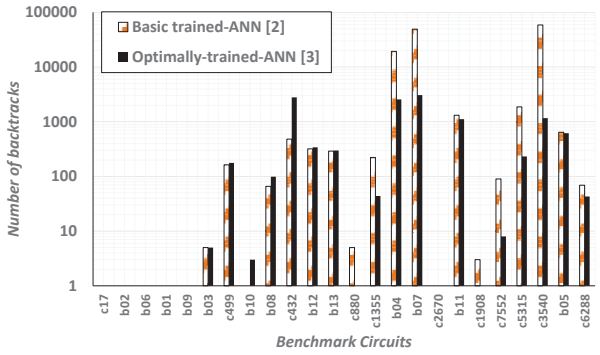

Figure 5.5: Backtracks in PODEM ATPG with various guidance mechanisms for 100 hard-to-detect checkpoint stuck-at faults (including detected and redundant).

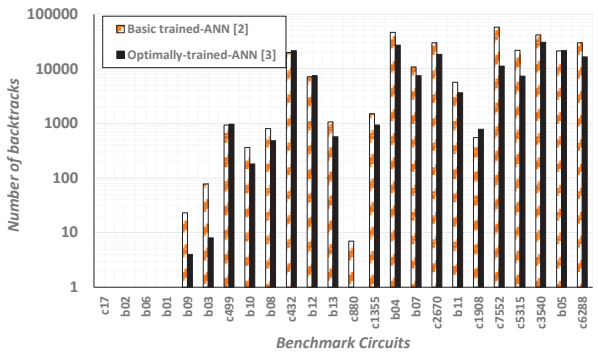

Figure 5.6: Backtracks in PODEM ATPG with various guidance mechanisms for all checkpoint stuck-at faults (including detected and redundant).

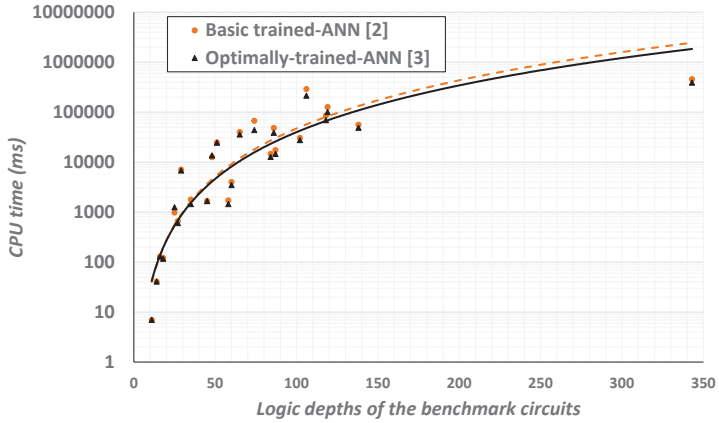

Figure 5.7: CPU times (ms) of PODEM ATPG guided by basic trained-ANN [2] and optimally-trained-ANN [3] for all checkpoint single stuck-at faults (including detected and redundant).

Table 5.1 has some notable observations. First, c17, b02, and b01 had no reconvergent fanouts and therefore had no backtracks. Of course, there was no scope for reducing backtracks by new ANN guidance. I observed that the number of backtraces was either constant or reduced in these reconvergent fanout-free circuits by the optimally-trained-ANN over the basic trained-ANN [2]. Second, except for c1908, c432, c499, b12, and b05, the new ANN exceeded expectations in terms of performance based on reductions in backtracks. Third, the new ANN was able to reduce backtracks and backtraces for b09, but the CPU time increased. Possibly more time per backtrace was used to reduce backtracks by expensive

Table 5.1: Performance of PODEM ATPG for all checkpoint faults in benchmark circuits, guided by basic trained-ANN (chapter 4) [2] and optimally-trained-ANN (this chapter). Boldface numbers show reduced backtracks by the latter.

Circuit name	**Basic trained-ANN [2]**			**Optimally-trained-ANN [3]**		
	CPU Time (ms)	**Backtrace count**	**Backtrack count**	**CPU Time (ms)**	**Backtrace count**	**Backtrack count**
c17	7	64	0	7	64	**0**
b02	41	236	0	41	236	**0**
b06	128	514	1	130	506	**0**
b01	121	514	0	117	498	**0**
b09	984	3293	23	1250	3239	**4**
b03	662	2166	78	605	1901	**8**
c499	7097	22418	933	6828	22888	965
b10	1784	3718	361	1448	3225	**181**
b08	1683	4420	804	1665	4205	**481**
c432	12610	26253	19840	13626	30150	21441
b12	24877	33814	7161	24496	33945	7482
b13	1720	5481	1063	1453	4871	**570**
c880	4014	11889	7	3481	10459	**0**
c1355	40231	58658	1498	35825	57788	**934**
b04	67329	66182	46423	43855	47139	**27110**
b07	14660	18851	10810	12741	16420	**7476**
c2670	48296	72347	29924	38816	65159	**18355**
b11	17397	21008	5673	14601	18242	**3641**
c1908	30616	39150	549	27685	35982	779
c7552	291393	297572	57874	214372	222395	**11183**
c5315	85702	105072	21782	69560	86429	**7321**
c3540	126861	82609	41842	100468	71492	**30529**
b05	56152	43078	21260	48830	39892	21540
c6288	457206	212968	29982	390805	180588	**16525**

evaluation of weights and biases of ANN edges that involved matrix multiplication and computation of the sigmoid [45] function. Forth, the optimally-trained-ANN heuristic did not perform as well on a few circuits compared to the basic trained-ANN [2], but it outperformed the conventional heuristics (like Distance [21] and COP [108]), except in case of c432. Fifth, Fig. 5.7 illustrates no CPU time reduction for smaller circuits, and a reduction for high depth circuits was observed, perhaps due to simultaneous reduction in backtracks and backtraces.

In the subsequent discussion, the technique of this chapter will be referred to as "optimally-trained-ANN guidance".

Chapter 6

Unsupervised Learning in Test Generation for Digital Integrated Circuits

The exponential complexity of ATPG necessitates the use of heuristics in making choices during test generation. However, in practice no single heuristic fits all situations. Unsupervised learning can combine any number of known ATPG inputs, such as input-output distance (logic depths), gate type, fanout information, and testability measures like COP and SCOAP as heuristics through PCA, and then the major PCs can guide ATPG choices. This chapter combines three ATPG inputs–distance, COP, and SCOAP–into a single heuristic. Some ATPG input data are complemented and two major PCs are obtained. These PCs guide backtrace directions in a PODEM ATPG program. For most circuits, the number of backtracks either matches the best of the three heuristics or is lower than all.

Major portions of this chapter are taken verbatim from my previously published research work [14].

6.1 Modus Operandi

MI has two phases: learning from problem-specific data and then using that knowledge to solve problems. In supervised learning, these phases may be ANN training and ANN guidance. In contrast, unsupervised learning uses statistical tools such as PCA [40, 41] and k-means clustering [47, 113–115]. In the first phase, the tool analyzes the problem-specific data to extract relevant characteristics, which in the second phase directly helps solving problems. I applied unsupervised learning to the ATPG problem using the PCA as discussed in Section 3.2.

6.1.1 Dimensionality Reduction

This chapter uses SVD to obtain PCs based on a parameter known as *explained variance* [48]. PCs represent the same amount of information as the original data, i.e., the original data can be restored from the PCs. Moreover, the total variances of the original and transformed data are the same but are redistributed unequally among the PCs. The first PC (also known as the major PC) has the highest variance. The standard quality measure *explained variance* π_j of the jth PC is the ratio of its variance λ_j to the total variance (sum of variances of all PCs):

$$\pi_j = \frac{\lambda_j}{\sum_{i=1}^{p} \lambda_i} \tag{6.1}$$

where, p is the number of PCs and λ_i is the individual variance of ith PC. The progressive nature of PCs means that a proportion of total explained variance for a subset of S PCs is expressed as a percentage of the total variance: $\sum_{i \in S} \pi_i$. It is a common practice to set a threshold for this total variance to decide how many PCs to use; only first one, two, or three PCs may be required. However, there are circumstances, such as outlier detection [47] or image analysis, where the last few PCs may be of interest.

6.1.2 Multi-Heuristic Guidance for ATPG

This chapter combines a variety of ATPG inputs, such as, the shortest distance d to primary inputs [21], COP controllabilities [108], and SCOAP testability measures [109]. From all the signal probabilities of COP–$CC0$ and $CC1$–I only used $CC1$ because of their complete dependence on each other: $CC0 = 1$ - $CC1$. SCOAP 0 and 1 combinational controllabilities were denoted here as $SC0$ and $SC1$. For every node in the circuit four quantities, d, $CC1$, $SC1$, and $SC0$, were computed using known linear-time algorithms [21, 108, 109]. Each quantity was normalized to [0,1] range with respect to its maximum value over all nodes.

Table 6.1: Heuristic-based input selection criteria for backtracing through a gate to justify output value.

Gate	**d**		**CC1**		**SC1**		**SC0**		**P**	
***Value*→**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
AND	min	max	min	min	max	max	min	min	**?**	**?**
NAND	max	min	min	min	max	max	min	min	**?**	**?**
OR	max	min	max	max	min	min	max	max	**?**	**?**
NOR	min	max	max	max	min	min	max	max	**?**	**?**

The same heuristic data were used in supervised learning-based PODEM ATPG [2, 3]. There, backtrace choices were directed by a trained ANN that computed relative metrics for the available nodes. The highest metric implied best chance of finding a test without a backtrack. For training the ANN, the information about how each heuristic influences success was derived from ATPG runs on sample circuits. In the unsupervised learning system, however, no ANN was used. Instead, multiple circuit data were combined through PCA for directly guiding the backtrace.

6.1.3 Principal Component Analysis (PCA)

In combining multiple ATPG inputs, it was necessary that they worked cooperatively without contradicting each other in comparing the effectiveness of inputs of a logic gate while justifying the output value. Table 6.1 shows how individual heuristics worked. For example, consider a backtrace through an AND gate with two or more inputs being guided by d. To justify the output value 0, the backtrace must take the input closest to PIs [21]. In Table 6.1, this is indicated by "min" under d for AND gate and value = 0. To justify a 1 at the output, the backtrace followed the input with highest d, shown as "max". I observed that the four heuristics do not agree for any of the gates. Hence, if combined by PCA, the major component (P) cannot be given guidance criteria.

Table 6.2 takes a two-step approach to overcome the above difficulty. First, some circuit data were complemented. For example, when an AND gate output was 1, its inputs–$CC1$ was replaced with 1 - $CC1$ and $SC0$ with 1 - $SC0$. Also, when the AND gate output was

Table 6.2: Principal components ($P0$ and $P1$) for gate output = 0 and 1. *Italicized* decision criterion (*min* or *max*) shows complemented heuristic data to achieve synchronization.

Gate	**d**		**CC1**		**SC1**		**SC0**		**P0**	**P1**
***Value*→**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
AND	min	max	min	*max*	*min*	max	min	*max*	**min**	**max**
NAND	max	min	*max*	min	max	*min*	*max*	min	**max**	**min**
OR	max	min	max	*min*	*max*	min	*max*	*min*	**max**	**min**
NOR	min	max	*min*	max	min	*max*	*min*	max	**min**	**max**

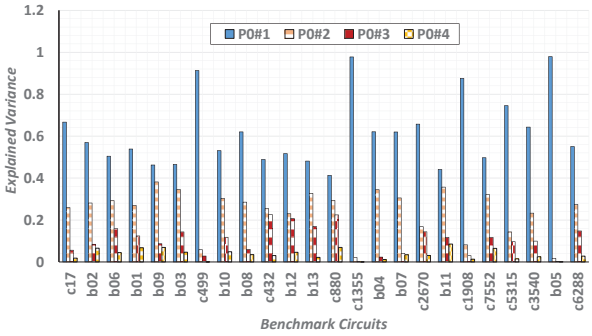

Figure 6.1: PCA for ISCAS'85 and ITC'99 benchmarks. Heuristic data are complemented according to Table 6.2 assuming 0 output for all gates. The major PC, P0#1, is shown in blue.

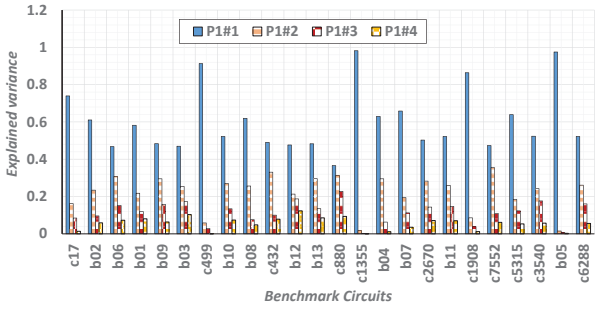

Figure 6.2: PCA for ISCAS'85 and ITC'99 benchmarks. Heuristic data are complemented according to Table 6.2 assuming 1 output for all gates. The major PC, P1#1, is shown in blue.

0, $SC1$ was replaced by 1 - $SC1$. This reversed the corresponding backtrace criteria now shown in italics. Similar changes were made for NAND, OR, and NOR gates, giving complete synchronization of the choice criteria for all conventional heuristics and a way for PCA to interpret the circuit data. However, it necessitated separate PCs for gate outputs 0 and 1, respectively, requiring two major PCs, $P0$ and $P1$, with corresponding backtrace criteria (see Table 6.2).

6.1.4 Preprocessing and ATPG

To run the ATPG, circuit netlist was preprocessed to compute four values for each signal node, namely, D [21], CC1 [108], and SCOAP combinational measures SC0 and SC1 [109]. Complemented values were computed according to Table 6.2 and P0 and P1 were from PC

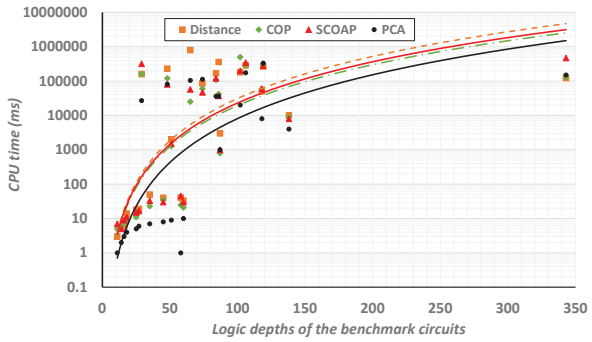

Figure 6.3: CPU time for detecting all faults with ATPG using conventional heuristic and PCA-guidance.

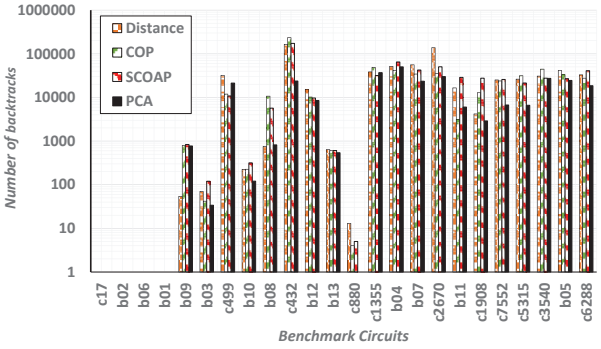

Figure 6.4: Total backtracks for detecting all faults with ATPG using conventional heuristic and PCA-guidance. Four circuits on the left required no backtracks and c880 required no backtracks only when PCA was used.

analyses for all gate outputs assumed as 0 and 1, respectively. PCA results for ISCAS'85 [110] and ITC'99 [111] benchmarks were shown in Figures 6.1 and 6.2. The blue bars show the major PCs, $P0$ and $P1$.

6.2 Experimental Results

In this chapter, a workstation containing an Intel i7-8700 processor and 8 GB of RAM performed all experiments. Tools were implemented in C++ using the MSVC++ 14.15 compiler with maximum performance optimization, and all PCA activity was executed in Python. PODEM ATPG [21] was reproduced along with event-driven fault simulation [4] such that any heuristic (distance [21], COP [108], SCOAP [109], or PC) can be applied across ISCAS'85 [110] and ITC'99 [111] benchmark circuits without favoring a single heuristic. This chapter will surely polarize EDA vendor mindset to deploy MI in their ATPG software. However, EDA vendors hesitate to divulge their program source code, and it is impossible to conduct research-based experiments using the executable. Therefore, I prefer to run the experiments using the in-house EDA tools.

Experiments used testable and redundant faults to prove the efficacy of guidance provided by PCA to PODEM ATPG [21]. Figures 6.3 (circuits are arranged according to

increasing logic depths) and 6.4 show relevant findings on ATPG CPU time (ms) and the number of backtracks with respect to distance [21], COP [108], SCOAP [109], and PCA ($P0$ and $P1$ guidance).

This experiment demonstrated how combining multiple ATPG inputs as heuristics into a linear combination through PCA can achieve better ATPG CPU time and fewer backtracks compared to conventional single heuristics. Circuits c1355, c2670, c3540, b04, b11, b08, c499 and c6288 showed improvement of reduced backtracks, but these circuits needed more backtraces, thus CPU time increased. Most frequently, PCA was the best guidance for ATPG, but when it was not it was never the worst performing. Circuits c17, b02, b01, and b06 had no reconvergent fanouts, and therefore had no scope for reducing backtracks. Circuit c880 was the good example of zero backtracks in PCA-based PODEM ATPG compared to other conventional heuristics, which was significant in terms of the ability to achieve no backtracks.

The technique of this chapter is referred to as "PCA-guidance". I observe that PCA can provide guidance to ATPG without using ANN. Seemingly, an advantage might be that the PCA-guidance is now customized to the circuit under test (CUT). In contrast, the ANN must be pre-trained using data from training circuits, which are different from the CUT. However, the trained ANN contains more information than the heuristics; it is also trained with sample ATPG data. In the next chapter, I incorporate PCA in the training of ANN, which will then guide the ATPG.

Chapter 7

Principal Component Analysis in Machine Intelligence-Based Test Generation

Human clairvoyance in the form of heuristics was successfully used in ATPG programs, but it has been reported that no single heuristic works optimally for all instances, and the use of multiple heuristics can be computationally expensive [106, 107]. Although MI is not a new technique, considerable performance impact of ANN was found in test point insertion (TPI) [13, 76, 116–118]. Recent advances in MI-based ATPG demonstrated that heuristics can be easily incorporated in ATPG through MI [2,3]. Thus, MI could harness the benefits of multiple heuristics. However, as the volume of heuristic data increases, the workhorse of MI, i.e., the ANN, tends to be overloaded to the extent that its efficiency suffers. PCA [40, 41] can amalgamate training features to enhance supervised learning of ANN. Although the application of PCA-trained-ANN is not new, this chapter demonstrates its novel application to ATPG.

This chapter improves ANN training quality and efficiency by pre-processing training data with PCA [40, 41] that extracts relevant features from a list of many features to train an ANN [119–121] and offers a viable pre-processing step [122] to decrease ANN complexity. The heuristics in ATPG were built around topological data of the circuit, and this chapter used correlation among data to achieve compaction [40, 41] and extracted the PCs of the circuit data. Thus the ANN complexity was reduced as it was now trained only with a few selected PCs. Additionally, the ATPG CPU time was reduced since the trained ANN was now less complex, and evaluating weights and biases of ANN edges required smaller matrix multiplication and fewer computations of non-linear sigmoid functions [45].

This chapter is organized as follows. Section 7.1 outlines the contribution of this study that explains the PC extraction with detailed mathematical backgrounds and technique to

choose major PCs. Section 7.2 evaluates the performance of PODEM guided by the PC-trained ANN against that of PODEM guided by ANN without PCs [2, 3].

Major portions of this chapter are taken verbatim from my previously published research work [15].

7.1 Modus Operandi

The performance of ANN-guided PODEM [2] was further improved by optimizing the training methodology [3]. However, the addition of more features to the ANN would cause problems of high volume of the training dataset and the ANN complexity to absorb and retain the information. This leads to the storage crunch of such a high volume of training data and leads to high ANN training time. Also, it may be possible that some training features are irrelevant, and therefore extraction of useful training features is one such novel contribution of this chapter, which enhances ATPG performance (both in terms of backtracks and CPU time).

An increase in the training data set is prevalent and alarming. A multivariate statistical method, popularly known as PCA [40, 41], reduces the data sets' dimensionality and increases interpretability with minimum information loss. PCA creates new uncorrelated variables (also known as PCs) with maximum variances. Finding PCs reduces to solving an eigenvalue/eigenvector problem; the new variables are not defined a priori, but by the data set at hand, making PCA a pliant data analysis technique.

PCA is a technique to identify patterns in data and express the data to show the similarities and dissimilarities. Any patterns in high-dimensional data are hard to find, but PCA plays a vital role in analyzing these data where the luxury of graphical representation is not available. PCA is also useful in compressing data by reducing the number of dimensions without losing necessary information once patterns in the data are found [40, 41]. Before this work, PCA found significant application in image processing and recently in the form of unsupervised learning in ATPG [14], but this statistical tool has not been explored as

Table 7.1: Example of 8-dimensional feature (signal characteristic) data for first 5 signals of training circuit c6288.

Fan-out	***Gate type***	***COP CC***	***COP CO***	***SCOAP SC0***	***SCOAP SC1***	***SCOAP SCO***	***Dist.***
0.000	0.000	0.063	1.000	0.013	0.058	0.000	0.237
0.000	0.000	0.063	1.000	0.013	0.058	0.004	0.211
0.000	0.000	0.938	1.000	0.034	0.016	0.004	0.184
0.000	0.000	0.938	0.063	0.034	0.016	0.045	0.158
1.000	0.330	0.500	0.125	0.007	0.011	0.034	0.132

a pre-processing step of ANN training in ATPG or MI-based ATPG (also known as *PCA-trained-ANN guided ATPG*).

This chapter demonstrates PCA-based pre-processing of training data obtained from circuits, c6288, c3540, and b05, chosen due to their large logic depths. The use of deep circuits in training ANN for ATPG was empirically found to be effective [3]. The ANN training data was obtained from successful and failed backtraces in a COP-based ATPG, as illustrated in Chapter 4. In this chapter, the ANN recognizes eight features (characteristics) for each signal line (PI, gate output, or fanout branch). The feature values were normalized in the range [0, 1]. They are specified below, with examples shown in Table 7.1:

1. **Fanout** - Its value is 0 for a signal (line) with single destination, and 1 for multiple destinations.

2. **Gate type** - The type of a signal is specified numerically. PI, fanout branch, and inverter output are type 0.0. A multiple-input gate output signal, which can be a non-fanout signal or a fanout stem, is type 1 through 6 (without any significance to a gate being any type) corresponding to AND, NAND, OR, NOR, XOR, or XNOR gate, respectively. After normalization the gate-type becomes 0.0, 0.167, 0.33, 0.5, 0.67, 0.83, or 1.0.

3. **COP CC** - Combinational controllability computed by COP [108] as probability of setting the signal to 1.

4. **COP CO** - Combinational observability computed by COP [108] as probability of observing the signal at POs.

5. **SCOAP SC0** - Effort of setting the signal to 0 as computed by SCOAP [109], normalized with respect to the maximum $SC0$ of the circuit.

6. **SCOAP SC1** - Effort of setting the signal to 1 as computed by SCOAP [109], normalized with respect to the maximum $SC1$ of the circuit.

7. **SCOAP SCO** - Effort of observing the signal at POs as computed by SCOAP [109], normalized with respect to the maximum SCO of the circuit.

8. **Distance** - Number of lines on the shortest path between the signal and PIs, normalized with respect to the maximum PI to PO depth of the circuit.

7.1.1 Mathematical Background of PCA

This section explains how PCA is performed on a set of data and also attempted to provide elementary mathematical background required to understand PCA's mechanisms, such as calculating mean, covariance matrix, eigenvectors, and eigenvalues of a covariance matrix, choosing the components that formed a feature vector, and finally deriving a new data set based on feature vectors.

Step 1: Getting data

Examples of 8 input features of the ANN are given in Table 7.1. For simplicity, 2-dimensional data for "COP CO" and "Dist." are illustrated in Table 7.2. Figures 7.1 and 7.2 show the combined training data from circuits c6288, c3540, and b05 for two pairs of features. These are scatter plots of raw data and show how some features can have stronger correlation. The circular or elliptical concentration with minimum outliers indicates either uncorrelated or correlated data, respectively. Although not empirically proven here, this issue will be revisited in later sections.

Table 7.2: Example of features x = COP CO and y = distance in Table 7.1, and mean-adjusted values for first 5 signals of c6288. Means $< x >$ and $< y >$ are computed for all signals of training circuits c6288, c3540 and b05.

COP (CO) x	*Distance* y	$x_{adjust} = x - < x >$	$y_{adjust} = y - < y >$
1	0.237	0.795	-0.021
1	0.211	0.795	-0.048
1	0.184	0.795	-0.074
0.063	0.158	-0.143	0.100
0.125	0.132	-0.080	-0.127

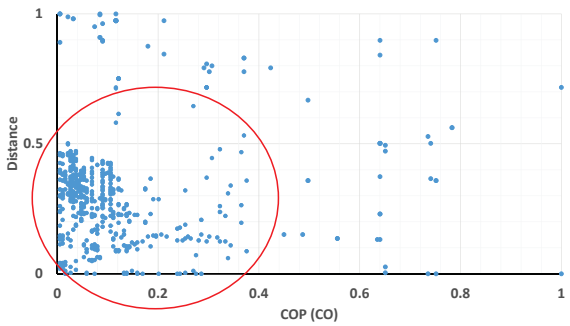

Figure 7.1: A two-dimensional scatter plot of "distance" and "COP CO" data for all signals in training circuits c6288, c3540, and b05. A nearly circular concentration indicates a weak correlation between two features.

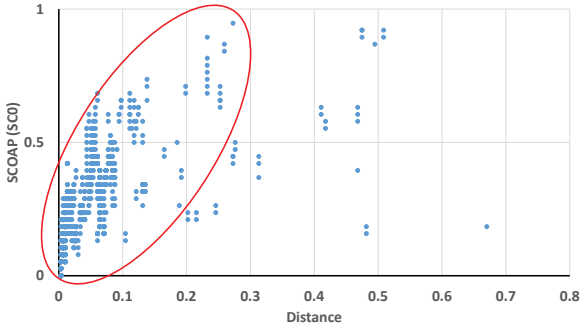

Figure 7.2: A two-dimensional scatter plot of "SCOAP SC0" and "distance" data for all signals in training circuits c6288, c3540, and b05. The elliptical concentration indicates significant correlation between two features.

Step 2: Subtracting mean

The mean of each data type in Table 7.1 was calculated and subtracted from the respective data as shown in Table 7.2. The well known procedure for computation of mean is [123]:

$$< a >= \frac{1}{N}\sum_{i=1}^{N} a_i = \frac{a_1 + a_2 + \cdots + a_N}{N} \tag{7.1}$$

where a is data sample, and N is the total number of samples.

Step 3: Calculating the covariance matrix

Data sets can be either single or multi-dimensional, and depending on which statistical tools are used. Standard deviation and variance are such statistical tools that can be used for one-dimensional data sets to calculate the standard deviation for each dimension of the data, independent of the other dimensions. However, it was essential to have an evaluation metric to find how much the dimensions vary from the mean concerning each other, known as *covariance*. The mathematical formula for covariance is similar to variance [123]:

$$Var_x = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - < x >)^2 \tag{7.2}$$

$$Cov_{x,y} = \frac{\sum_{i=1}^{N}(x_i - < x >)(y_i - < y >)}{N-1} \tag{7.3}$$

The total number of covariances for 2 or more data dimensions is $\frac{n^2-n}{2}$, where n is the dimension of the data set. As this chapter deals with 8-dimensional data sets, $n = 8$, and the total number of covariances (i.e., the number of elements in either the lower triangle or the upper triangle of a symmetric matrix) is 28.

For a simple illustration, let us consider 2-dimensional data of Fig. 7.1 that produced only one covariance. The higher the covariance, the stronger is correlation between features. In this case, covariance was extremely low, suggesting that these features were almost uncorrelated. The covariance matrix was computed using equation 7.2 as:

$$C = \begin{pmatrix} 0.057 & 0.008 \\ 0.008 & 0.026 \end{pmatrix} \tag{7.4}$$

Step 4: Calculating eigenvectors and eigenvalues

Eigenvectors and eigenvalues [123] are calculated from the covariance matrix C. These signify the relative strengths of data components, which help identify significant PCs. Eigenvectors are orthogonal to each other and provide useful reorganization of data. Eigenvectors represent a rotation matrix, and the eigenvalues corresponded to the square of the scaling factor in each dimension.

$$eigenvectors = \begin{pmatrix} 0.973 & -0.232 \\ 0.232 & 0.973 \end{pmatrix} \tag{7.5}$$

$$eigenvalues = \begin{pmatrix} 0.059 & 0.024 \end{pmatrix} \tag{7.6}$$

Step 5: Forming a feature vector

This step illustrates compressing and reducing dimensionality of a data set. All eigenvectors are different and have different eigenvalues. The eigenvector with highest eigenvalue is the major PC of the data set. It carries maximum significance among data dimensions. Eigenvectors are obtained from the covariance matrix and ordered according to decreasing eigenvalues. One may choose the significant eigenvectors based on their high eigenvalues and discard the rest of the eigenvectors without losing much information, as shown below. In the present situation, the two variables had rather low correlation and hence none were dropped. However, just for illustration, one can drop the second variable:

$$selected\ eigenvector = \begin{pmatrix} 0.973 \\ 0.232 \end{pmatrix} \tag{7.7}$$

Finally, n dimensional data may produce at most n eigenvectors and corresponding eigenvalues. A subset of p eigenvectors may be chosen by eliminating those with relatively small eigenvalues. Finally, a data set of dimension p ($p \leq n$) is created in the next step.

Step 6: Reconstructing a new data set

A transform T is an $n \times n$ square matrix containing eigenvectors as rows. The mean-adjusted feature data for each line is an n-dimensional vector. This vector, when multiplied by T–produces a new n-dimensional vector of PCs.

7.1.2 Selecting Major Principal Components

This section highlights details on generation of six major PCs for each line, as discussed in Section 7.1.1. There were various avenues to fix the number of significant PCs, but the Pearson correlation coefficient (PCC)-based technique was chosen to compress/extract the final dimensions of PC-based data [124]. However, datasets contained a mix of correlated and uncorrelated items. The neural network training became more efficient when all its input features are strictly orthogonal or, in other words, un-correlated. Therefore, PCC is a handy technique by which one can compress unnecessary correlated data, keeping the uncorrelated data intact. The well-known equation for PCC [123] is as follows:

$$r = \frac{\sum_{i=1}^{N}(x_i - < x >)(y_i - < y >)}{\sqrt{\sum_{i=1}^{N}(x_i - < x >)^2}\sqrt{\sum_{i=1}^{N}(y_i - < y >)^2}} \tag{7.8}$$

where r is PCC, x and y are data samples of a two-dimensional dataset, $< x >$ and $< y >$ are computed mean of data samples, and N is number of samples.

The entire data set from the three training circuits, of which only a sample is shown in Table 7.1, was analyzed to compute correlation coefficients as shown by the 8×8 matrix below. I observed that diagonal elements are self correlated. Because they are highly correlated, the PCC is 1 (highlighted in bold). Pair-wise correlation coefficients are off-diagonal elements and, as pointed out earlier, considering the diagonal symmetry there are 28 of them.

$$\begin{pmatrix}
\mathbf{1.000} & 0.246 & -0.099 & -0.086 & -0.145 & -0.005 & 0.031 & -0.029 \\
0.246 & \mathbf{1.000} & -0.107 & -0.100 & -0.142 & 0.025 & -0.049 & -0.091 \\
-0.099 & -0.107 & \mathbf{1.000} & 0.064 & 0.322 & -0.164 & 0.039 & 0.212 \\
-0.086 & -0.100 & 0.064 & \mathbf{1.000} & 0.130 & 0.214 & -0.364 & 0.201 \\
-0.145 & -0.142 & 0.322 & 0.130 & \mathbf{1.000} & \mathbf{0.408} & 0.286 & \mathbf{0.622} \\
-0.005 & 0.025 & -0.164 & 0.214 & \mathbf{0.408} & \mathbf{1.000} & 0.175 & \mathbf{0.559} \\
0.031 & -0.049 & 0.039 & -0.364 & 0.286 & 0.175 & \mathbf{1.000} & 0.258 \\
-0.029 & -0.091 & 0.212 & 0.201 & \mathbf{0.622} & \mathbf{0.559} & 0.258 & \mathbf{1.000}
\end{pmatrix}$$

Items 5, 6, and 8, i.e., SCOAP $SC0$, SCOAP $SC1$, and distance as shown in Table 7.1, displayed strong correlation as highlighted in bold-green color in the PCC matrix. The correlation of SCOAP $SC0$ and distance was also observed in Fig. 7.2. Six major PCs in this study were based on PCC, as five were weakly correlated and were assumed uncorrelated, and of the remaining three, two can be dropped. Thus, only six PCs were used to train the ANN, and to facilitate algorithmic decisions in ATPG.

7.1.3 Preprocessing and ATPG

I recorded 8 features listed at the beginning of this section for all signal lines in the three training circuits, c6288, c3540, and b05, in a $L \times 8$ matrix, where L was total number of lines in the three circuits. Next, PCA created the 8×8 matrix appearing above. COP-based ATPG was run on the training circuits to record training data along with major PCs for all backtraced lines. Each backtrace was also labeled either as success if it leads to a test, or failure if it was undone by a backtrack. An ANN was then trained and replaced the conventional heuristics-based sub-routine of PODEM ATPG. When backtracing through a gate with multiple inputs, the ANN rated the chance for success for each input, and the input with highest rating was chosen. To prepare a circuit under test for ATPG, first, all six features were computed for each line, and the values were transformed into PCs in a similar way as was done for training circuits.

7.2 Experimental Results

In this chapter, a workstation containing an Intel i7-8700 processor and 8 GB of RAM performed all experiments. Tools were implemented in C++ using the MSVC++ 14.15 compiler with maximum performance optimization, and all PCA and ANN activities were executed in Python. PODEM ATPG [21] was reproduced along with event-driven fault simulation [4] such that any heuristic (distance [21], COP [108], SCOAP [109], or PC) can be applied across ISCAS'85 [110] and ITC'99 [111] benchmark circuits without favoring a single heuristic. This chapter will surely polarize EDA vendor mindset to deploy MI in their ATPG software. However, EDA vendors hesitate to divulge their program source code, and it is impossible to conduct research-based experiments using the executable. Therefore, I prefer to run the experiments using the in-house EDA tools.

ATPG was applied to all testable and redundant single stuck-at faults in each circuit. Figures 7.3 and 7.4 show the total number of backtracks and ATPG CPU times (ms) for three ANN-based guidances: basic trained-ANN of Chapter 4 [2], optimally-trained-ANN guidance of Chapter 5 [3], and PCA-trained-ANN guidance [15] of this chapter. The circuits are arranged left to right in terms of increasing logic depth. For each circuit, three bars record total backtracks in Fig. 7.3 and three points show CPU times in Fig. 7.4, corresponding to the three versions of PODEM. Curves in Fig. 7.4 are MATLAB power-law curve fits ($y = cx^b$) for the three PODEM versions. For circuits b10, b12, c880, b04, b11, c1908, c7552, c5315, and c3540 the new ANN outperformed the other two ANN-based heuristics [2,3] in terms of backtracks and ATPG CPU time. Circuits c2670, b07, and c3540 show reduced backtracks but required more backtraces, alleviating the CPU time benefit. Circuits c432, c2670, b07, b13, c6288, b09, b03, c499, b08, b13, and c1355 show that quite often the PCA-trained-ANN guided PODEM ATPG gave the best guidance, but when did not it was never the worst performing. Circuits c17, b02, b01, and b06 had zero reconvergent fanouts and provided

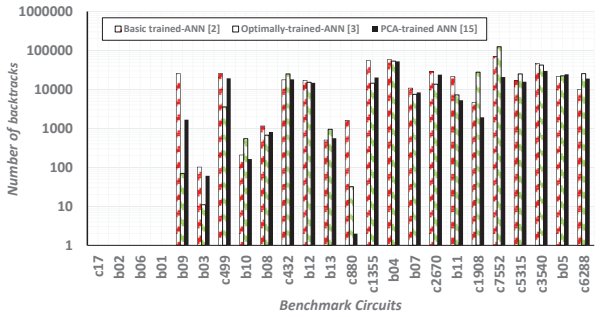

Figure 7.3: Backtracks required to find a test or verify redundancy for all checkpoint stuck-at faults in benchmark circuits, arranged left to right in order of increasing logic depth.

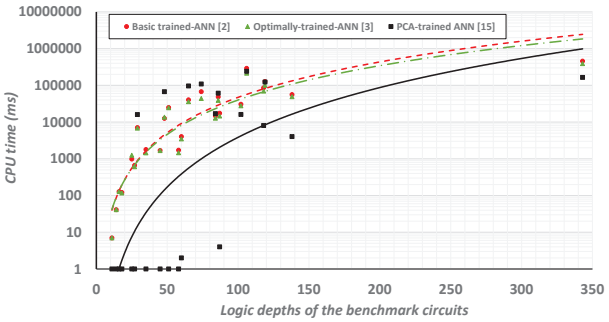

Figure 7.4: CPU time to find a test or verify redundancy for all checkpoint stuck-at faults for circuits of Fig. 7.3.

no scope for reducing backtracks through the new ANN. Circuit b05 had no reduction in backtracks but a significant reduction in CPU time.

The result of circuit b05 is significant in terms of the ability to achieve the so-called "sweet-spot" [2, 3]. In extreme cases, MI-based ATPG can reduce several backtracks to one backtrack in an enormous amount of time, which is infeasible. However, if MI-based ATPG can generate test vectors using the most effective backtraces irrespective of several backtracks in a lesser amount of time than conventional heuristic-based ATPG–it can be worthwhile exploring. The point at which MI-based ATPG may consume backtracks in the lowest possible time is known as "sweet-spot"–effective for detecting hard-to-detect faults in a circuit.

Finally, Fig. 7.5 compares CPU times (ms) of PODEM ATPG guided by the four ML strategies developed, respectively, in Chapter 4 (basic trained-ANN-guidance) [2], Chapter 5 (optimally-trained-ANN guidance) [3], Chapter 6 (PCA-guidance) [14], and this chapter (PCA-trained-ANN guidance) [15]. Once again, benchmarks are arranged by logic depth increasing from left to right. The test generation time of optimally-trained-ANN guidance was reduced a little bit for high depth benchmarks compared to that of the basic trained-ANN guidance. However, it can be observed that the test generation time of PCA-guidance was reduced substantially for low depth benchmarks but increased for high depth benchmarks.

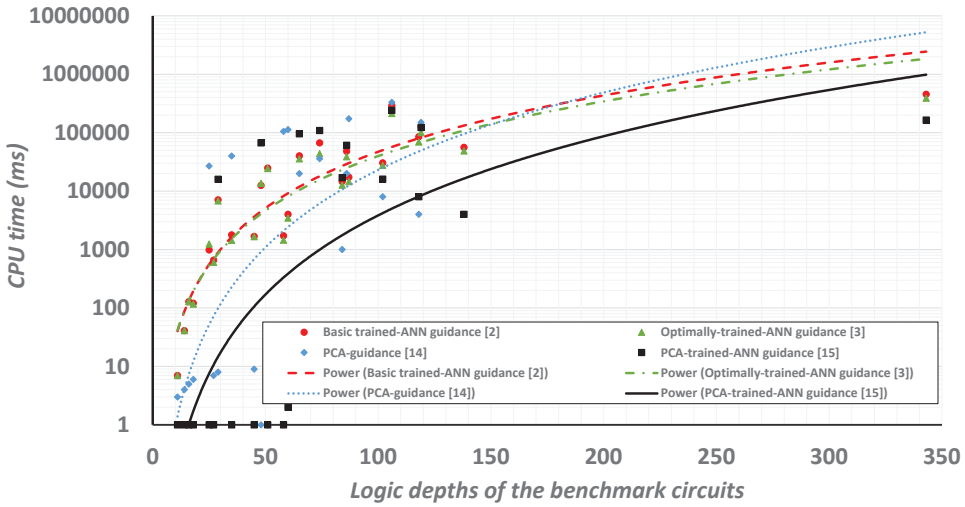

Figure 7.5: CPU times (ms) of PODEM ATPG guided by four ML strategies for all checkpoint single stuck-at faults (detected and redundant).

In contrast, the test generation time of PCA-trained-ANN guidance was reduced uniformly across all the benchmarks. Therefore, it can be concluded that the PCA-trained-ANN guidance of this chapter is the best of the present research.

The preceding evaluation is based on a combined ATPG performance (backtracks and CPU time) for all faults. However, in practical applications a more important criteria is the performance with respect to the hardest-to-detect or even redundant faults. Thus, a fault-by-fault micro-evaluation of the ATPG guidance techniques is recommended for the future.

Chapter 8

Discussion and Future Work

MI, big data, and data mining are hot-topics with ample media coverage, upcoming start-up companies, and outstanding mergers and acquisitions. In the past few decades, MI allowed the extraction and use of domain-specific knowledge to solve computationally hard problems. VLSI design and test have benefited too: MI has been in use for analog, digital, and memory testing, along with emerging technology-based device test and hardware security [51].

This thesis discovered that one can solve the test generation problem using MI. Assuming that fewer backtracks represent higher ATPG performance, I made an interesting observation from Fig. 4.5. For the normalized data, the height of a bar above the 0 line indicates how much worse a heuristic performs compared to MAR. Also, the relative heights of other bars for the same circuit give a comparison between the Distance and COP heuristics. A shorter bar, such as for b09, shows Distance is better than COP. On the other hand, a shorter orange bar for b07 shows a better performance by the COP heuristic. This indicates that there is no single best heuristic for all circuits [106, 107]. It is not surprising that MAR, with capability to use data from both heuristics, does well on both circuits.

I conjectured that any ATPG algorithm that performs backtracing will improve with the MAR heuristic. The ANNs of Chapter 4 and Chapter 5 [2, 3] do not need to be retrained for different ATPG algorithms as long as the goal of the backtrace guidance is to prevent future backtracks and the backtracing subroutine inputs are solely circuit topology. Given that ATPG algorithms beyond PODEM [21], like SOCRATES [24–26] and FAN [22], substantially decrease CPU time through advanced ATPG subroutines but still require backtracking, I hope to see MAR used in these algorithms and provide similar benefits as seen in this

dissertation. I also found that training with just hard-to-detect faults was not sufficient for obtaining a more useful ANN [3]. Therefore, I had to include some easy-to-test faults in the training process. A possible reason is that hard-to-detect faults may cover only some parts of the circuit topology while ignoring others.

Nevertheless, the proposed MI techniques [2, 3] were supervised and needed ANNs that are trained and used in place of heuristics. The result is fewer erroneous backtraces (i.e., fewer backtracks) and more efficient ATPG. But, the increase in the volume of heuristic data may overburden the ANN, making it more complex and leading to increased training time. Chapter 6 used no ANN, and instead opted for a statistical data analysis technique, i.e., PCA, which can be used with minimal cost to implement PODEM ATPG. This PCA-guided ATPG [14] outperforms conventional heuristics in terms of backtracks and CPU time, and circuit c880 (although small) shows the possibility of no backtracks using a linear combination of multiple features. Though previous work [106,107] has reported that no single heuristic performs well in all cases, results of Chapter 6 showed that the major PC combined multiple circuit data effectively and either outperforms or matches the best heuristics on most circuits, with few exceptions, as shown in Figures 6.3 and 6.4.

Chapter 7 attempted to improve upon the previous MI-based test generation system's detriments by introducing PCA [40, 41]. I think that the ANN of MI-based test generation systems may have incorporated too many features, and their large number could have been a detriment in previous research. PCA has a powerful ability to combine even larger-dimensional correlated data, reduce the data volume, and continue to improve the ANN training efficiency. Chapter 7 went beyond expectation by showing an order of magnitude reduction in test generation time (except for c6288) through effectively combining multiple circuit data.

Chapters 4, 5, 6, and 7 provided several avenues to be explored by future researchers [2, 3, 14, 15]:

1. My focus was on reducing backtracks, but the performance of backtraces, particularly in reconvergent fanout-free circuits, can also be improved.

2. My exploration for eliminating backtracks had a cost in CPU time. Thus, a "sweet-spot" may be found where the reduction of backtracks would be optimum for minimizing CPU time.

3. Finding untestable/redundant faults earlier can make ATPG for the entire circuit faster.

4. Recent work [76] demonstrated that arbitrary random circuits can generate limitless training data.

5. This dissertation's experiments were demonstrated on academic benchmark circuits and not on larger industrial circuits. I believe that the MI-guided ATPG performance trends are quite promising (which is important) and likely to scale on a wider variety and larger set of designs to show broader capabilities in the future.

6. MI was used in backtracing guidance of PODEM ATPG, but never used for D-Frontier drive selection in PODEM ATPG to witness more ATPG performance improvement.

7. Using k-means clustering or other unsupervised learning models may give some interesting observations compared to PCA-based ATPG.

8. MI-based PODEM ATPG detected some faults in a circuit with many backtracks compared to conventional heuristic based PODEM ATPG. I think further investigation is needed regarding the characteristics of those faults that did not favor MI-based PODEM ATPG.

9. PCA can combine multiple circuit topological data and testability measures to create a novel testability measure.

10. Although performance improvements diminish for larger circuits, further improvement may be possible with more ANN features, such as reconverging signal characteristics, fanout information, etc., that can add to the capability of the ANN.

11. Yet another area is to examine ANN structures beyond the single hidden layer [43].

Finally, the efficacy of PCs, as illustrated in Chapters 6 and 7, can always be improved by maximizing the explained variance. The usual practice of PCA is to keep only the first $k < p$ principal components, where k is the dimension of transformed subspace that comprises PCs and p is the dimension of the original space. PCA is an orthogonal transformation that projects data from a p-dimensional space to a k-dimensional subspace, and the remaining $p - k$ dimensions vanish in this kind of projection. It is rational to minimize variability in those $p-k$ directions and maximize the variance of the first k variables as the total variance of both p and k dimensional spaces is constant. Figure 6.1 shows that the major PC (or the first PC) of some circuits do not reach close to 1, which signifies that there is room to maximize the explained variance of such circuits by adding more isomorphic features to the original dataset. In the future, one will make choices in PCA based on specific objectives: 1) PCA is an orthogonal transformation and will need maximum variance in the first k components and minimum variance in $p-k$ components; 2) choosing the first k components for maximum variability; and 3) choosing large k to reduce information loss and variance of $p - k$ components.

Researchers are limited in their imagination to find solutions for NP-complete problems. Therefore, all ATPG algorithms use various heuristics to achieve a lower test generation runtime, but achieving lower ATPG run-time is still an open problem. This dissertation attempts to bank upon MI to combine various ATPG inputs as heuristics, narrow down the search space, and achieve speed-up in ATPG runtime as illustrated in Chapters 4, 5, 6, and 7 [2, 3, 14, 15]. However, with the dramatic rise in research on quantum computing, it is only natural to use those ideas to break the VLSI Testing area out of its plateau. The discovery of quantum-based test generation algorithms may break the computational barrier

and achieve the theoretical run-time complexity of $\sqrt{N}$. Until or unless it is proven that a given solution is the most optimal, there is always a ray of hope that paradigms will continue to shift. Attempts have been reported but the field is still open [125, 126].

Finally, practical ATPG systems may combine a simple program (e.g., random vector ATPG) for easy faults and a complex program (e.g., MI-based or quantum-based ATPG) for hard-to-detect faults. This system-level experiment is yet to be explored in the future.

Chapter 9

Conclusion

In this dissertation, a thorough study was conducted to quantitatively and qualitatively analyze the effectiveness of various MI-based heuristics in ATPG. A major benefit of the MI approach is the ability to combine the benefits of multiple heuristics, which may be difficult otherwise. Minimizing test generation time has been the sole motive for many IC test researchers in the past decades. As the time to generate tests depends on the anatomy of the ATPG algorithm, innovations can improve the algorithm's efficacy. The use of heuristics in backtrace is one such technique that attempts to find a test with minimal bad decisions or "***backtracks***" and more successful "***backtraces***." These two ATPG activities play significant roles in the way the search space is explored for finding tests while trying to minimize the CPU time. The search for a test is terminated as soon as a suitable vector is found, making the exploration of the remaining space unnecessary. Conventional heuristics attempt to enhance ATPG performance, but with the introduction of MI, this attempt can be further improved to an extent a single conventional heuristic can never achieve [106, 107].

Chapter 4 concluded using ANN-based backtracing, "MAR", most often reduced backtracks and CPU time compared to other conventional heuristics, which suggests ANNs can assist complex EDA problems like ATPG with few drawbacks.

Chapter 5 concluded that the methodical training of an ANN for guiding ATPG as presented here has benefits. Although many cases showed significant improvements, there were circuits that demand more. Finding the most suitable training circuits remains an open problem. ANN training is only a one-time cost, after which the MI imparted to the ATPG can have long-term benefits.

For the first time, Chapter 6 attempted to integrate unsupervised learning of MI with PODEM ATPG to demonstrate the effectiveness of ATPG in terms of reduction of backtracks and ATPG CPU time. This chapter used PCA to combine multiple features, and a linear transformation projects conventional ATPG backtracing features into a new major PC (the first PC), which is considered to be the carrier of maximum variance and replaces the traditional single-heuristic guidance of ATPG.

To be successful, an ATPG algorithm must backtrack when necessary and then move forward again and it's efficiency is derived by minimizing backtracks. Chapter 7 concluded that PCA effectively increased the amount of useful information fed into an ANN during ATPG. It brings ATPG closer to the elusive goal of zero backtrack. This chapter showed that PCA reduced the dimension of the training dataset and effectively trained the ANN. This is known as *PCA-assisted supervised learning* in contrast with the supervised learning [2, 3].

Bibliography

[1] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. on Electronic Comp.*, vol. EC-16, no. 5, pp. 567–580, Oct. 1967.

[2] S. Roy, S. K. Millican, and V. D. Agrawal, "Machine Intelligence for Efficient Test Pattern Generation," in *Proc. IEEE Int'l Test Conf.*, Washington D.C, Nov. 2020.

[3] S. Roy, S. K. Millican, and V. D. Agrawal, "Training Neural Network for Machine Intelligence in Automatic Test Pattern Generator," in *Proc. 34th Int'l Conf. on VLSI Design & 20th Int'l Conf. on Embedded Systems*, India, February 2021.

[4] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Publishing Company, Incorporated, 2013.

[5] P. K. Nag, A. Gattiker, S. Wei, R. D. Blanton, and W. Maly, "Modeling the Economics of Testing: A DFT Perspective," *IEEE Design & Test of Comp.*, vol. 19, no. 1, pp. 29–41, Jan/Feb 2002.

[6] R. D. Eldred, "Test Routines Based on Symbolic Logical Statements," *J. ACM*, vol. 6, no. 1, pp. 33–37, Jan. 1959.

[7] J. M. Galey, R. E. Norby, and J. P. Roth, "Techniques for the Diagnosis of Switching Circuit Failures," in *Proc. 2nd Annual Symp. on Switching Circuit Theory and Logical Design (SWCT)*, 1961, pp. 152–160.

[8] J. M. Galey, R. E. Norby, and J. P. Roth, "Techniques for the Diagnosis of Switching Circuit Failures," *IEEE Trans. on Communication and Electronics*, vol. 83, no. 74, pp. 509–514, Sep. 1964.

[9] G. E. Moore, "Cramming More Components onto Integrated Circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp. 114 ff." *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, 2006.

[10] O. H. Ibarra and S. K. Sahni, "Polynomially Complete Fault Detection Problems," *IEEE Trans. on Comp.*, vol. C-24, no. 3, pp. 242–249, Mar. 1975.

[11] H. Fujiwara and S. Toida, "The Complexity of Fault Detection Problems for Combinational Logic Circuits," *IEEE Trans. on Comp.*, vol. 31, no. 6, pp. 555–560, Jun. 1982.

[12] G. Seroussi and N. H. Bshouty, "Vector Sets for Exhaustive Testing of Logic Circuits," *IEEE Trans. Information Theory*, vol. 34, no. 3, pp. 513–522, May 1988.

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

[14] S. Roy, S. K. Millican, and V. D. Agrawal, "Unsupervised Learning in Test Generation for Digital Integrated Circuits," in *Proc. IEEE European Test Symp.*, Belgium, May 2021.

[15] S. Roy, S. K. Millican, and V. D. Agrawal, "Principal Component Analysis in Machine Intelligence-Based Test Generation," in *Proc. IEEE Microelectronics Design and Test Symp. (MDTS'21)*, USA, May 2021.

[16] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM J. Res. Dev.*, vol. 10, no. 4, pp. 278–291, Jul. 1966.

[17] T. Williams and N. Brown, "Defect Level as a Function of Fault Coverage," *IEEE Trans. on Comp.*, vol. C-30, no. 12, pp. 987–988, Dec. 1981.

[18] M. Turner, "Testing CMOS VLSI: Tools Concepts, and Experimental Results," in *Proc. Int'l Test Conf.*, 1985, pp. 322–328.

[19] D. B. Armstrong, "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Nets," *IEEE Trans. on Electronic Comp.*, vol. EC-15, no. 1, pp. 66–73, Feb. 1966.

[20] M. A. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital Systems.* Computer Science Press, 1976.

[21] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. on Comp.*, vol. C-30, no. 3, pp. 215–222, Mar. 1981.

[22] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Trans. on Comp.*, vol. C-32, no. 12, pp. 1137–1144, Dec. 1983.

[23] T. Kirkland and M. R. Mercer, "A Topological Search Algorithm for ATPG," in *Proc. 24th ACM/IEEE Design Automation Conf.*, Jun. 1987, pp. 502–508.

[24] M. H. Schulz and E. Auth, "Advanced Automatic Test Pattern Generation and Redundancy Identification Techniques," *Proc. Eighteenth Int'l Symp. on Fault-Tolerant Computing. Digest of Papers*, pp. 30–35, June 1988.

[25] M. H. Schulz and E. Auth, "Improved Deterministic Test Pattern Generation With Applications to Redundancy Identification," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 7, pp. 811–816, Jul. 1989.

[26] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 1, pp. 126–137, Jan. 1988.

[27] M. L. Bushnell and J. Giraldi, "A Functional Decomposition Method for Redundancy Identification and Test Generation," *J. Electronic Testing*, vol. 10, no. 3, pp. 175–195, Jun. 1997.

[28] K.-T. Cheng, "On Removing Redundancy in Sequential Circuits," in *Proc. 28th ACM/IEEE Design Automation Conf. (DAC)*, June 1991, pp. 164–169.

[29] K.-T. Cheng and V. D. Agrawal, *Unified Methods for VLSI Simulation and Test Generation*. Springer, 1989.

[30] W. Kunz and D. K. Pradhan, "Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation in Digital Circuits," in *Proc. IEEE Int'l Test Conf.*, USA, Sept. 1992, pp. 816–825.

[31] S. T. Chakradhar and V. D. Agrawal, "A Transitive Closure Based Algorithm for Test Generation," in *Proc. 28th ACM/IEEE Design Automation Conf.*, ser. DAC '91. USA: Association for Computing Machinery, June 1991, pp. 353–358.

[32] S. T. Chakradhar, V. D. Agrawal, and S. G. Rothweiler, "A Transitive Closure Algorithm for Test Generation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 7, pp. 1015–1028, Jul. 1993.

[33] T. Larrabee, "Efficient Generation of Test Patterns Using Boolean Difference," in *Proc.Int'l Test Conf.*, USA, Aug. 1989, pp. 795–801.

[34] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Trans. on CAD*, vol. 11, no. 1, pp. 4–15, Jan. 1992.

[35] S. T. Chakradhar, "Neural network models and optimization methods for digital testing," Ph.D. dissertation, Rutgers University, USA, 1991.

[36] J. P. Marques Silva and K. A. Sakallah, "GRASP - A New Search Algorithm for Satisfiability," in *Proc. Int'l Conf. on Computer Aided Design*, USA, Apr. 1996, pp. 220–227.

[37] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Combinational Test Generation Using Satisfiability," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 1167–1176, 1996.

[38] M. Henftling, H. Wittmann, and K. J. Antreich, "A Formal Non-Heuristic ATPG Approach," in *Proc. Conf. on European Design Automation*, ser. EURO-DAC '95/EURO-VHDL '95. England: IEEE Computer Society Press, Sept. 1995, pp. 248–253.

[39] P. Tafertshofer, A. Ganz, and M. Henftling, "A SAT-based Implication Engine for Efficient ATPG, Equivalence Checking, and Optimization of Netlists," in *Proc. IEEE Int'l Conf. on Computer Aided Design (ICCAD)*, USA, Nov. 1997, pp. 648–655.

[40] K. Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[41] H. Hotelling, "Analysis of a Complex of Statistical Variables into Principal Components," *Journal of Educational Psychology,*, vol. 24, no. 6, pp. 417–441, 1933.

[42] S. S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Upper Saddle River, NJ: Pearson Education, 2009.

[43] D. K. Hunter, H. Yu, M. S. P. III, J. Kolbusz, and B. M. Wilamowski, "Selection of Proper Neural Network Sizes and Architectures - A Comparative Study," *IEEE Trans. Industrial Informatics*, vol. 8, no. 2, pp. 228–240, 2012.

[44] Y. Lecun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature Cell Biology*, vol. 521, pp. 436–444, 2015.

[45] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," *ArXiv*, vol. abs/1811.03378, 2018.

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

[47] I. Jolliffe, *Principal Component Analysis*. New York: Springer-Verlag New York, 2002.

[48] J. Guo, G. James, E. Levina, G. Michailidis, and J. Zhu, "Principal Component Analysis With Sparse Fused Loadings," *Journal of Computational and Graphical Statistics*, vol. 19, no. 4, pp. 930–946, 2010.

[49] H. Stratigopoulos, "Machine Learning Applications in IC Testing," in *Proc. IEEE 23rd European Test Symp. (ETS)*, Germany, May 2018, pp. 1–10.

[50] M. Pradhan and B. B. Bhattacharya, "A Survey of Digital Circuit Testing in the Light of Machine Learning," *WIREs Data Mining Knowl. Discov.*, pp. 1–18, 2020.

[51] S. Roy, S. K. Millican, and V. D. Agrawal, "Special Session – Machine Learning in Test: A Survey of Analog, Digital, Memory, and RF Integrated Circuits," in *Proc. IEEE VLSI Test Symp. (VTS'21)*, USA, Apr. 2021, pp. 1–10.

[52] H. Stratigopoulos and S. Mir, "Adaptive Alternate Analog Test," *IEEE Design Test of Comp.*, vol. 29, no. 4, pp. 71–79, 2012.

[53] H. Stratigopoulos and S. Sunter, "Fast Monte Carlo-Based Estimation of Analog Parametric Test Metrics," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1977–1990, 2014.

[54] H. Stratigopoulos, "Test Metrics Model for Analog Test Development," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 1116–1128, 2012.

[55] D. Maliuk, H.-G. Stratigopoulos, H. Huang, and Y. Makris, "Analog Neural Network Design for RF Built-In Self-Test," in *Proc. Int'l Test Conf. (ITC)*, USA, Nov. 2010, pp. 23.2.1–23.2.10.

[56] D. Banerjee, S. K. Devarakond, X. Wang, S. Sen, and A. Chatterjee, "Real-Time Use-Aware Adaptive RF Transceiver Systems for Energy Efficiency Under BER Constraints," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1209–1222, 2015.

[57] C. Xanthopoulos, P. Sarson, H. Reiter, and Y. Makris, "Automated Die Inking: A Pattern Recognition-based Approach," in *Proc. IEEE Int'l Test Conf. (ITC)*, USA, Oct. 2017, pp. 1–6.

[58] J. Tikkanen, S. Siatkowski, N. Sumikawa, L. Wang, and M. S. Abadir, "Yield Optimization using Advanced Statistical Correlation Methods," in *Proc. Int'l Test Conf.*, USA, Oct. 2014, pp. 1–10.

[59] N. Sumikawa, M. Nero, and L. Wang, "Kernel Based Clustering for Quality Improvement and Excursion Detection," in *Proc. IEEE Int'l Test Conf. (ITC)*, USA, Oct. 2017, pp. 1–10.

[60] Y. Huang, R. Guo, W. Cheng, and J. C. Li, "Survey of Scan Chain Diagnosis," *IEEE Design Test of Comp.*, vol. 25, no. 3, pp. 240–248, 2008.

[61] 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 *Proc. 24th Asia and South Pacific Design Automation Conf. (ASP-DAC)*, Tokyo, Japan, Jan. 2019, pp. 341–346.

[62] H. Wang, O. Poku, X. Yu, S. Liu, I. Komara, and R. D. Blanton, "Test-Data Volume Optimization for Diagnosis," in *Proc. Design Automation Conf.*, USA, June 2012, pp. 567–572.

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

[64] J. E. Nelson, W. C. Tam, and R. D. Blanton, "Automatic Classification of Bridge Defects," in *Proc. IEEE Int'l Test Conf.*, USA, Nov. 2010, pp. 1–10.

[65] L. R. Gómez, A. Cook, T. Indlekofer, S. Hellebrand, and H.-J. Wunderlich, "Adaptive Bayesian Diagnosis of Intermittent Faults," *J. Electron. Test.*, vol. 30, no. 5, p. 527–540, Oct. 2014.

[66] Y. Xue, O. Poku, X. Li, and R. D. Blanton, "PADRE: Physically-Aware Diagnostic Resolution Enhancement," in *Proc. IEEE Int'l Test Conf. (ITC)*, USA, Sept. 2013, pp. 1–10.

[67] Z. Sun, L. Jiang, Q. Xu, Z. Zhang, Z. Wang, and X. Gu, "AgentDiag: An Agent-Assisted Diagnostic Framework for Board-level Functional Failures," in *Proc. IEEE Int'l Test Conf. (ITC)*, USA, Sept. 2013, pp. 1–8.

[68] Z. Sun, L. Jiang, Q. Xu, Z. Zhang, Z. Wang, and X. Gu, "On Test Syndrome Merging for Reasoning-based Board-level Functional Fault Diagnosis," in *Proc. 20th Asia and South Pacific Design Automation Conf.*, Japan, Sept. 2015, pp. 737–742.

[69] F. Ye, Z. Zhang, K. Chakrabarty, and X. Gu, "Board-Level Functional Fault Diagnosis Using Artificial Neural Networks, Support-Vector Machines, and Weighted-Majority Voting," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 723–736, 2013.

[70] S. Wang and W. Wei, "Machine Learning-Based Volume Diagnosis," in *Proc. Design, Automation & Test in Europe Conf. & Exhibition*, France, Apr. 2009, pp. 902–905.

[71] L. M. Huisman, M. Kassab, and L. Pastel, "Data Mining Integrated Circuit Fails with fail Commonalities," in *Proc. Int'l Test Conf.*, USA, Oct. 2004, pp. 661–668.

[72] W. Cheng, Yue Tian, and S. M. Reddy, "Volume Diagnosis Data Mining," in *Proc. 22nd IEEE European Test Symp. (ETS)*, Cyprus, May 2017, pp. 1–10.

[73] Z. Li, J. E. Colburn, V. Pagalone, K. Narayanun, and K. Chakrabarty, "Test-Cost Optimization in a Scan-Compression Architecture Using Support-Vector Regression," in *Proc. IEEE 35th VLSI Test Symp. (VTS)*, USA, Apr. 2017, pp. 1–6.

[74] J. Immanuel and S. K. Millican, "Calculating signal controllability using neural networks: Improvements to testability analysis and test point insertion," in *Proc. IEEE 29th North Atlantic Test Workshop (NATW)*, USA, May 2020, pp. 1–6.

[75] 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 Conf. (DAC)*, USA, June 2019, pp. 1–6.

[76] 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)*, India, Dec. 2019, pp. 13–18.

[77] M. Pradhan, B. B. Bhattacharya, K. Chakrabarty, and B. B. Bhattacharya, "Predicting X-Sensitivity of Circuit-Inputs on Test-Coverage: A Machine-Learning Approach," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2343–2356, 2019.

[78] Y. Liu, C. Han, S. Lin, and J. C. Li, "PSN-Aware Circuit Test Timing Prediction Using Machine Learning," *IET Comp. Digital Techniques*, vol. 11, no. 2, pp. 60–67, 2017.

[79] C. FAGOT, P. GIRARD, and C. LANDRAULT, "On using machine learning for logic bist," in *Proc. IEEE Int'l Test Conf.* USA: IEEE Computer Society, 1997, p. 338.

[80] M. Sadi, G. K. Contreras, J. Chen, L. Winemberg, and M. Tehranipoor, "Design of Reliable SoCs With BIST Hardware and Machine Learning," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 11, pp. 3237–3250, 2017.

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

[82] A. Manzini, P. Inglese, L. Caldi, R. Cantero, G. Carnevale, M. Coppetta, M. Giltrelli, N. Mautone, F. Irrera, R. Ullmann, and P. Bernardi, "A Machine Learning-Based Approach to Optimize Repair and Increase Yield of Embedded Flash Memories in Automotive Systems-on-Chip," in *Proc. IEEE European Test Symp. (ETS)*, Germany, May 2019, pp. 1–6.

[83] P. Mazumder and Y.-S. Jih, "A New Built-In Self-Repair Approach to VLSI Memory Yield Enhancement by Using Neural-Type Circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 1, pp. 124–136, Jan. 1993.

[84] A. Singhee and R. A. Rutenbar, "Statistical Blockade: Very Fast Statistical Simulation and Modeling of Rare Circuit Events and Its Application to Memory Design," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 8, pp. 1176–1189, 2009.

[85] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support Vector Machines," *IEEE Intelligent Systems and Their Applications*, vol. 13, no. 4, pp. 18–28, 1998.

[86] A. Heuser and M. Zohner, "Intelligent Machine Homicide," in *Proc. Third Int'l Conf. on Constructive Side-Channel Analysis and Secure Design*, ser. COSADE'12. Germany: Springer-Verlag, May 2012, p. 249–264.

[87] G. Hospodar, B. Gierlichs, E. Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: A first study," *J. Cryptographic Engineering*, vol. 1, pp. 293–302, 12 2011.

[88] K. Huang, J. M. Carulli, and Y. Makris, "Parametric counterfeit IC detection via Support Vector Machines," in *Proc. IEEE Int'l Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, USA, Oct. 2012, pp. 7–12.

[89] T. Iwase, Y. Nozaki, M. Yoshikawa, and T. Kumaki, "Detection technique for hardware Trojans using machine learning in frequency domain," in *Proc. IEEE 4th Global Conf. on Consumer Electronics (GCCE)*, 2015, pp. 185–186.

[90] D. Jap, M. Stöttinger, and S. Bhasin, "Support Vector Regression: Exploiting Machine Learning Techniques for Leakage Modeling," in *Proc. Fourth Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '15. New York, NY, USA: Association for Computing Machinery, 2015.

[91] Y. Jin, D. Maliuk, and Y. Makris, "Post-deployment trust evaluation in wireless cryptographic ICs," in *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, Germany, Mar. 2012, pp. 965–970.

[92] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," in *Proc. 14th Int'l Joint Conf. on Artificial Intelligence - Volume 2*, ser. IJCAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, p. 1137–1143.

[93] N. Asadizanjani, M. Tehranipoor, and D. Forte, "Counterfeit Electronics Detection Using Image Processing and Machine Learning," *Journal of Physics: Conf. Series*, vol. 787, p. 012023, Jan. 2017.

[94] H. Dogan, D. Forte, and M. M. Tehranipoor, "Aging analysis for recycled FPGA detection," in *Proc. IEEE Int'l Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Netherlands, Oct. 2014, pp. 171–176.

[95] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-Means Clustering Algorithm." *Journal of the Royal Statistical Society*, vol. 28, pp. 100–108, Nov. 1979.

[96] M. M. Alam, M. Tehranipoor, and D. Forte, "Recycled FPGA Detection Using Exhaustive LUT Path Delay Characterization," in *Proc. IEEE Int'l Test Conf. (ITC)*, USA, Nov. 2016, pp. 1–10.

[97] C. Bao, D. Forte, and A. Srivastava, "On Application of One-Class SVM to Reverse Engineering-Based Hardware Trojan Detection," in *Proc. 15th Int'l Symp. Quality Electronic Design*, USA, Mar. 2014, pp. 47–54.

[98] C. Bao, D. Forte, and A. Srivastava, "On Reverse Engineering-Based Hardware Trojan Detection," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 49–57, 2016.

[99] J. Li, J. Cheng, J. Shi, and F. Huang, "Brief Introduction of Back Propagation (BP) Neural Network Algorithm and Its Improvement," in *Advances in Computer Science and Information Engineering*, 2012, pp. 553–558.

[100] J. Li, L. Ni, J. Chen, and E. Zhou, "A Novel Hardware Trojan Detection Based on BP Neural Network," in *Proc. 2nd IEEE Int'l Conf. on Computer and Communications (ICCC)*, China, Jul. 2016, pp. 2790–2794.

[101] Y. Liu, Y. Jin, A. Nosratinia, and Y. Makris, "Silicon Demonstration of Hardware Trojan Design and Detection in Wireless Cryptographic ICs," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1506–1519, Apr. 2017.

[102] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking Cryptographic Implementations Using Deep Learning Techniques," in *IACR Cryptol.*, USA, Dec. 2016, p. 921.

[103] R. L. Rivest, "Learning Decision Lists," in *Mach. Learn.*, vol. 2, no. 3. USA: Kluwer Academic Publishers, Nov. 1987, p. 229–246.

[104] H. Peng, F. Long, and C. Ding, "Feature Selection Based on Mutual Information Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226–1238, Aug. 2005.

[105] H. Salmani, "COTD: Reference-Free Hardware Trojan Detection and Recovery Based on Controllability and Observability in Gate-Level Netlist," *IEEE Trans. on Information Forensics and Security*, vol. 12, no. 2, pp. 338–350, Dec. 2017.

[106] J. H. Patel and S. Patel, "What Heuristics are Best for PODEM?" in *Proc. First Int'l Workshop on VLSI Design*, Chennai, India, Dec. 1985, pp. 1–20.

[107] S. Patel and J. H. Patel, "Effectiveness of Heuristics Measures for Automatic Test Pattern Generation," in *Proc. 23rd ACM/IEEE Design Automation Conf.*, ser. DAC '86, USA, Jun. 1986, p. 547–552.

[108] F. Brglez, "On Testability Analysis of Combinational Circuits," in *Proc. Int'l Symp. Circuits and Systems*, Canada, June 1984, pp. 221–225.

[109] L. Goldstein, "Controllability/Observability Analysis of Digital Circuits," *IEEE Trans. on Circuits and Systems*, vol. 26, no. 9, pp. 685–693, Sep. 1979.

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

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

[112] R. Kemker, A. Abitino, M. McClure, and C. Kanan, "Measuring Catastrophic Forgetting in Neural Networks," in *AAAI*, New York, 2018.

[113] J. E. Jackson, *A User's Guide to Principal Components*. New York: Wiley, 1991.

[114] K. Diamantaras and S. Kung, *Principal Component Neural Networks: Theory and Applications*. New York: Wiley, 1996.

[115] B. Flury, *Common Principal Components and Related Models*. New York: Wiley, 1988.

[116] Y. Sun, S. K. Millican, and V. D. Agrawal, "Special session: Survey of test point insertion for logic built-in self-test," in *Proc. IEEE 38th VLSI Test Symp. (VTS)*, USA, May 2020, pp. 1–6.

[117] S. Roy, B. Stiene, S. K. Millican, and V. D. Agrawal, "Improved Random Pattern Delay Fault Coverage Using Inversion Test Points," in *Proc. IEEE 28th North Atlantic Test Workshop (NATW)*, VT, May 2019, pp. 206–211.

[118] S. Roy, B. Stiene, S. K. Millican, and V. D. Agrawal, "Improved Pseudo-Random Fault Coverage Through Inversions: A Study on Test Point Architectures," *J. Electron. Test*, vol. 36, no. 1, p. 123–133, Feb. 2020.

[119] J. Olsson, C. B. Uvo, K. Jinno, A. Kawamura, K. Nishiyama, N. Koreeda, T. Nakashima, and O. Morita, "Neural Networks for Rainfall Forecasting by Atmospheric Downscaling," *Journal of Hydrologic Engineering*, vol. 9, no. 1, pp. 1–12, Jan. 2004.

[120] G. J. Bowden, "Forecasting Water Resources Variables using Artificial Neural Networks," Ph.D. dissertation, University of Adelaide, Australia, 2003.

[121] M. Gibbs, N. Morgan, H. R. Maier, G. C. Dandy, J. B. Nixon, and M. Holmes, "Investigation into the Relationship Between Chlorine Decay and Water Distribution Parameters Using Data Driven Methods," *Mathematical and Computer Modelling*, vol. 44, no. 5, pp. 485–498, Sep. 2006.

[122] E. Ranaee, G. Porta, M. Riva, and A. Guadagnini, "Investigation of Saturation Dependency of Oil Relative Permeability during WAG Process through Linear and Non-linear PCA," in *Proc. European Conf. on the Mathematics of Oil Recovery*, no. 1. Italy: European Association of Geoscientists and Engineers, Sep. 2014, pp. 1–14.

[123] I. Jolliffe, *Principal Component Analysis*. Springer-Verlag New York, 2002.

[124] W. Kirch, Ed., *Pearson's Correlation Coefficient*. Dordrecht: Springer Netherlands, 2008, pp. 1090–1091.

[125] M. Venkatasubramanian and V. D. Agrawal, "Quest for a Quantum Search Algorithm for Testing Stuck-at Faults in Digital Circuits," in *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, Amherst, MA, Oct. 2015, pp. 128–133.

[126] M. Venkatasubramanian, "Failure Evasion: Statistically Solving the NP Complete Problem of Testing Difficult-to-Detect Faults," Ph.D. dissertation, Auburn University, Auburn, Alabama, USA, 2016.

Publications from this Dissertation

List of Conference Publications

1. **S. Roy**, B. Stiene, S. K. Millican, and V. D. Agrawal, "Improved Random Pattern Delay Fault Coverage Using Inversion Test Points," in *Proceedings of IEEE 28th North Atlantic Test Workshop* (NATW), Burlington, VT, May 2019.

2. 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 *Proceedings of IEEE 28th Asian Test Symposium* (ATS), India, Dec., 2019, pp. 13-18.

3. **S. Roy**, S. K. Millican, and V. D. Agrawal, "Machine Intelligence for Efficient Test Pattern Generation," in *Proceedings of the IEEE International Test Conference* (ITC), Washington D.C, Nov. 2020.

4. **S. Roy**, S. K. Millican, and V. D. Agrawal, "Training Neural Network for Machine Intelligence in Automatic Test Pattern Generator," in *Proceedings of the 34th International Conference on VLSI Design and the 20th International Conference on Embedded Systems* (VLSID), Virtual Event, Feb. 2021.

5. **S. Roy**, S. K. Millican, and V. D. Agrawal, "Special Session – Machine Learning in Test: A Survey of Analog, Digital, Memory, and RF Integrated Circuits," in *Proceedings of the IEEE VLSI Test Symposium* (VTS), Virtual Event, Apr. 2021.

6. **S. Roy**, S. K. Millican, and V. D. Agrawal, "Unsupervised Learning in Test Generation for Digital Integrated Circuits," in *Proceedings of the IEEE European Test Symposium* (ETS), Belgium, May 2021

7. **S. Roy**, S. K. Millican, and V. D. Agrawal, "Principal Component Analysis in Machine Intelligence-Based Test Generation," in *Proceedings of the IEEE Microelectronics Design and Test Symposium* (MDTS), Albany, NY, May 2021

List of Posters

1. S. K. Millican, Y. Sun, **S. Roy**, and V. D. Agrawal, "Applying Artificial Neural Networks to Test-point Insertion: Delay Fault Coverage and Training Circuit Generation," in *Proceedings of International Test Conference* (ITC), 2019.

2. **S. Roy**, B. Stiene, S. K. Millican, and V. D. Agrawal, "Improved Random Pattern Delay Fault Coverage Using Inversion Test Points," in Auburn Research Student Symposium, Auburn, USA, 2019.

Journal Publication

1. **S. Roy**, B. Stiene, S. K. Millican, and V. D. Agrawal, "Improved Pseudo-Random Fault Coverage Through Inversions: A Study on Test Point Architectures," *J. Electron. Test.*, vol. 36, pp. 123–133, Feb. 2020.

List of US Patents

1. **S. Roy**, S. K. Millican, and V. D. Agrawal, "A Machine Intelligence for Automatic Test Pattern Generation for Digital Logic Circuits," 2020.

2. S. K. Millican, Y. Sun, **S. Roy**, and V. D. Agrawal, "System and Method for Optimizing Fault Coverage Based on Optimized Test Point Insertion Determinations for Logic Circuits," 2020.

Author's Biography

Soham Roy received his Bachelor of Technology (*B.Tech.*) Degree in Electronics and Instrumentation from West Bengal University of Technology, Kolkata, India, in 2011. He was with Wipro Ltd., VLSI Division, Bangalore, India, as a design for test engineer from 2011–2015. He received his Master of Science (*MS*) degree from the Department of Electrical and Computer Engineering, Technical University of Dresden, Dresden, Germany, in 2018. He received his Doctor of Philosophy (*Ph.D.*) in Electrical and Computer Engineering from the Auburn University, USA, in 2021. He has published several articles and has filed patents in applying machine learning in test point insertion (TPI) and automatic test pattern generation (ATPG). He will join Intel Corporation, Hillsboro, USA as a Product Development Engineer. His research interest includes VLSI design and test and artificial intelligence.