Studies on Diagnostic Coverage and X-Sensitivity in Logic Circuits:
Combinatorial and Machine Learning Approaches

Manjari Pradhan

Advisor

Prof. Bhargab B. Bhattacharya

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Indian Statistical Institute
Kolkata 700108, India
April 2020

Dedicated to
My Parents

Acknowledgements

What seemed to be a long and arduous journey, has indeed transpired to be a beautiful, enriching and the most cherished one. As I look back to the years that has led to the completion of this thesis, I realize how fortunate I am to find such kind people around me to whom I will always be grateful. I would like to take this opportunity to express my gratitude to them.

I feel blessed to have worked with my supervisor Prof. Bhargab B. Bhattacharya. His expertise, novel ideas, and above all active participation in research have made this thesis possible. I am grateful to him for patiently correcting my technical reports and papers and also for his kind and patient technical interactions and discussions.

I would like to acknowledge Prof. Krishnendu Chakrabarty and Prof. Bhaswar B. Bhattacharya for their valuable suggestions and guidance. I express my sincere thanks to Prof. Bhaswar B. Bhattacharya for his valuable discussions and contribution in my thesis.

I am very grateful to all my friends and colleagues of ACMU for making the journey enjoyable and memorable. Special mention goes to Sudip-da, Sukanta-da, Sandip-da, Manob, Oishila Di, Animesh, Tapalina and Mr. G. P. Samanta for being such good colleagues during the journey of my Ph.D. I would like to specially thank Sudip-da and Animesh for their kindness and assistance whenever I asked for it.

My Ph.D journey would have been grim without my dear friends Sudipta, Aparajita, Sanchayan, and Avisek. They have always gone out of their way to help me. I always found Sudipta around whenever I needed her. Aparajita was ever ready to help me whenever I struggled with the basics of Machine Learning. She was invariably there to cheer me up when I needed it and made my stay at hostel a joyful experience. I am grateful to Sanchayan for being my best friend and my constant companion. He has been by strength and helped me regain my courage whenever I felt low. Lastly, I would like to thank my friends, seniors, and juniors of our mess for making the meal times so good and memorable.

Above all, this thesis would not have been completed without the love and support of my parents. I am indebted to my parents for their patience, sacrifice and their faith in me. I am also thankful to my brother for his support.

Manjari Pradhan

Abstract

Today's integrated circuits comprise billions of interconnected transistors assembled on a tiny silicon chip, and testing them to ensure functional and timing correctness continues to be a major challenge to designers and test engineers with further downscaling of transistors. Although substantial progress has been witnessed during the last five decades in the area of algorithmic test generation and fault diagnosis, applications of combinatorial and machine-learning (ML) techniques to solve these problems remain largely unexplored till date. In this thesis, we study three problems in the context of digital logic test and diagnosis. The first problem is that of fault diagnosis and can be stated as follows. Given the output syndromes for test responses in a circuit-under-test (CUT), localize the fault (root-cause) in the net-list; solution to this problem is required in order to fine-tune the process technology and to improve yield in subsequent production cycles. The second problem deals with the issue of unknown logic value (X) and answers the following question. Given a CUT and a test set T, how the fault-coverage of T is impacted when an unknown logic value (X) arrives at one of its inputs. Is there any computationally efficient mechanism to grade the CUT-inputs based on their X-sensitivity? A solution to this problem is needed in X-cancellation for optimizing test costs and to improve reliability. In the third problem, we investigate, given a CUT, how the structure of the underlying network can be represented in a compact and lossless form, so as to make them easily readable by ML-tools for test and diagnostic purposes. This is a classical problem of representing directed graphs that facilitates feature extraction. We present a combinatorial solution to the first problem, a learning-based technique to handle the second problem, and a novel solution to the third problem that relies on Prüfer sequence.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER

ONE

INTRODUCTION

Digital logic testing relies heavily on automatic-test-pattern generation (ATPG) which has been well investigated during the last five decades [BA05]. Essentially, the aim of ATPG is to produce a test set such that each modelled fault f in the circuit-under-test (CUT) produces an output response (error) which is not identical to the fault-free response for at least one test vector. Going one step further ahead, if we demand that the output response in the presence of each fault f, be non-identical to the output responses of the rest of the non-equivalent faults, for at least one test vector per fault in T, then we would be able to solve the fault localization problem in the CUT. Alternatively, in some cases, the ATPG may fail to test f because it cannot predict the value of either one or more internal lines or some output response bits, that are sensitive to f. The former, is the aim of a sub-field of digital testing called fault diagnosis. The latter is called the unknown (X) value problem. The presence of unknowns in the output response severely affects the testability, fault coverage and diagnosability of the circuit as well. In this thesis, we address two problems in these domains and propose new solution methodologies. Additionally, we explore various potentials for applying machine learning to this field. For the fault diagnosis problem, we propose a new combinatorial approach. For the X-value problem, we develop a machine-learning based solution for predicting their impact on fault coverage and grade them depending on their sensitivity. In order to enable better training of samples, we propose a new encoding scheme for large graphs that captures the structures of logic circuits. Representation of large graphs for machine learning applications is a challenging problem. This chapter briefly describes the overall flow of this thesis. The motivation behind the work is presented in Section 1.1. Section 1.2 describes the overall scope of the thesis and summarizes our contributions. Finally, Section 1.3 narrates the organization of the thesis.

1.1 Motivation

With a history of over five decades, the digital integrated circuit (IC) technology has traveled through a long path, starting from the invention of transistors in 1954. While the technology has immensely matured, it has not stopped pushing new boundaries. In fact, the Moore's law of 1965 holds true till today, with the current technology reaching less than $5nm$. This rapid downscaling of process technology has given way to enormous level of integration. Such progress has not only miniaturized the size of digital chips but also improved their speed. This, in turn, has opened new windows of applications that demand high computation and memory. The level of miniaturization and the scale of integration, both inevitably have led to various situations, which pose several challenges to the IC industry.

1. The manufacturing process is required to be fine-tuned for every new technology in order to obtain good yield. An important step in this process is a logic-level process called fault diagnosis.
2. Unknown (X) values in circuit nodes appear frequently. Their presence not only degrade the detectability of the faults in the circuit but also affect test cost.
3. The defect complexity has grown significantly. As a result, the faulty behavior of a circuit has become probabilistic in nature.
4. The circuit size has increased manifold. This necessitates the need for fast techniques for circuit testing.

Machine-learning (ML) techniques provide good solutions in the last two situations.

Before a new chip or technology is introduced, many failures are experienced during the early stages. So, the yield (fraction of good chips) is low. In order to improve the yield, the manufacturing process needs to be tuned. This requires a study of the limitations of the present manufacturing process in order to take corrective measures, which is called yield learning. At the lowest level, it involves physical failure analysis (PFA), wherein the chip is physically inspected. Since this process is time consuming, failure analysis (FA) is executed before PFA to locate the faulty block. It is a process of investigating the root-cause of failure [OY18]. That is, the circuit need to be scanned to pin-point the defective region, and to classify the defect type. FA is further assisted by a logical level process called fault diagnosis. It guides and speeds up FA, by analyzing the failure log of a chip. It then produces a set of faults which are possible candidates for the actual defect. There are two types of fault diagnosis techniques: those based on cause-effect or effect-cause. The cause-effect technique relies on a dictionary-based approach whereas the effect-cause approach follows a simulation-based, inject and evaluate approach. A simple flow of the yield improvement process [OY18] is given in Figure 1.1.

Note that, fault diagnosis is a logic-level process. It is an extension of the domain of logic testing. It relies on the fault models defined for logic testing and makes use of the ATPG test set. The details of this process are discussed in Section 1.1.1. Thus, a logical

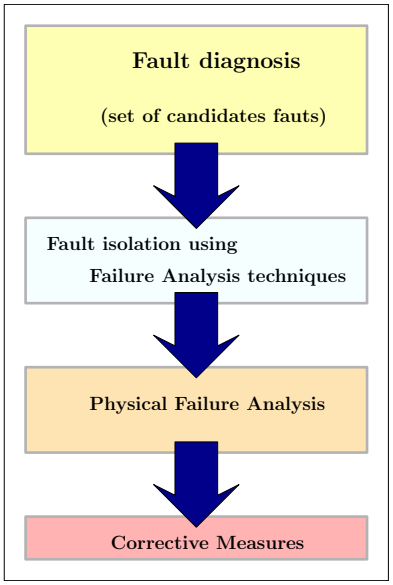


Figure 1.1: Flow of the yield improvement process.

testing framework provides solutions to advance technology by aiding the yield improvement process.

While logical testing solutions contribute to technological upgradations, new technologies are thrusting fresh challenges to logical testing. One crucial challenge is due to the increase in the number of unknown (X) sources. These values cannot be handled during simulation by an ATPG tool. They affect test coverage, diagnostic resolution, and compression quality significantly. X-values create several problem during testing:

1. It is hard to simulate the circuit in the presence of an X value. Generally, pessimistic simulation approach is followed. Hence such X-signals propagate in the circuit with much more effect than the actual.
2. The controllability of the lines which have X-values, is lost implying that the faults in these lines cannot be detected. Thus, it affects the detectability of the circuit, which might severely affect its fault coverage.
3. The presence of X-values severely affects the response data in the compressor/ decompresser (CODEC) environment. In a large circuit, the number of input/output ports and flip-flops are too huge, and hence, test application time/test data also increase significantly. To alleviate such problems, a scan chain is broken into multiple chains to decrease test time. The input patterns are supplied in a compressed form [LLEP07], and are decompressed by a decompresser before they are applied to the chip. Similarly, test responses are compressed at the output before they are stored. Such compressor/decompresser architecture is often adopted to reduce the test cost in VLSI circuits. The compressors are generally built using XOR gates, and the most

common type is based on the use of multiple input signature register (MISR). Since XOR-trees always allow X-values to propagate to their outputs, they may corrupt a large number of output bits. In fact, it may corrupt the entire response data in the case of MISRs, where XOR operations are performed with the data arriving in succeeding scan cycles. An MISR-based compressors requires that the response data is free from X-values [TGP17]. The CODEC architecture with an X-value in the first scan chain is shown in the Figure 1.2. To illustrate the effect of X to the scan chain and the response data, the corrupted data is marked with red color.

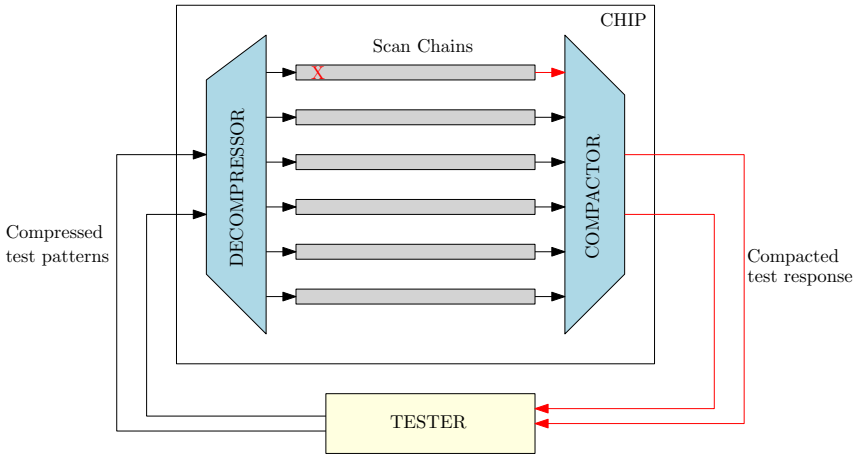


Figure 1.2: Test data compression.

4. In the above architecture, an X-value in a scan chain may corrupt all subsequent response bits when the test response is scanned out.
5. Fault diagnosis based on such corrupted response is hampered gravely.

Moreover, today's chip host analog and mixed signal components on a single chip called SoC (system-on-a-chip). All such factors have increased the number of potential sources for X-values. These signals affect fault coverage as well as test data compression. The various X-sources that might affect testing of chip are listed below [MGBK12, WWN08].

- Unmodelled blocks: These include analog or mixed signal components. They are modelled as X-values.
- Memory issues. In the case of scan-based ATPG, uninitialized RAM-cells may lead to X-values on reads.
- Specific flip-flops. These are unscannable flip-flops or those which do have set/reset. During power-on, these flip-flops are left uninitialized and hence their states are treated as unknown.
- Multiple clock domains. In a circuit with multiple asynchronous clocks, the signals from each domain may interfere causing metastability in the flip-flops fed by them, and cause an X-state [MGBK12].

- LBIST. Logical built-in-self-test (LBIST) [BS13] refers to an architecture built for logic testing of SoCs. They serve the role of the ATE (Automatic Test Equipment). LBIST generally, partitions the chip into blocks in test mode. While testing a particular block, the inputs from flip-flops of other partitions are treated as unknown. Thus, they act as X-sources during partition testing.
- Contention. When multiple signals attempt to drive a net, the signal in the net cannot be predicted and is considered unknown (X).
- Scan chains. As stated earlier, the X-value in a scan chain increases the number of X's per shift.
- Design bug. Various design bugs that could not be corrected during verification express themselves as unknown (X) during testing.

Thus, it is evident that the X-values may impact a circuit in multiple ways. We will look at the present solutions used for handling this issue in Section 1.1.2.

Technology scaling also give rise to increased defect complexity and circuit size. Recently, many ML-based solutions have been reported in the literature. Here, we will briefly look at certain attributes of ML; thereafter, in Section 1.1.3 we will discuss how ML-based tools are evolving to provide an efficient platform for solving problems in logic testing. The various aspects of ML that have led to huge popularity and wide acceptance in diverse fields including VLSI automation tools are as follows.

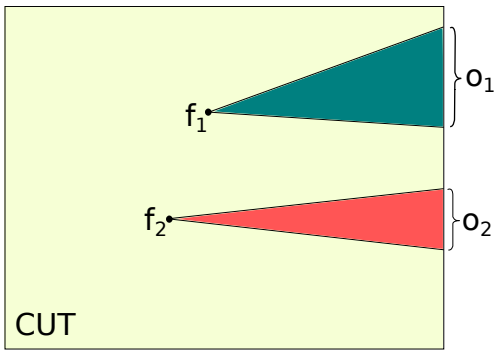
1. Learn from data. One commonality in various sectors, is the availability of profusion of data. As a result, data processing and analysis have gained momentum since recent past. ML has emerged as an integral part of data science, and provides techniques to extract useful information from the data. This may be in the form of some useful pattern in the data that could be helpful in deducing certain properties of the system or may help to build some models that the data represent, which can be used to gain knowledge from new data. Such models are used either for classification of data or prediction of an event.
2. Complexity of model learned. ML techniques can learn complex relationships in data and generate complex models. Thus they are more dynamic than statistical learning and hence are applicable to a wide range of data.
3. Fast. Once a model is learned, evaluating a new data involves some function evaluation. Thus, they provide quick inference. Moreover, with improved computation power of modern CPU/GPU, the learning (training) of data can be achieved with high computation speed.
4. Feature learning. One of the biggest challenges in ML is feature deduction. Formulating the features from the unprocessed data would take enormous time and domain expertise. This process would generally be done manually. However, with new techniques in ML, feature learning has been mostly automated. This gives a new impetus

to this field.

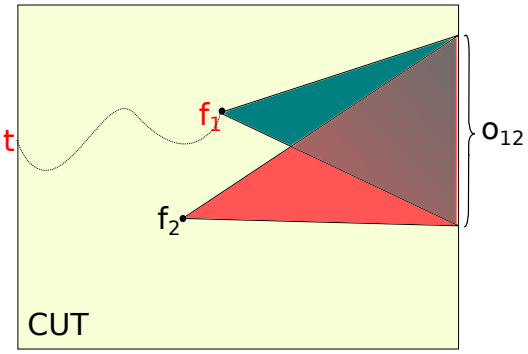
5. Application to unstructured data. In most of the applications handled so far, data are in the form of feature vectors or are structured as in digital images. Unstructured data like graphs and motifs cannot be expressed easily in the form of vectors. Graphical data exist in wide range of domains. In fact, the netlist of a circuit can also be thought of as a graph with a rich source of data. ML-tools towards this direction are fast maturing.

1.1.1 Fault Diagnosis: Utilizing Test Concept and Data for Diagnosis

As stated earlier, fault diagnosis of a failed chip provides a set of candidate-faults in the chip based on its failure response. We also saw that this is the first guiding step for yield improvement. Fault models used in logical testing, capture the effect of some defects in the chip. Test vectors are generated based on the circuit netlist, and each line in the netlist represents some wire in the chip. For ATPG algorithms, the primary aim is to produce a set of test vectors with high fault-coverage. That is, to ensure that the maximum number of faults in the circuit-under-test (CUT) can be detected. Such test sets also have some diagnostic power. A fault is said to be diagnosable if it can be distinguished from every other fault in the fault set F. A fault f_1 is said to be distinguishable from a fault f_2 by a test vector t if the corresponding output vector when f_1 present, is different is from the output vector when f_2 is present. There are many scenarios where this is possible for a test vector in a test set T. Two of the scenarios are discussed below and shown in Figure 1.3. In the first case (Figure 1.3a), the output cones of the faulty lines are independent and hence are the set of output ports (O_1 and O_2). Thus, if a test vector $t \in T$, detects one of them, it is also capable of distinguishing them. Even if t detects both faults, the faulty response when either of the two faults is present, will be different. Thus, in this case a test set will



(a) Independent output cones



(b) Intersecting output cones; only one (f_1) of the faults is detected by $t \in T$.

Figure 1.3: Instances where a fault pair is distinguishable by a test set T.

always distinguish the faults. In the second case, the output cones totally intersect (Figure 1.3b). There is a possibility that a test set may produce same response for the fault-pairs to all the test vectors in the test set. However, there are still some possibilities where in some cases, only one of the faults is detected by a test vector t; in such case they can be distinguished. Thus the failure response can be used to diagnose the faulty lines.

1.1.2 Present Solutions to the Unknown Value (X) Problem

Various techniques that have been adopted for handling the unknown value problem are discussed here. There are two ways in which the affect of X is alleviated. The first approach is to obstruct the value at its source. This method prevents an X-value from entering the chip. This may not be always possible and it depends on the X-source. All X-sources may not be accessible for blocking. The most detrimental effect of X is felt in the test compression environment. In the second approach, one aims to prevent the X-values, present in the output response captured in scan-cells (flip-flops), from entering the test compressor. The methods used are listed below.

1. X-bounding. In the BIST environment, extra hardware is added to a circuit to block the X-source. An X-bounding method for LBIST is shown in Figure 1.4 [BS13]. When the partition L2 of the circuit is under test, the flip-flop signals from the other two partitions are blocked by adding multiplexers. The X-signal is replaced by a known signal from a flip-flop in partition L2.
2. Resetting flip-flops. In certain at-speed BIST environments, the X-state of a flip-flop is replaced by resetting it to a known value by a reset signal [MS08].
3. X-masking. In this scheme, either the X-signal in the response is masked by a known value [TWE+06], or the scan chain itself is masked [SK17], so that the X-value does

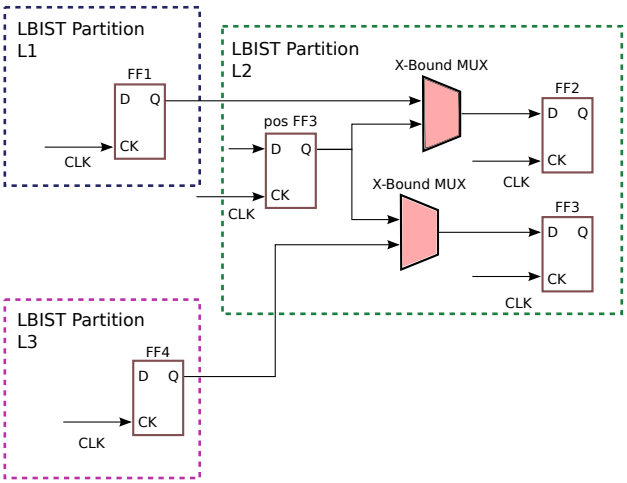


Figure 1.4: X-bounding in LBIST [BS13].

not enter the compressor.

1.1.3 Machine Learning in Circuit Testing

As the advancement of technology brings in new challenges, ML-techniques are gaining more popularity in the digital test framework. ML-based research and application in testing is rapidly growing [DED+17, DED19]. Figure 1.5 shows an abstract view the current state-of-the-art.

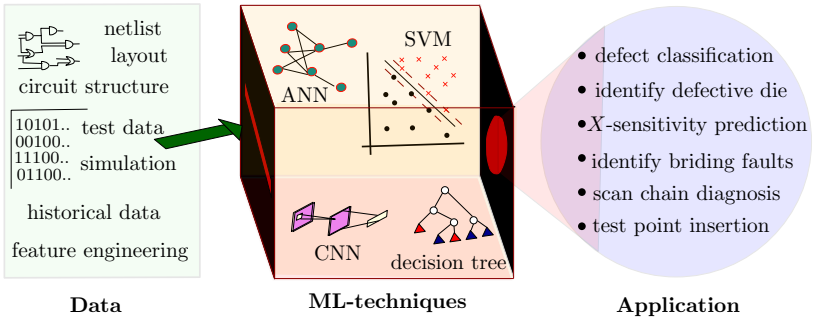


Figure 1.5: ML in logic testing.

Since the failure data obtained during IC-testing is typically of high volume and complex in nature, ML is likely to provide befitting solutions not only because of its ability to efficiently crunch data but also for its fast solution. Thus, this approach may reduce "time-to-market" of the product. Another potential data source is the repertoire of circuit netlists and layout geometry. Such information can be efficiently used for diagnosis and for guiding ATPG. Simulation tools offer an added advantage for ML applications since the required amount of data can be generated using them. The main challenge lies in creating standard data set and automating feature extraction. With such a large repository of data sources in the area of logic testing, there are many opportunities for solving these problems.

1.2 Summary of Contributions

We present an overview of our contributions in this section. We study three problems in this thesis. The first one aims at improving fault diagnosis in a combinational or scan-based sequential circuits. We propose a method for selecting diagnostic vectors using a combinatorial approach. The second work is concerned with an analysis of X-sensitivity in logic circuits. We have developed a predictor, that provides the detectability-loss information of the circuit due to an X-source. The technique is based on machine learning that utilizes structural features of the circuit. The third work presents a method for lossless structural encoding of a digital circuit which can serve as an input to machine learning tools in the future in order to obviate the need for feature engineering.

1.2.1 Selecting Diagnostic Vectors from Detection Test Sets for Logic Circuits: A Combinatorial Solution

Problem Description

The aim of this work is to generate a diagnostic test set (DTS), which is a set of test vectors that are capable of distinguishing as many fault pairs as possible in the CUT in addition to detecting them. Fault diagnosis localizes candidate faults based on the failure response of DTS. Computing the test vector (distinguishing vector) that distinguishes a pair of faults, is equivalent to test generation [ZA10], and so the former is a computationally hard problem like the latter. Also, some fault pairs are functionally equivalent, which means that no distinguishing vector exists for them. A DTS partitions the fault set in to a set of faults called test-equivalent sets. Each fault-pair in a partition is indistinguishable by the test vectors in the DTS. If the average partition size is big, it indicates a large set of candidates faults during fault diagnosis. So, the quality of DTS determine the efficiency of fault diagnosis. Although the modern ATPG tools can efficiently produce near-optimal test sets with high fault-coverage for a CUT, they do not explicitly target fault isolation. Thus, a lot of distinguishable fault pairs are not distinguished by a detection test set. As a result, fault diagnosis may produce a poor set of candidate faults, which may be of little help for failure analysis. So, the construction of DTS is an important and challenging task.

Solution Overview and Results

Various techniques of generating diagnostic test vectors have been proposed earlier. Most of them aim at adding additional vectors to the detection test set. Several of them are miter based; they target a fault pair to generate a distinguishing vector and does so by adding extra circuitry to the CUT. All of them depend on the analysis of the circuit. Similar to ATPG, techniques targeting diagnostic test generation (DTPG) have also been reported. In this work, we report a novel technique for selecting a powerful DTS for stuck-at faults. It does not use any diagnostic test generation, circuit modification, or analysis. Its sole requirement is the availability of ATPG and simulation tools. Table 1.1 highlights the scope of our technique against previous work. Overheads associated with these tasks are not required in our method. We do not explicitly aim at adding more vectors to a detection test set.

Table 1.1: A comparison of our work with previous work; (s): stuck-at, (t): transition

Work \ Technique	Circuit modify	Circuit analysis	SAT	Add tests	Use ATPG	DTPG	Fault model
[ZA10], [ZA11]	✓			✓	✓		s, t
[YZHL10]	✓		✓	✓	✓		s
[WLL14]	✓			✓	✓		s
[RSRB15]	✓		✓		✓		t
[PR07a]		✓		✓	✓	✓	s
[GMK91]				✓	✓	✓	s
[Pom19]		✓		✓	✓		s
proposed				✓	✓		s

Our method is based on the observation that the detection set of a circuit is not unique. This is because a fault can be detected by several test vectors at any one of the outputs where the fault can be propagated. Thus, an ATPG tool has many options to choose from. In fact, experiment with various tools [LH93, J-2] proves so. The proposed method works in two phases. In the first phase, an ATPG is invoked to create a pool of test vectors as follows. In each iteration, the new test set generated by ATPG is merged with the pool, if the test set distinguishes new fault-pairs. Once we obtain the pool, a set of test vectors (which form the DTS) is obtained which have the same diagnostic cover (measuring diagnostic efficiency) as the pool. Two variants of our algorithm are proposed for this technique. The first algorithm follows a greedy approach, where, once the pool is formed, in each iteration, a test vector is selected, which is essential and contributes to the maximum increase in diagnostic and detection coverage of the test vectors so far selected. The second algorithm is based on partial simulation and can be executed faster. Experimental results on several combinational and scan-based benchmark circuits demonstrate the effectiveness of our method in terms of the size of the DTS, diagnostic coverage, and CPU-time.

1.2.2 Prediction of X-Sensitivity of Circuit-Inputs on Test-Coverage

Problem Description

The X-sources of a circuit may influence its detectability to various extent. In this work, we study the relation of the underlying structural features of a circuit to the detectability-loss (`DT-loss`) when its inputs are set to X. The objective of this work is to extract a set of structural features of a circuit which are related to `DT-loss` experienced in the presence of X-inputs. The features also need to capture the structural variations (and the corresponding `DT-losses`) among different circuits. Our ultimate aim is to design a regressor that predicts

the `DT-loss` for the X-sources for any given circuit. The prediction can be thought of as functional evaluation, and a suitable regressor if designed, can be used for instant grading of input X-sources. Such prediction will be useful for treating the X-sources based on the `DT-loss` such that they can be suitably handled during ATPG or test application.

Solution Overview and Results

We carried out an extensive study of the structural feature of benchmark circuits [BBK89, CRS00] and corresponding values of DT-loss observed on setting their inputs to X-values. These circuits are diverse in their size, structure and functionality. We observe that a set of features that explain the `DT-loss` in a given circuit may not be as good in explaining the `DT-loss` observed in some other circuits. Also, the variance of `DT-loss` within each circuit is noted to be unique among the circuits. Since the circuits have different sizes, each feature needs to be suitably normalized. In order to study and formulate the features, we represent circuit netlist by a graph called circuit-graph, where the logic gates and input/output ports form the vertex set, and the connections form the edge set. The vertices are identified as either 'nodes' or 'non-nodes' vertices. Non-node vertices are those which do not block an X-signal, e.g., an inverter or buffer. Only the "nodes" are considered for feature computation. Special attributes for a node are assigned, such as level, depth and X-depth. They roughly represent the maximum logical delay from the circuit inputs, the proximity to the circuit inputs and from the X-source, respectively, where the distance (delay) measure is computed based on nodes only. Finally, we propose a set of twelve structural features as given in Table 1.2. In the table, $\mathcal{P}_1$ denotes the output cone of an X-source and $\mathcal{P}_2$ denotes the part of the circuit which influences $\mathcal{P}_1$.

Table 1.2: Summary of features

No.	Features
1	Number of nodes in $\mathcal{P}_1$.
2	Number of nodes in X-depth-1.
3	Average level of nodes in X-depth-1.
4	Maximum level among nodes in X-depth-1.
5	Number of output ports in $\mathcal{P}_1$.
6	Normalized sum of levels of output ports in $\mathcal{P}_1$.
7	Average X-depth-to-level ratio of output ports in $\mathcal{P}_1$.
8	Number of input ports feeding $\mathcal{P}_1$.
9	Average depth-to-X-depth ratio of nodes in $\mathcal{P}_1$.
10	Number of lines from $\mathcal{P}_2$ feeding the nodes in $\mathcal{P}_1$.
11	Maximum X-depth among the nodes in $\mathcal{P}_1$.
12	Categorical binary variable

We performed experiments on circuits in ISCAS'89 [BBK89] and ITC'99[CRS00] benchmark suits. The data points here, consist of circuit inputs, which are considered as X-sources. We have used support vector regressor (SVR) as a ML-tool with (Radial Basis Function) RBF kernel. Experimental results show a good prediction over test data and output fare grading of the X-sources. The predictor could clearly distinguish the X-sources that cause negligible `DT-loss`. Identifying such X-sources may find good applications while addressing the issue of X-bounding or initialization, cutting down on hardware overhead and test cost. Also, X-sources with high value of `DT-loss` can also be identified by the predictor. Such results are useful for improving the fault coverage of an ATPG tool, and this application is demonstrated in our work.

1.2.3 Encoding Large Graphs for Representation of Logic Networks

Problem Description

Although the graph structure of a circuit netlist provides a rich source of data, as discussed in Section 1.2.2, formulating structural features requires great amount of analysis and time. Moreover, the relevance of features may differ depending on the problem settings. Our objective is to devise a representation of a circuit-graph, based on Prüfer sequence, such that (i) the structure of the entire graph is encoded, and (ii) the encoding has linear-size with respect to the size of the graph so that it is scalable to large graphs. Such encoding can be conveniently used to feed the graph directly to ML-tools so that the features can be learned by the tool. We also study some new properties of Prüfer codes that could make them interpretable and suitable for ML-based applications.

Solution Overview

Recently, graphs have become the focus in the ML research community because of the ubiquitous presence of graphical data in many real world scenarios. Due to the inherent highly unstructured nature of graphs, it is difficult to devise a structured, vector-like representation for them that stores all the structure/connectivity information of the graph. Moreover, due to the large size of real-world graphs, including circuit-graphs, it is difficult to encode them in a compact form. Here, we present a graph encoding based on Prüfer sequence [Pru18]. This sequence was first used in 1918 to prove Cayley's formula, which was used to count the number of possible spanning trees in a graph with a given number of vertices. Prüfer sequences offer the following two advantages. Firstly, the size of the Prüfer sequence is of the order of the number of edges in the tree. We observe that the real world graphs, especially circuit graphs, though large, are very sparse. Thus, for a graph $G(V, E)$, $E = \mathcal{O}(V)$. So, employing Prüfer sequence to represent the graph would provide a compact representation. Secondly, Prüfer sequences the preserve structure of the entire

tree. However, classical Prüfer sequences are applicable only to trees. So, in this work, we present a method, called $\mathcal{GT}$-enhancement, to modify a graph to a tree, by augmenting the vertex set so as to break all the cycles in the graph while keeping the edge-count intact. We present two methods for $\mathcal{GT}$-enhancement. The first approach is based on partitioning the graph into trees. This method requires an additional list of labels of some vertices to preserve the structure of the graph. Experimental results on benchmark circuits show that this list is much smaller compared to the edge count of the graph. The second approach gives an improved solution where it is no longer required to store the extra list of labels, and hence, a single Prüfer sequence is enough to represent the graph. We call this approach *Seek-Edge-aNd-Split-On-Revisit* (SENSOR) $\mathcal{GT}$-enhancement. Further, we study the properties of Prüfer code in order to improve its interpretability. Since current ML tools only process vector data, we also discuss how it can be used for graph embedding, which provides a vectorial representation of graph.

1.3 Organization of the Thesis

- Chapter 2: This chapter describes the background and literature survey on diagnostic test generation, unknown-value problem, and various machine-learning approaches used in digital testing.
- Chapter 3: This chapter presents a method to obtain a diagnostic test set based on combinatorial properties of ATPG test sets.
- Chapter 4: In this chapter, we describe a regressor which predicts the detectability-loss due to an X-input in a circuit based on its structural features. We present a method for grading X-sensitive inputs based on machine learning.
- Chapter 5: In this chapter, we present a new technique for efficient encoding of large graphs representing logical circuits.
- Chapter 6: Finally, in this chapter, we summarize the content of this thesis and discuss possible future research directions in the area of testing, fault diagnosis and applications of machine learning to such problems.

CHAPTER

TWO

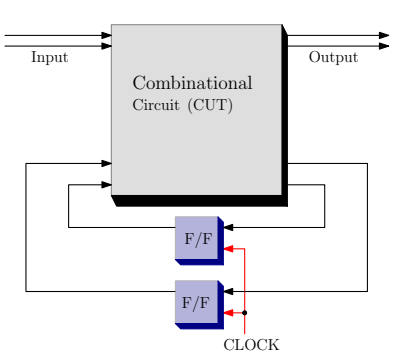
LITERATURE REVIEW

In this chapter we discuss the basics of digital logic test and diagnosis. We also present an overview of machine learning (ML) and its relevance to this field. A literature survey on diagnostic test pattern generation, solutions to handle unknown values, and machine learning techniques that are used in digital test is summarized here.

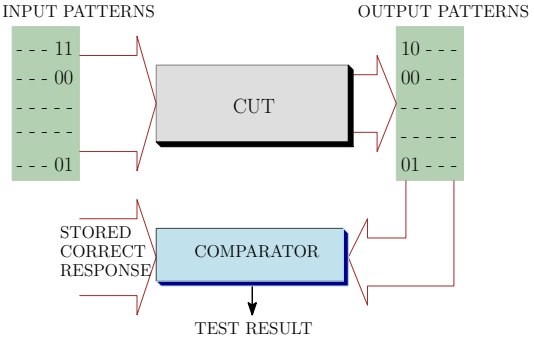
2.1 Overview of Digital Circuit Testing and Diagnosis

From an theoretical view, a digital circuit is made up of a combinational component consisting of logic gates, which are interconnected such that the signal flows in one direction from the input to the output of the circuit, and sequential components that consist of memory elements called flip-flops (F/F), which may feed the signal both in forward and reverse directions, and input/output ports. Unlike an analog circuit, signals in a digital circuit are discrete in nature and assume only two states denoted as '0' and '1'. A typical representation of a digital circuit is shown in Figure 2.1a [ABF02]. The F/Fs may not be directly observable or controllable, and hence they add more difficulty in testing the circuit-under-test (CUT). A popular design-for-testability (DFT) mechanism called *scan-chain*, is adopted to alleviate this situation, which allows a mechanism where the F/Fs are made directly controllable (observable) by a scan input (output). The F/F-outputs (inputs) are called pseudo-inputs (pseudo-outputs) of the circuit.

The basic principle of testing is briefly explained here. There are two aspects in testing: *test generation* and *test application* [BA05]. For a circuit with n inputs, since the input vector space is explosive (2^n), test generation, commonly called ATPG (automatic test patterns generation), involves devising algorithms to select a set of input vectors, called a *test set*, from the input vector space so that maximum possible modelled faults can be detected at observable outputs, on application of these vectors to the inputs. For this



(a) Canonical structure of a digital circuit [ABF02].



(b) Test application of input patterns generated by ATPG, on the chip in the ATE [BA05].

Figure 2.1: Digital circuit testing

purpose, the defects are modeled as logical *faults*. Two common fault models are the stuck-at fault model and the transition fault model. Once we have the test set, during test application, the chip is tested using ATE [Gro06], where input patterns are applied and its output patterns are collected. They are then compared to the expected (correct) circuit outputs to decide if they pass or fail the test. If a test fails, the output response is further analyzed for diagnosis. A simple flow of test application is shown in Figure 2.1b, although this does not tell the whole story.

While an ATPG targets to find tests for individual faults, diagnostic test pattern generation targets a pair of faults f_1 and f_2. Consider a combinational circuit which produces an output function $F_i(t)$ at its i^{th} output, for a test pattern t. Let us assume that in the presence of f_1 (f_2), the output function becomes $F_i(t)_{f_1}$ ($F_i(t)_{f_2}$). The test vector t, would distinguish the fault pair, if the following Boolean equation is satisfied [GMK91].

$$F_i(t)_{f_1} \oplus F_i(t)_{f_2} = 1 \tag{2.1}$$

Equation 2.1 implies that t should produce different output values in the presence of the two faults. This implies that only one of the fault should be detected by t. A test pattern which can detect both the fault can distinguish the fault pair only if they are detected at two different outputs. Since a circuit generally has several output ports there are many ways the fault pair can be distinguished.

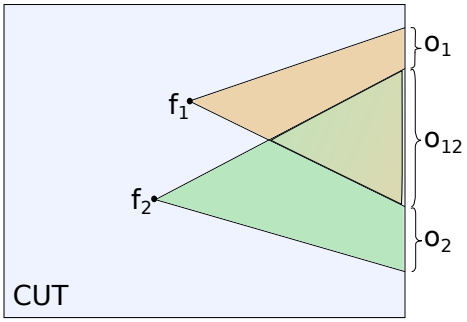


Figure 2.2: Intersecting output cones of two faults. O_1, O_{12} and O_2 are the set of output ports reachable from only f_1, both f_1 and f_2 and, only f_2, respectively.

In Figure 2.2, if the fault f_1 (f_2) is propagated to any one output in O_1 (O_2), by t, they are distinguishable by it. However, if both faults are propagated to O_{12}, then they should be propagated to different outputs so as to be distinguished by t. If no input vector distinguishes the fault pairs then they are called *functionally-equivalent fault pairs*. If the diagnostic/detection test vectors do not distinguish a fault pair, then it is called test-equivalent fault pair. Thus, there are two aspects of diagnostic test generation, identifying distinguishable fault-pairs along with the distinguishing vector, and identifying the functionally-equivalent fault pairs. Both help in improving the diagnostic test generation process.

An important aspect in test application is the test compression environment which is employed to reduced test cost. It was briefly introduced in Chapter 1, Section 1.1. One of the features of test compaction environment is the response compactor (Figure 2.3). As shown in the Figure 2.3, in each scan-cycle, the data from n scan chains is compacted into m bits where $m << n$. The compactor can either be a spacial or temporal compactor.

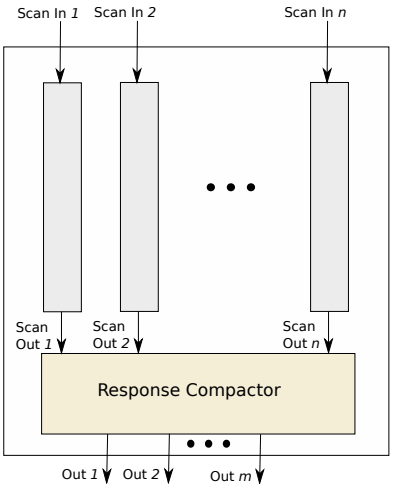


Figure 2.3: Multiple scan with response compactor [MK02].

In the case of a spacial compactor, the data per cycle is compressed. However, in a temporal compactor, which uses multiple input signature registers (MISR), the data from more than one scan cycle is considered for compression. In the case of a temporal compactor, the number of outputs is generally one. Such a compactor poses two drawbacks. First is the problem of aliasing which reduces test and fault coverage. The second and most detrimental drawback is the effect of unknown (X) since they can mask the error bits.

2.2 Overview of Machine Learning

ML, as the name suggests, is a field that aims at "learning" information and knowledge from data utilizing the computing power of the modern "machine", namely computers. While its general aim, that of inference from the data, is similar to that of statistical learning, it is more liberal in including the methodologies applied and the kind of data for which it is applicable. Its methodology extends beyond statistical learning, from applying geometry and computer science to just mimicking biological process of neural networks, which are yet to be fully understood. The range of data may vary from simple tabular, structured data to more complex structured data like images and videos, to unstructured data such as motifs and graphs. Depending on the nature of data, two basic approaches in ML are deployed. The first approach is to analyze the data for the presence of any pattern, which is called clustering, and falls under unsupervised learning. In the second approach, each data point has labels attached to it and the data can be used to approximate some function/model that the labels are assumed to represent so that any future unlabelled data can be given a suitable label. This comes under supervised learning. The common techniques used for unsupervised ML are Bayesian inference, k-means clustering, and spectral clustering. The popular techniques that are used for supervised learning are decision trees, support vector machines (SVMs), artificial neural networks (ANNs), Bayesian networks and random forests (RFs) [HTF09]. Supervised learning is more popular because of the availability of standard tools, especially SVMs and ANNs.

While supervised learning is often preferred over unsupervised, many a time, labels are not present or difficult to obtain. Hence, the method has to depend on the kind of the data available. In the area of digital logic testing, there are many opportunities for applying supervised learning. Various data sources that have been used or can be potentially used for ML applications in the field of digital electronic testing are listed below:

1. Manufacturing test response: The failure response patterns of a volume of defective chips obtained during production testing can be used for diagnosis[HKP04, XPLB13].
2. Simulation : A variety of labelled test data can be generated using simulation with CAD tools for testing. They can be performed for different fault models or defect types [GW16, CLH+19].

3. Historical data on diagnosis : These are labelled data, which consist of the faulty components responsible (root cause) which were diagnosed during earlier test cycles and their test failure response/syndromes. This technique is applied especially to board-level diagnosis [SJX+13, YZCG13, YCZG15].
4. Circuit parameters : Number of input/output port, details of scan-chain[LCP+17].
5. Circuit structure: The logical network structure, represented as a directed graph provide a rich source data [PBCB18, MRK+19]. The gate-level description of the netlist also provide functional and state information of the logic circuit.
6. Physical layout [NTB10, GW16]..

Oftentimes, we do not work with the data directly in order to obtain a meaningful model/inference. (i) The dimensions of the data many be too high, (ii) the data may be biased, (iii) the data may be noisy, (iv) they may be unstructured. Thus, extensive pre-processing may be required before we make them suitable for ML-based computation. One of the major steps in this direction is feature engineering, i.e., formulation of features from the available data. In some cases, feature selection is also needed. In the case of data such as graphs, a new method called representation learning [BCV13] is becoming popular.

Once we obtain a set of good data for learning (also called training data), we need to choose which learning technique to apply. Next, for supervised learning, in order to obtain a model which has good generalization capability such that it gives good results for yet unseen data (test data), the various hyper parameters should be carefully selected. An nice introduction to ML is presented in [LGEC17], while more insights into trends can be found in [JM15]. A number of guidelines on how to obtain a good model appear in [Dom12] and electronic design automation (EDA) and test specific discussions in [Wan17b].

2.3 Diagnostic Test Generation

One of the earliest works that aim at generating a diagnostic test is by Gruning et al. [GMK91]. Similar to ATPG algorithms, they proposed an algorithm for diagnostic test pattern generation (DTPG). Their approach is based on the observation that generating a detection test set is similar to generating a diagnostic test set. With respect to the j^{th} output port PO_j, to generate a test for a fault, the faulty line need to be sensitized and then propagated to PO_j. Similarly, in the case of generating a diagnostic test for a fault pair, either one of the faults is sensitized and propagated or both is sensitized and only one is propagated to PO_j. Moreover, like ATPG algorithms FAN [FS83] and SOCRATES [SA88], the set of lines which can be uniquely assigned can be similarly identified in each step of the search process by extensive analysis of circuit connectivity and function. Based on this observation, they proposed a diagnostic test generation algorithm based on branch-and-bound method. Instead of formulating a new test generation algorithm to analyze the circuit

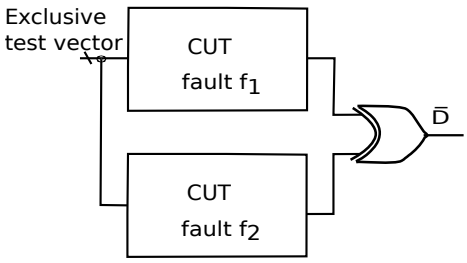


Figure 2.4: Exclusive test generation by running ATPG on circuit constructed by XOR-ing two copies of the CUT [ABKS03].

for generating the distinguishing vectors, a number of work [ABKS03, YZHL10, VCAA04] aim at reusing ATPG, without modifying it, by using a miter-based technique. Based on the observation given in Equation 2.1, a method to generate the distinguishing vector is proposed in [ABKS03]. Such distinguishing test vectors are referred to as exclusive test vectors since only one of the faults is exclusively detected. Two copies of a single output circuit, with their output XORed are taken, as shown in Figure 2.4. The first copy of the circuit is injected with fault f_1 and the second copy, by fault f_2. The pair of faults are modeled by a single fault [YAS02] such that a detection test vector of this fault is the distinguishing test vector of the fault pair. Such duplication of the circuit required the ATPG to run on a circuit of 2X-size, and hence, increases the overhead. By incorporating two multiplexers (corresponding to the fault pair) with a common selection line S, to the circuit, the same technique can be applied without the need for duplication [VCAA04]. The ATPG for fault, S-stuck-at-0 would give the distinguishing vector. On the other hand, if the ATPG marked the fault as redundant, then the two faults are equivalent. This method is simplified and F applied to generate exclusive tests in [ZA10]. They also proposed a method of fault dropping during diagnostic simulation. While generating exclusive tests, the faults which have been uniquely distinguished from the rest of the faults, are dropped and not considered for further simulation. Such circuit-modification based methods target one fault pair at a time. Hence, each time the circuit is to be modified and an ATPG needs to be run for each pair. These two aspects are improved in the method presented in [WLL14]. All fault-pairs that remain indistinguishable by the detection test set, are targeted concurrently. Thus, the ATPG needs to be run only once. While the above methods dealt with adding diagnostic tests to distinguished the faults which were not distinguishable by the test set, authors in [YZHL10] proposed diagnostic test generation that gives a test set with high diagnostic power. This also follows a miter-based method, which uses ATPG to generate the distinguishing vector. Their approach is based on the observation that those fault-pairs which belong a fan-out free region are harder to distinguish than the faults being in different fan-out free regions. Thus, the fault-pairs belonging to same fan-out free region are targeted first. The work proposed in [RSRB15] similarly aims to improve the diagnostic power of a

test. They further put constraint on the size of the test set to be not more than the size of detection test set. They used a SAT-based approach. For a detection set T of size N, the method works in N iterations, where in each iteration a test vector t in T is taken and a new vector is added to the improved test set T' with same detection power as t but with improved diagnostic power. A diagnostic test set may not be able to distinguish all the fault pairs. Such indistinguishable fault pairs are referred to as diagnostic holes in [Pom19]. Based on the observation that though a pair of single stuck-at fault may not be distinguishable, a pair of faults in these lines based on different fault model may be distinguishable. So, in this work, for those stuck-at fault pairs, which are not distinguishable, the corresponding distinguishable test for the pairs of bridging-faults are added to the test set. Since the number of indistinguishable fault pairs can be very large, a method to reduce the target fault pairs is proposed in [PR07a]. It is based on the observation that only targeting a subset of fault pairs is enough to distinguish all fault pairs. Moreover, based on the observation that faults located in close proximity are hard to distinguish, it only targets a subset of fault pairs which are in close neighborhood.

Identifying the equivalent faults helps in diagnostic pattern generation by avoiding the time wasted in attempting to find distinguishing vectors for them. Several techniques to identify equivalent faults are proposed in [AFPB03]. These include evaluating the implication of the faulty values on a common (denominator) gate and function evaluation in this gate to check for their equivalence. Another method to assist diagnostic test generation is by identifying those fault-pairs which are guaranteed to be distinguished by the detection test set [PVRS04]. It is based on the structural property of each fault, like the analysis of output ports in the output cones and also simulation properties which define the output where the fault is propagated by a test vector. A third method of reducing the number of fault pair to be processed by a diagnostic test generation process is fault-pair collapsing [PR07b]. Here the dominance/equivalence relation defined for faults is extended to fault pairs such that a fault pair which is dominated by or equivalent to anther fault may not be exclusively targeted. So, in addition to structural properties, the fault dominance/equivalence properties are used here. For instance, in the case of three faults f_1, f_2 and f_3 such that f_3 dominates f_2, and f_2 dominates f_1, the fault pair (f_3,f_1) dominates (f_2,f_1).

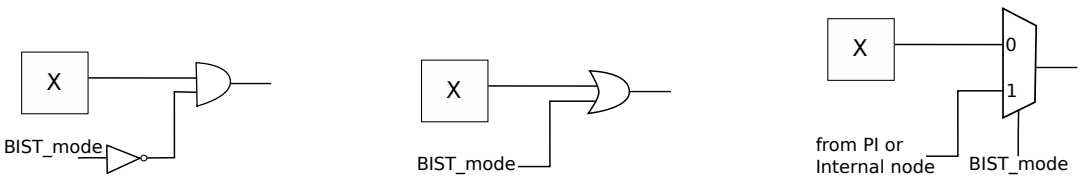
2.4 Unknown Value in Digital Circuits

In order to mitigate the effect of unknown (X) values in the circuit, three main approaches have been proposed. The first approach is aimed at improving ATPG and fault simulation in the presence of X. The second one is preventing X from entering the CUT before the test is applied. The third kind of solutions alleviate the effect of X in a test compression based environment.

Both logic and fault simulation in the presence of an X-value is computationally hard

[CA87, EKR+15]. The n-valued logic has been used to represent X-values in test generation [Rot66, Mut76]. However, such symbols cannot denote all the states arising due to correlation of X-values during their reconvergence and this leads to pessimistic simulation. An accurate logic simulation in presence of X's is given in [CP89] which uses a three-valued logic. For improving test pattern generation in the presence of X's, various methods have been proposed [EKR+15, SEB16, EKS+13]. Quantified boolean formulas (QBF) can be applied for accurate computation of signal values [EKR+15]. This allows the representation of the X-values using universal quantification. A fault which may not be detectable (propagatable to output), due to an X in a cycle, may become detectable in multiple cycles [EKS+13]. Thus the fault coverage can be improved in presence of X-sources in environments like partial scan. A SAT-based ATPG for accurate detection of faults in the presence of unknown is proposed in [SEB16]. A number of work also target [EKS+14, KSR04, HKWB12] fault simulation. A SAT-based method to compute the test coverage of a test set in the presence of X is given in [HKWB12, EKS+14]. The process is executed in two steps. Firstly, heuristics are applied to compute the signal whenever possible. For the rest of the lines, exact computation of signal is computed by using SAT formulation. A method for logic and fault simulation using indirect implication by applying three-valued logic simulation is proposed in [KSR04].

The X-blocking/X-bounding technique is used to mask the effects of X-sources by adding extra hardware to the circuit or the BIST-core [WWW06]. Some of the methods are shown in Figure 2.5 [WWW06]. An X-bounding method for LBIST is discussed in



(a) 0-control point; (b) 1-control point; (c) bypass logic

Figure 2.5: Some examples of X-blocking [WWW06].

[XST+01]. Since addition of hardware for X-bounding would affect the timing of the circuit, a method to mitigate this is proposed in [XST+01]. Another method of preventing X-values, where the X-sources are memory elements, is by initializing them to fixed values [MS08]. A method to reset the memory element during at-speed testing such that all the flip-flops are set to a unknown value without any timing disparity is discussed in [MS08].

Since all the X-sources may not be available for blocking at source, they enter the circuit. They damage the test response especially if they are compressed. So, various techniques are employed for alleviating the effect of X's in different test response compaction environments. The spacial compacter proposed in [MK02] called the X-compaction, consists of a network of XOR-gates. The network is designed in such a way so that the X-values can

be handled to some extent. They guarantee that the error in one or two scan chains with one unknown value in another scan chain in the same scan cycle does not mask the error. An improved form of compactors which are called convolutional compactors [RTWR03] employ a hybrid technique of both XOR-network and memory elements, and perform much better than spacial ones. With advanced architectures [RTWR03, RT05, AFI06], convolutional compactor can mask more than one X-value in the same scan cycle. A method based on scan-chain switching is employed in [WWW+10] to handle unknowns before they enter the space compactor. The scan chain that carry error bits that could be masked by X is assigned to different scan compactor cone such that the scan chain containing the X-value does not affect it. A novel method of output compaction based on output-bit selection is proposed in [LLH11]. Here a small subset of bits are selected using an extra hardware such the fault coverage can be maintained. So, the X-values in the scan chain implicitly do not affect the output. A counter-based implementation of the response-bit selector is proposed in [LLH+13]. Although an MISR is the most efficient compactor, the response signatures become completely invalid in presence of X's. A method for X-canceling in MISR compactors is proposed in [YT12]. Using symbolic simulation and Gaussian elimination, the X-values captured in the MISR is reduced.

The above methods belong to the class of X-tolerant compactor since they do not aim at masking the effect of X. The other class of compactors follow the X-masking approach [WSRW09, CKSF05, NPRK03]. A masking logic along with its synthesis technique in BIST environment is proposed in [PKR02]. An X-logic that are also allowed to mask some known bits reduces the area cost of the making logic [TWE+06]. Masking of few known bits do not affect fault coverage since a fault is generally detected by many patterns in the test set. While it is generally not taken into account, a circuit may have a significant number of faults which produce X-values in the response even though the fault-free response does not contain any X [Pom14b]. Also such X may be produced in significant proportion [Pom14b]. Taking this fact into consideration, a masking technique is proposed to mask such X values [Pom14b].

Along with the above methods, the effect of X's can further be reduced by assigning all the scan cells, capturing X's with high probability, to special scan chains. Such assignment is possible due to observation that the distribution of X in the response among the scan cells is not random. Based on the circuit structure and test set, a subset of scan cells can be identified which have high probability of capturing the X-values [WYL18]. Thus the compactor can be suitably optimized to handle such scan chains. Similar observation is also made in [WWN08], where such scan cells are identified by using various parameters and random pattern simulation. Special compactors have also been proposed such that each scan chain can be uniquely observed.

2.5 Machine Learning in VLSI Testing

In this section, we walk through the different problem settings in the context of digital logic testing where ML-techniques have been applied and to focus on the various features and modeling approaches that have been used. Most of the problems in this area are novel and there are ample scopes for improving the effectiveness of the solutions. What is important to note is that ML provides automated tools to handle many hard test problems, which otherwise would not have been easy to tackle. A majority of ML based digital test applications belong to the area of diagnosis. In the industrial domain, there have been some efforts to apply ML to circuit testing [LAP19]. A major bottleneck in this area is the unavailability of useful data. Additionally, unlike other fields where ML tools have been thoroughly investigated and successfully applied, their modeling in the area of digital testing is quite new and only a few studies have been reported so far in the literature. In the rest of the section, firstly, a note on ML in the field of analog circuit testing is briefly presented. We discuss applications of ML to diagnosis in Section 2.5.2. ML-based test compression is discussed in Section 2.5.3. In Section 2.5.4, studies related to circuit testability, are reviewed. Applications to timing analysis are discussed in Section 2.5.5. Finally, the possible future directions and challenges of circuit testing in the perspective of ML are highlighted in Section 2.5.6.

2.5.1 Analog Circuit Testing

An electronic chip consists of both analog and digital components. However, the two vary widely in their working principle and complexity and hence their testing. Analog operations are much more complex compared to digital operations and testing of analog circuits is much more challenging [Hat17, Mil98]. Unlike digital circuits, the signals in analog circuits are not discrete in nature, and thus, it is not easy to consider a suitable fault model that captures all error patterns while testing them. Furthermore, they exhibit non-linear behaviour and their outputs are very sensitive to various circuit and environment parameters. Most of the test techniques used for analog circuits are parameter-based, and it is difficult to design a deterministic test method. Various statistical and ML approaches have been explored in this area [Wan17b, Str18, BND16]. The good side is that analog circuits are small in size and they constitute only around 10% of the chip [KKH11].

On the other hand, testing of digital circuits has been extensively studied over several decades and is now well understood. The components used in a digital circuit are much simpler and fault models therein are well defined. Automated tools for test generation, fault simulation, and DfT insertion are available, which provide further options for designing efficient test strategies. Nonetheless, various technological advancements have added newer challenges to digital logic testing [KMG04]. A number of ML based approaches have recently been developed in this evolving area. We will review some of them and discuss future

challenges in this direction.

2.5.2 Diagnosis

Rapidly scaling technology demands the manufacturing process to be intricate and precise. Since manufacturing processes only improves over time, there is a low yield (fraction of good chips) in the initial phase. Yield learning [Ait12] has thus become a crucial step for the yield ramp up during volume production. Yield learning involves understanding the failures, locating the defects, and thereafter applying corrective measures to improve the manufacturing process. Figure 2.6 shows the various stages in yield learning [HSEL02].

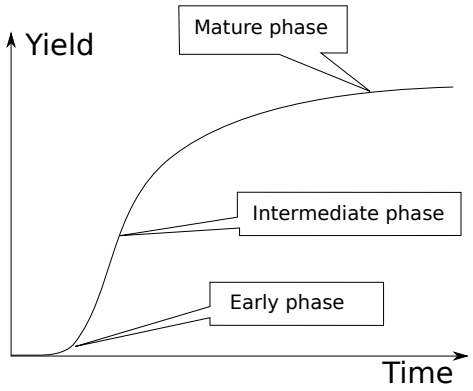


Figure 2.6: Yield learning phases [HSEL02].

The method of defect location utilizes a feedback mechanism called diagnosis. As the IC technology scales down and the level of integration scales up, the number and variety of defects inevitably escalates. Understanding these defects is an integral step in improving the manufacturing process. Traditionally, defects are located using a physical-level process called Physical Failure Analysis (PFA) [LHCL13, ZGC+18]. However, the intricacy and multitude of defects has not only made this process challenging but also very time consuming and costly. In order to guide PFA, a common technique is to deploy a logic-level process called *fault diagnosis*. Given the failure response and the circuit netlist, fault-diagnosis produces a set of candidate faults (root cause) responsible for the failed response. Fault diagnosis follows either a cause-effect (dictionary based) or an effect-cause (inject and evaluate based) approach. There are also techniques to analyze the response for useful inference before fault diagnosis (pre-processing) and after fault diagnosis to improve PFA (post-processing). One drawback of these methods is that they are inefficient for diagnosis of a variety of unmodelled defects, which are common in the sub-micron technology. Recently, a new method called volume diagnosis has been studied and also integrated in the diagnosis process, to guide PFA. The failure response of a volume of defective chips is analyzed to understand the defects and their root cause [TMR+07a]. Hence, volume diagnosis may be a suitable arena for exploring ML applications.

Note that both combinational and memory components may be affected by defects. Thus, while conducting manufacturing test, the flip-flops are tested first and diagnosed for defects, if any, through a process called scan-chain diagnosis [HGCL08]. Thereafter, the faults in the rest of the circuit, are diagnosed. Diagnosis is carried out hierarchically. In this section, we look at various ML-approaches that have been used for diagnosis at different levels of circuit hierarchy: wafer-level, scan-chain level, chip-level (fault diagnosis, pre- and post-diagnosis, and volume diagnosis), and board-level.

Wafer-Level Diagnosis

As mentioned earlier, the intricacies involved in the modern manufacturing process have made it challenging to maintain good yield during the fabrication of IC-chips. Also, the production of silicon wafers requires a long time-cycle [SNW17]. It is therefore important to identify the defects in the wafer early in the manufacturing process so that the process can be improved to reduce time and yield loss. It has been observed that the defects generally occur in clusters in a wafer in certain locations [OSK+10]. A kernel-based method to detect such clusters is proposed in [SNW17].

Die inking is a process of marking those dies which have latent defects. Burn-in tests are also generally applied for detecting such latent defects. However, burn-in tests are difficult because of the cost and the complexity involved [XSRM17]. A method to automate this process using ML has been reported in [XSRM17]. Faulty dies, which are close to defective clusters in a wafer, are inked manually while training the model. During the test phase, a binary classifier is used to decide whether or not a die is defective. An SVM with a radial basis function (RBF) kernel is used, which is one of the most common kernels used in the literature that uses Gaussian function for distance computation. Morphological operations consisting of erosion/dilation are used to remove noise. A feature vector based on the distance of the die from defective clusters is used during classification. The corresponding flow diagram is shown in Figure 2.7.

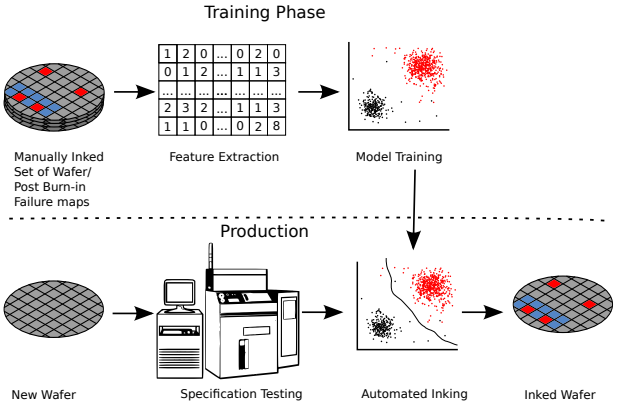


Figure 2.7: Automated die-inking [XSRM17].

One of the main reasons behind chip failure is process variation. There are numerous process parameters which contribute to such defect distributions. The values of these parameters may vary from wafer to wafer, and also within the same wafer. A major objective in testing is to select appropriate parameters that take care of most of the failures. in [TSS+14], a canonical correlation analysis (CCA) is used to determine the correlation by noting measurements over several samples. It also discusses the correlation of parameters corresponding to two different locations (inner and outer) of a wafer.

Scan-Chain Diagnosis

Scan-chain diagnosis aims at locating the faulty scan cells. The presence of a defective scan cell might affect a large number of response bits. Some defects, called permanent faults, are known to be easy-to-model since they always produce a fixed failure response pattern for a given test vector. These faults can be diagnosed by heuristic based techniques. The remaining scan-chain defects are due to intermittent or hard-to-diagnose faults since they do not produce a fixed failure response pattern during the production test. A method based on Bayesian learning [Tip04] is proposed in [HBK+17] to identify the faulty scan cells in the presence of such defects. Given a test set and the failure log for a faulty scan-chain, for each scan cell i, two kinds of bit-count are considered: $sbit_i$ is the count of patterns for which the cell i is supposed to capture a failure bit (sensitive bit) differing from the fault-free response bit; $fbits_i$ is the count of bits out of $sbit_i$ which actually fail in the production test. For a given faulty chain, the probability that a scan cell i captures $fbits_i$ faulty bits is computed by assuming a binomial distribution. Diagnosis is then facilitated using the Bayes formula.

In the above-mentioned technique, Bayesian learning is used for clustering which is an unsupervised learning technique. It relies on the unknown priors which may not be always readily available. The problem of scan-chain diagnosis for intermittent faults is modelled following a supervised learning method [CLH+19] using another ML-tool based on artificial neural network (ANN). An ANN mimics the working principles of a biological neural network. Its architecture consists of several layers of nodes called perceptrons, where each node computes a linear combination of the inputs feeding it from the preceding layer. A non-linear component is added by incorporating a non-linear activation function at each node. The weights attached to the edges incident on each node are updated and learned during the training phase by a method called back-propagation. [CLH+19] employs a multi-stage ANN for diagnosis of faults following a coarse-to-fine approach. Each data point, called the modelled faults, represents a fault type, the faulty cell and fault intermittency (probability that a fault is activated by the test patterns). In the first stage, called Coarse-Global Neural Network (CGNN), the binary response vectors for the test set are reduced to a single vector called integer failure vector (IFV). It is computed by performing bitwise addition of the binary response vectors. For illustrations, the figure from [CLH+19] is

redrawn in Figure 2.8. The input to the ANN is an IFV whose length is determined the

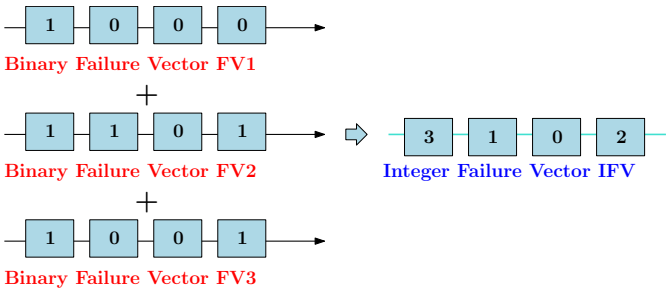


Figure 2.8: Compressing binary failure vectors into an "integer failure vector" [CLH+19].

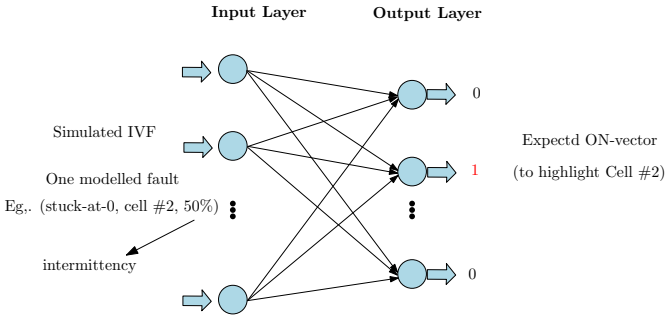


Figure 2.9: An example of CGNN training-vector [CLH+19].

number of scan-cells, and each node in the output layer represents a scan-cell of a particular scan-chain for which the ANN is being trained. Such an ANN gives a candidate faulty scan-cell as output called the center cell. An example of CGNN training vector given in Figure 2.9. In the subsequent stages, called RLNN (Refined Local Neural Networks), an affine group is computed over those scan-cells whose IFV is close to that of the center cell in terms of Euclidean distance. The ANN is built for each scan-cell based on its affine group. In these stages, instead of compressing the binary response vectors to a single IFV, they are concatenated sequentially to form a single vector. Its length can be reduced by removing the bits at certain positions based on the affine group and the new vector is called reduced cascaded vector (RCV). So, the ANN in these stages is also a two-layer network where the number of nodes of the first layer is equal to the length of RCV, and the number of nodes in the output layer is equal to the number of cells in the affine group. It is shown that with this supervised technique, the diagnostic accuracy could be increased by 20%.

Fault Diagnosis: Pre-Processing

The failure log of defective chips can provide useful information that could guide the fault diagnosis process. The entire failure data may not be available for diagnosis because data collection is expensive and time consuming [WPY+12]. Since only a small fraction of the

total response data is available, it usually leads to poor diagnosis. To alleviate this problem, a method is proposed in [WPY+12] to determine the minimal number of test responses that are required for proper diagnosis. The method utilizes a binary classifier that decides when the response collection process should be stopped. When a test response is collected, the classifier decides whether or not to continue to the next response. The features are based on the output response of the chip up to the application of the last test pattern. They have reported results for various classifiers such as kNN, SVM, and decision tree.

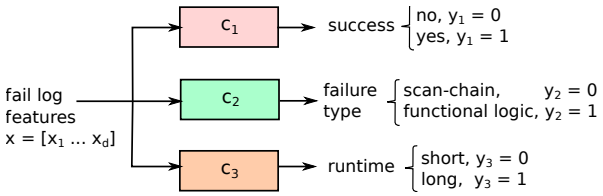


Figure 2.10: Three-output classifiers where X is a feature vector with d elements, and y_1, y_2 and y_3 are discrete variables denoting the classes [HFMB18].

The work in [HFMB18] introduces a classifier to predict the following: (i) whether the failure log is at all useful for diagnosis, (ii) the location of defects: scan-chain or functional logic, and (iii) the time needed for diagnosis. They have presented a set of features based on the failure log and used random forest to design the classifier. This is illustrated in Figure 2.10.

Fault Diagnosis: Post-Processing

Although fault diagnosis plays a major role in guiding the process of PFA, it is conducted at the abstract level. Moreover, the number of candidate faults reported (diagnostic resolution) is generally large. Many methods have been proposed to fine-tune the results of fault diagnosis based on ML techniques. They are usually concerned with two objectives: (i) defect identification, that is, mapping the diagnosed fault to a defect. This is challenging especially when it is based only on the failure response of the circuit [NTB10, GW16, GCI+17]; (ii) improving diagnostic resolution, where the candidate faults are analyzed so as to further prune the set in order to improve the diagnostic resolution [XPLB13]. The features used in both approaches are derived from the layout and logical information of the circuit and the output response of the failing chip.

Defect Identification: The problem of identifying bridging defects in a failing circuit has been addressed in [NTB10]. Such information is helpful for estimating defect density and size distribution (DDSD), which is required in yield learning [NZD+06]. Since a bridging defect represents a short between two signal lines, for each candidate fault involving line A, a set of bridging faults is considered involving its neighboring lines (B, X, Y) which are $\{(A, B), (A, X), (A, Y)\}$. The logical information of the circuit is expressed as Boolean

features. For example, the feature called "feedback" checks whether there is a structural path between the pair of lines of a bridging fault. Under the fault, such a path might create a latch or induce an oscillating behavior affecting the test result. Hence, these sites could be disregarded as a possible candidate bridging fault. Similarly, other Boolean features are used for checking whether the lines drive a parity gate or same gates or have logical correlation. The test-dependent features are formulated by analyzing the correlation between the tester output and the simulated response of the circuit in the presence of the candidate defect under various bridging fault models [WWW06]. These faults are processed by a rule-based classifier followed by decision-tree based classification. The faults that possess high scores of both logical and test-based features are classified as bridge faults. Similarly, non-bridge fault are identified by rule-based classification and the remaining faults are then classified by a decision tree [RM14]. The flow of the classification scheme [NTB10] is shown in Figure 2.11. The training set is created by using SPICE simulation as well as from the results of defect diagnosis obtained by PFA.

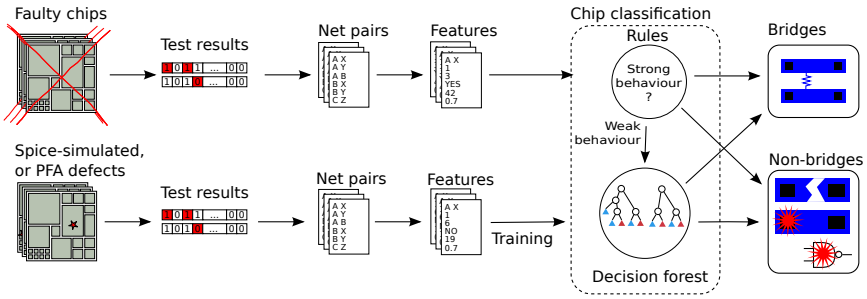


Figure 2.11: Flow for the classification of bridging defects [NTB10].

A neural-network based defect classifier for various faults is proposed in [GW16]. This can be used to classify defects at the early stages of volume diagnosis without using any special diagnostic test patterns. The network provides a warning signal as soon as the frequency of certain defects crosses a threshold. A candidate fault may be classified as various defect types, e. g., crosstalk induced delay, dominant-and(-or) bridge, Byzantine bridge, slow-to-rise or slow-to-fall bridge. Thirteen different features are computed based on the simulation of failing patterns in the presence of the faulty candidate. The features are expressed as the proportion of the number of failing patterns exhibiting certain properties. For example, one feature is defined as the proportion of failing patterns which set logic value 0 at the victim line. This is used for identifying dominant-or-bridging. Furthermore, layout information is used to find the neighborhood geometry, which helps to define a feature that characterizes cross-talk induced faults. The scheme for this ANN-based classifier is shown in Figure 2.12. Two sets of data are used in the experiments. First, the simulated data for a circuit is used to classify the faults in the same circuit. Second, the simulated data for a group of circuits is used to classify faults in a new circuit. The latter experimental

setup seems to be more practical value. This is because, generating the training data by simulation for every new circuit is time consuming and may not be feasible. However, this would require rich training datasets, which are still a challenge to collect.

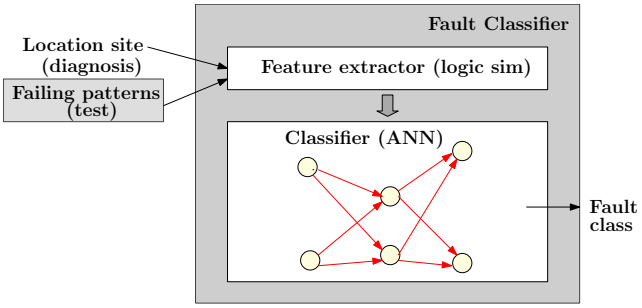


Figure 2.12: Defect classifier [GW16].

Another set of faults which are targeted for classification consists of transient and intermittent faults [GCI+17]. Both types produce similar test results and it is difficult to distinguish them. While intermittent faults lead to the degradation of the chip, transient faults contribute to unnecessary yield loss [GCI+17]. A tool based on Bayesian classifier is proposed in [GCI+17] to distinguish these two types of faults.

Improving Diagnostic Resolution: As discussed in Section 2.5.2, the second approach towards fault diagnosis aims at processing the candidate faults to improve the diagnostic resolution. Most of the diagnostic tools produce a larger number of candidate faults compared to the actual number of faults [XPLB13]. The efficiency of the ensuing steps of defect identification and PFA are thus impeded. In order to improve the diagnostic resolution, we need to pare down the set of candidate faults. A classification-based method similar to that in [NTB10] is proposed in [XPLB13] to label each candidate fault as either good or bad. A set of features is identified and thereafter the classification is performed in two steps: (i) the first one is rule-based, which identifies some bad candidates; (ii) the rest of the faults are analyzed by an SVM-based classifier. For the purpose of training, the labelled data available from PFA were not found to be so useful because of the fact that they are sparse and furthermore, being some kind of "old data", they may introduce error when applied to new circuits. In order to alleviate these problems, the authors in [XPLB13] proposed a method to generate labelled data for each circuit: the defective chips, for which fault diagnosis yields a single candidate fault, are used to mark "good" candidate class. The chips for which the fault diagnosis provides a large number of candidate faults (greater than a threshold value), are used to mark "bad" candidate class. It is, however, observed that the number of faults in the "bad" class is generally much larger than those in the "good class". As a result, the problem of managing unbalanced classes arises, which is handled by oversampling the faults from the "good" class.

Volume Diagnosis

Although today's diagnostic tools for handling faults in a digital circuit can achieve high accuracy, they still suffer from several drawbacks. To mention a few, they are often unable to distinguish functionally equivalent faults; they do not take into account the entire layout information on which the likelihood of defects is highly dependent. Feature-based systematic defects now impact deep sub-micron technology significantly. Available tools are incapable of diagnosing them. Even the yield learning methods such as PFA may not be suitable for handling them. In order to detect such defects, drawing inference from fail-logs of a large number of chips, has become imperative. This process is called volume diagnosis. Since it involves analysis of huge amount of data, the method needs to be time-efficient.

One of the crucial needs of the yield-learning process is the ability to identify systematic defects in the chips and to distinguish them from random defects [Mut14, HKP04]. A signature for each defective chip is created based on its failure response. Based on these signatures, the chips are clustered using the furthest-neighbor method [DG84]. Such clustering would help to analyze whether the chips in a cluster are failing due to a similar defect. Thus, it can be used to determine whether or not the defect is systematic. Another classification-based method for volume diagnosis is proposed in [WW09]. This classifier goes further and detects the location of the defect in terms of the fan-out free region. This is based on the observation that faults in a fan-out free region affect the same set of outputs. The CUT is decomposed into fan-out free regions, and each region is considered as a defect class. Defect classification is performed based on the failure outputs using an SVM. When a large number of chip failures occur due to a particular class, the presence of a system defect is inferred.

Volume diagnosis generally reports multiple failure features for each chip. A statistical-learning based approach is presented [TMR+07a] to estimate the failure feature probabilities. Another method based on Bayesian network is described in [CTR17].

A method to assist PFA by narrowing down the possible set of defects is discussed in [SBP+17]. A defect can have various signatures called "defective modes". During volume diagnosis, χ^2 independence test is applied to check whether the defects and the "defective modes" are related. Using the data obtained from layout-aware scan diagnosis, and test results, the values of χ^2 test are found. The p-values for the "defective modes" are used to rank them.

Board-Level Diagnosis

The technology of printed circuits boards has made it possible to integrate diverse components like application-specific integrated circuits (ASIC), memory, and I/O in a single board by using printed interconnections, and consequently testing and diagnosis is needed at the board level as well. It has been experienced that though the individual components pass the

manufacturing test in the ATE, they fail the board-level functional test. This is primarily due to difference in the real testing environment from that of ATE and the components are marked as no trouble found (NTF). This is a dreaded problem in industry which needs careful management to ensure reliability of digital systems and for their regular maintenance. Board-level functional fault diagnosis follows a reasoning based approach. The knowledge regarding the root cause of failure-syndromes for an initial set of boards which, could be repaired, is used as training data to predict defective components for new boards. The syndromes are gathered from the failure information of the components under a test set. These syndromes lead to a set of features and the underlying root-cause instances that are diagnosed serve as labels in the training set. A number of approaches based on various ML techniques such as ANN [OME05, ZCW+11], SVM [ZGX+12, YZCG14], and decision trees have been proposed in this direction [SJX+13, YZCG13, YCZG15]. In the ANN based approach [ZCW+11], the inputs are fed with different syndromes and the outputs denote the components. In order to handle large-size board-level diagnosis problems, [ZCW+11] applies a group of two-layer, single-output ANNs (Figure 2.13), where the output node represent a component and classify it as whether it is the root cause of failure or not.

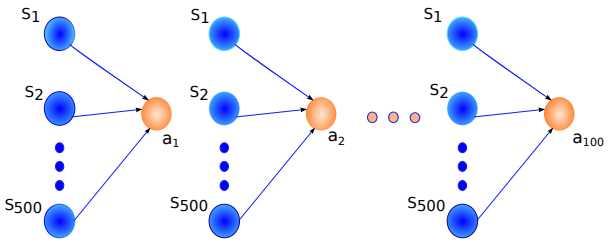


Figure 2.13: An illustration of the ANN architecture used in [ZCW+11].

The major concern of most of the ML applications in this area is dependence of training sets on historical data, which are often limited. Apart from having limited access to past data, the size of the feature vector is usually large as the test-set size is large, leading to over-fitting at the time of training. In order to overcome this, a scheme called syndrome merging was used to reduce the size of the feature vector [SJX+15]. Note that some type of syndromes may not be observable or computable. For such cases, another technique was proposed in [JYZ+16] to process the training set including those based on nave Bayes classifiers.

2.5.3 Test Compression

The test-cost is measured in terms of the test data volume, and test time. In scan-based test environment, one way to reduce test-cost is the adoption of the compressor/decompressor architecture (CODEC). A pseudo-random pattern generator (PRPG), sometime along with a decompressor, is used to load the scan-chains. Similarly, the test-response data is com-

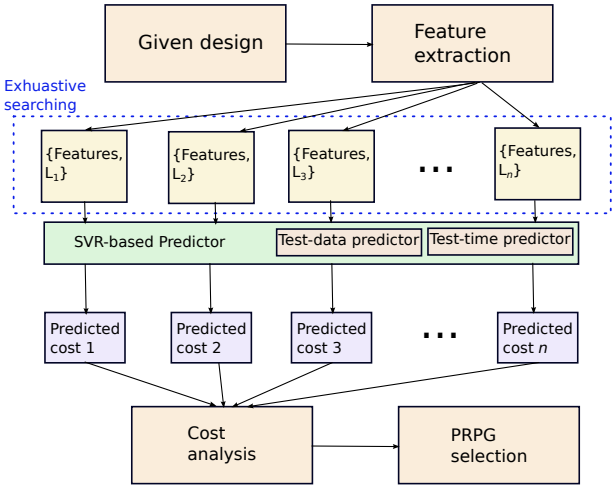


Figure 2.14: Illustration of the PRPG-selection method [LCP+17].

pressed by using a multiple-input-signature-register (MISR). It has been shown that besides various circuit parameters, the length of the PRPG greatly affects the test-cost [LCP+17]. While the problem of designing the pattern generator can be resolved exhaustively by running ATPG, it may become infeasible because of the time needed. A predictor-based on SVR (support vector regressor) is proposed to tackle this problem [LCP+17]. A number of features are extracted from the ATPG log-file from which suitable features are selected.

An illustration for selecting the length of PRPG is given in Figure 2.14. Two separate predictors, one for test time and the other for test-data volume are trained. For each choice of the length of PRPG, test-cost is predicted, and the length corresponding to the minimum cost is selected.

2.5.4 Circuit Testability

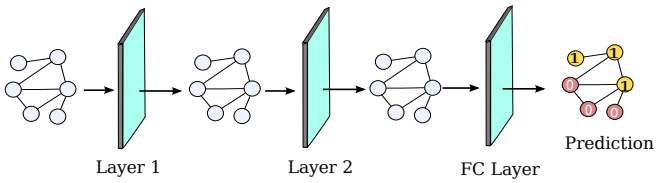


Figure 2.15: Network architecture of GCN. Node embeddings are generated in Layer 1 and Layer 2. The fully-connected (FC) third layer execute nodes classification [MRK+19].

The problem of test-point insertion in a logic circuit has been studied in [MRK+19] from ML perspective and a classifier has been built. This is the first time where a deep-learning based technique has been deployed in handling a test problem. Moreover, attempts have been made to learn from circuit-graphs; this is a challenge because ML-tools are more

suited for structural/vector data, whereas a graph mostly comprise unstructured information. The authors in [MRK+19] propose a neural network called graph convolutional network (GCN) to analyze graphical data. The nodes of the graphs representing the circuit netlist are classified as either easy-to-observe or difficult-to-observe points. Graphical features are represented through a node embedding method. A number of attributes related to testability obtained using the tool SCOAP [GT80], are attached to each node. Based on these attributes and the local neighborhood information of a node, an embedding of each node is produced by the GCN. The overall flow of the classifier is shown in Figure 2.15.

2.5.5 Timing Analysis

Timing analysis of a circuit is required to determine the clock frequency of the circuit. The timing of a circuit depends on many static as well as dynamic (input pattern dependent) characteristics.

Power supply noise (PSN) affects the input voltage reaching the gates and hence the propagation delay. It is one of the factors that influences dynamic timing analysis (DTA) of the circuit. In order to speed up DTA, a ML approach to predict the circuit timing, taking into account the PSN-effect is given in [LHLL17]. The prediction of circuit delay due to voltage droop is proposed in [YFY+16] using SVM.

2.5.6 Summary, Challenges and Future Directions

A summary of the relevant literature is given in Table 2.1. Next, we will look at the various challenges and future directions. Although there are many scenarios concerning digital logic testing where ML has been or could be applied, they still appear to be fragmented and unorganized. The success of ML-based techniques strongly relies on the availability of sufficient data with good quality and volume. Whilst some potential data sources can be accessed as enlisted in Section 2.2, standard ML-databases in regard to integrated circuit testing are yet to be prepared, and thus, their unavailability stands as major impediment towards the adoption of ML-tools. Some of the reasons behind this bottleneck are listed below:

1. Absence of industrial time-series test data: Most of the databases on the failure log collected during production testing of ICs along with the corresponding diagnostic information, are not available in the public domain. Such data would serve as a rich source for updating training models and help guide future diagnosis processes for IC-chips.
2. Complexity in simulation: The generation of simulated data is a very time consuming process. A general repository of simulated data would serve as a good data source and may strengthen ML-based tools.

Table 2.1: Summary

Work	Problem	Data	Method
Wafer level diagnosis			
[SNW17]	Identifying defect clusters	FD	Clustering
[XSRM17]	Automated die inking	Historical data	SVM
[TSS+14]	Correction of failure and parameters	FD	Statistical Correlation
Scan-chain diagnosis			
[HBK+17]	Targeting hard-to-model faults	FD	Bayesian method
[CLH+19]	-	SD	Multi-Stage ANN
Fault diagnosis pre-processing			
[WPY+12]	Regulation of test data volume	FD	Classification
[HFMB18]	Inferring diagnostic efficiency	FD	Random Forest
Fault diagnosis post-processing			
[GW16]	FI: Defect classification	SD	ANN
[NTB10]	FI: Identifying bridging defects	SD & FD	Decision Tree
[GCI+17]	FI: Transient and intermittent faults	SD	Bayesian Network
[XPLB13]	Improving diagnostic resolution	SD	SVM
Volume diagnosis			
[WW09]	VD of unmodelled faults	SD	SVM
[CTR17]	VD for root cause identification	Volume FD	Bayesian Network, MLE
[HKP04]	Identification of systematic defects	FD	Clustering (Furhest Neighbour)
Board level diagnosis			
[SJX+13, YZCG13, YCZG15]	Fault isolation	Historical data	SVM/ANN/Decision tree
[SJX+15]	Syndrome merging	-	-
[JYZ+16]	Missing syndrome computation	-	Naive Bayes
Test compression			
[LCP+17]	Test cost optimization	SD	SVR
Circuit testability			
[PBCB18]	Prediction of X-sensitivity	SF &SD	SVR
[MRK+19]	Test Point Insertion	SFs, SCOAP, SD	GCN
Timing analysis			
[LHLL17]	Based on PSN	SD	Multiple tools

FI: Fault identification; SD: Simulated data; FD: Failure data; VD: Volume diagnosis; SF: Structural feature

3. Absence of baseline: There are no benchmark circuits for evaluating ML-approaches to test problems. The circuit structure can, however, be extracted from the netlist of the benchmark circuits. They can serve as a potential source of data from which structural as well as functional features can be derived. The benchmark suits currently available [BF85, BBK89, CRS00, Alb05] are not meant for ML studies. Among them, there is a lack of diversity and volume. In order to create such data, a number of large benchmark circuits having variety in their interconnection structure and functionality are to be built. Such a repository may be built by collecting industrial circuits or by unbiased synthesis.
4. Feature extraction from circuits: It is evident that feature engineering is a crucial step in most of the settings. In the context of digital logic testing, there is an urgent need for automatic feature extraction from circuit net-lists. It is also required to avoid over-fitting and to filter out noisy data (in the case of output response data from a circuit-under-test). Note that circuit-data are not always in learnable or structured format (e.g., logical interconnection or physical layout data). Hence, a significant amount of time and effort is lost in this process. Also the features are rarely reusable, because most of the time, a new feature set is required for every different test problem. As manual feature engineering is a cumbersome process, automated feature extraction from circuits is highly needed. This is still an open problem in the area of logic testing albeit there have been many such endeavors in other fields such as pattern recognition and image analysis. Deep learning has been applied successfully on image and video data where the underlying features, instead of being separately extracted/selected, are implicitly utilized during learning and testing, based on convolutional techniques. Also, new methods such as representation learning are emerging to handle unstructured data, which prepares the data for direct application of ML-tools. The work proposed in [MRK+19] has pioneered efforts in this direction, with the introduction of graph convolutional network (GCN) for node embedding of a graph, representing the netlist of a circuit.

All the techniques discussed above for feature extraction require a large volume of data, which are not available in the domain of IC-testing and fault diagnosis. Solutions to these challenges of data generation and automated feature engineering will kick off the adoption of ML-approaches to chip testing in the future. Needless to say, there remains enough scope for data generation and representation techniques for digital circuits that will enrich industrial as well as academic research in the area of ML driven test.

2.6 Summary

This chapter provides the background of our research. In this chapter we have presented the basic of digital testing and diagnosis. We have also discussed the basics of machine

learning. We have presented a review of diagnostic test generation and solutions to the problem due to an unknown logic value. Lastly, we have reviewed the state-of-the-art of ML applications in this field.

CHAPTER

THREE

SELECTING DIAGNOSTIC VECTORS FROM DETECTION TEST SETS FOR LOGIC CIRCUITS: A COMBINATORIAL SOLUTION

3.1 Introduction

The increasing complexity of VLSI circuits necessitates nano-scale device miniaturization as well as an intricate manufacturing process for their fabrication. This, however, has led to greater process variation, and higher probability of defects, and consequently, has worsened yield. As a result, the diagnosis problem has not only surfaced as being more difficult but also more important since the reason for low yield needs to be ascertained at the netlist level, and necessary corrections be made in the manufacturing process so as to enhance the yield. Diagnosis is also needed to enable the use of partially defective chips, on the basis of the location of defects. Various methods for faulty-site location that are employed during yield ramp-up such as the traditional Physical Failure Analysis (PFA) and volume diagnosis [TMR+07b], depend on logic-level processing called fault diagnosis. This refers to the process of logically analyzing the failed chip and isolating a candidate fault or a set of faults that best explains the cause of erroneous response when the production test patterns are applied. Several earlier research work [Pom15, AA13, KPC+13, Pom16, LLC07, Pom14a, KJC+14, TCG+11, YB08] aimed at increasing the accuracy of fault or defect diagnosis. There are two approaches to diagnosing a failed chip: cause-effect analysis [ZA10] and effect-cause analysis [LLC07, Pom14a, KJC+14, TCG+11, YB08].

Various ATPG tools available today [LH93, J-2], employ powerful heuristics and are

capable of generating efficient and compact test sets for a circuit-under-test (CUT). These tools provide tests with high fault-coverage, and identify redundant faults in the CUT to a large extent, even though the underlying problem is known to be computationally hard. However, the production tests are primarily aimed only for fault detection, and they may not be very efficient for diagnosis. Note that a fault is usually detected in more than one outputs of the CUT for a given test; also, there are several test vectors that can detect the same fault. ATPG tools aim to minimize the size of the test set and maximize fault-coverage. Needless to say, the test sets, thus generated, are not unique, and their diagnostic ability, measured in terms of diagnostic coverage (DC) as defined later in Section 3.2, may also be diverse.

In this work, we consider single stuck-at fault model and address the diagnosis problem from a different perspective. Our solution is based on the diversity of the test sets that is observed when an ATPG tool is invoked multiple times on a given circuit netlist. Our method is simple and needs only input/output experiment with a pool of detection test sets, and a fault simulator [LH96], [J-2]. We do not require to run diagnostic ATPG [ZA10], or adopt any circuit-modification [WLL14], SAT-based approaches [YZHL10, RSRB15] or miter-based techniques [YZHL10]. The proposed method is purely combinatorial in nature - that of finding a cover of a response-matrix, which is obtained by invoking an ATPG and fault simulation tools multiple times on the CUT netlist. We propose a compact representation of the response matrix so as to reduce computational time and space complexity. We propose two variants of an algorithm, both based on the same framework. Algorithm 1 generates a response matrix for the entire test pool and aims at finding a diagnostic test cover following a greedy approach. Algorithm 2 runs faster as it performs partial simulation of the test pool while determining diagnostic test cover. Experimental results on several combinational and scan-based benchmarks reveal the effectiveness of the proposed method compared to prior art in terms of DC or the number of test-equivalent pairs, the number of test patterns, and CPU-time.

The rest of the chapter is organized as follows. Section 3.2 outlines the preliminaries on circuit diagnosis. Section 3.3 presents a review of earlier work. In Section 3.3, we show a motivating example. Section 3.5 describes the proposed technique. Experimental results are reported in Section 3.6. Concluding remarks and open problems appear in Section 3.7.

3.2 Background

The following preliminary concepts are needed while determining a *DTS*.

- *Distinguishable fault-pair*: A fault-pair is said to be distinguishable if there exists a test vector that detects one fault but not the other at some output, or if it detects both faults at different outputs. Such a test vector is called *distinguishing test vector*.

- *Functionally-equivalent (FE) class*: If no distinguishing vector exists for a fault-pair, the two faults are said to be *functionally equivalent* [ZA10] or *equal* [YZHL10]. Faults belonging to a functionally-equivalent class are indistinguishable, and they can be collapsed to a single representative fault. A pair of faults belonging to different equivalent classes are distinguishable [ABKS03]. Early studies on equivalent faults appeared in a classic paper [MC71].
- *Test-equivalent (TE) class:* A class of faults that cannot be distinguished from each other by applying a set T of test vectors, are said to be test-equivalent with respect to T. Each fault belonging to a TE-class having more than one element is called *undiagnosed fault*. A fault belonging to a TE-class with only one fault is said to be *diagnosed* since the test set can uniquely identify it.
- *Diagnostic simulation (DS)*: Given a test set T for a number of faults in a circuit, DS performs fault simulation for each test vector in T and partitions the faults into minimum number of groups based on their output values such that all faults belonging to a group are equivalent with respect to T. DS enables us to discover the TE-classes among the set of faults.
- *Diagnostic coverage (DC)*: This metric is quite similar to one defined in [ZA10]:

$$\boldsymbol{DC} = \frac{total\ number\ of\ TE\text{-}classes}{total\ number\ of\ FE\text{-}classes}$$

  Thus, DC indicates the diagnostic efficiency of a test set. When the number of TE-classes becomes equal to the number of FE-classes, DC becomes 100%.

3.3 Related Work

All related prior work aim at generating additional vectors so as to improve the diagnostic coverage that is achieved by the detection test set. Gruning et al. [GMK91], proposed a tool for diagnostic test pattern generation (DTPG) based on a branch-and-bound technique for circuit traversal. A miter-based technique to generate distinguishing test vectors using two copies of the circuit was proposed in [ABKS03]. Based on the observation that the faults in the same fan-out free region are harder to distinguish than the faults being in different fan-out free regions, a method for diagnostic test pattern generation is proposed in [YZHL10]. A miter-based circuit is considered for test pattern generation and a module "SA1" is used to inject a pair of faults to the circuit such that the fault-pair is injected if the selection line of the model is set at 1. Unlike [YZHL10], which uses two copies of a circuit for generating a test, the method proposed in [VCAA04] shows that similar result can be obtained by using only a single circuit. This technique is used in [ZA10] to generate *exclusive tests* for faults, if any, by inserting two multiplexers per fault-pair. It proposes a simple diagnostic metric and also discusses a method called dictionary-based fault diagnosis. The work presented

in [WLL14] provides an improved result using one exclusive test to distinguish many pairs of faults when they have non-intersecting output cones. A DTPG tool based on fault-pair collapsing is proposed in [PR07a]. It relies on certain kind of structural analysis of the circuit, and diagnostic test sets are generated using a method of test elimination [PR98]. A SAT-based technique to generate a test set with high diagnostic power is also reported [RSRB15].

Efficient fault collapsing makes DTPG more effective. Previous efforts in this direction include collapsing of equivalent faults [AFPB03], identification of fault-pairs that are guaranteed to be distinguished by the fault detection test set [PVRS04], or fault-pair collapsing [PR07b, PR07a].

3.4 Motivational Example

In this section, we present a motivational example to illustrate the proposed method. Consider the circuit as shown in Fig. 3.1. The circuit has six inputs (1, 8, 2, 3, 4, 5) and three outputs. It has 38 structurally collapsed (based on the dominance and equivalence relation of the faults)[APA03] stuck-at faults: {12^1, 6^1, 12^0, $10a^1$, $2a^1$, 1^1, 2^0, 2^1, 10^1, 10^0, $7a^1$, 8^1, 7^0, $2b^1$, 7^1, 3^1, 13^1, 11^1, 13^0, $10b^1$, $7b^1$, $4a^1$, 4^0, 4^1, 16^1, 15^1,1 4^1, 16^0, $5b^1$, $9b^1$, $10c^1$, $9a^1$, 9^0, $5a^1$, $4b^1$, 9^1, 5^1, 5^0}, all of which are detectable. It has three FE fault-pairs {{$5b^1$,$10c^1$}, {$9b^1$, 9^1}, {16^0,9^0}}, as depicted in the figure. The total number of FE-classes in this circuit is 35. Hence, in order to achieve 100% diagnostic coverage, we need to generate a test set that produces 35 TE-classes.

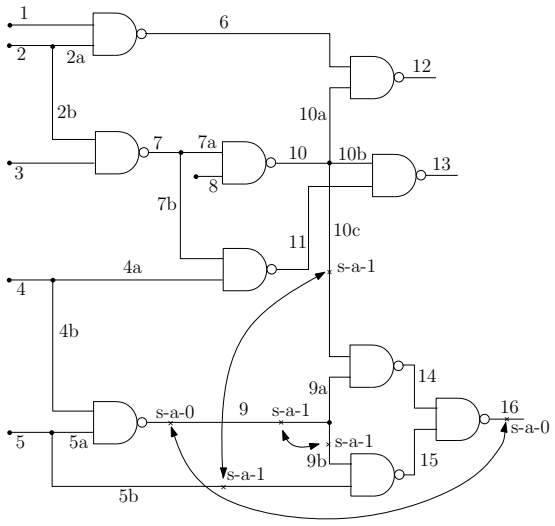


Figure 3.1: Example circuit

Table 3.1: Test-equivalence classes with more than one elements

Test set	Test-equivalent (TE) classes	Equivalent classes of the union so far
T_0	{{5b^1,10c^1},{9b^1, 9a^1,9^1}, {7a^1,7^1},{16^0,9^0},{4b^1,15^1}}	{{5b^1,10c^1},{9b^1, 9a^1,9^1}, {7a^1,7^1} ,{16^0,9^0}, {4b^1,15^1}}
T_1	{{5b^1,10c^1},{16^0,9^0},{5a^1,14^1}, {9b^1, 9a^1,9^1},{7a^1,7^1,7b^1,2^0}}	{{5b^1,10c^1}, {7a^1,7^1}, {9b^1, 9a^1,9^1},{16^0,9^0}}
T_2	{4b^1,15^1},{16^0,9^0},{5b^1,10c^1}, {7a^1,7^1},{9b^1,9^1}}	{{5b^1,10c^1},{7a^1,7^1} , {9b^1,9^1},{16^0,9^0}}
T_3	{{34,37},{4b^1,15^1},{5b^1,10c^1}, {16^0,9^0},{9b^1, 9a^1,9^1},{7b^1,7^1}}	{{5b^1,10c^1},{9b^1, 9^1}, {16^0,9^0}}

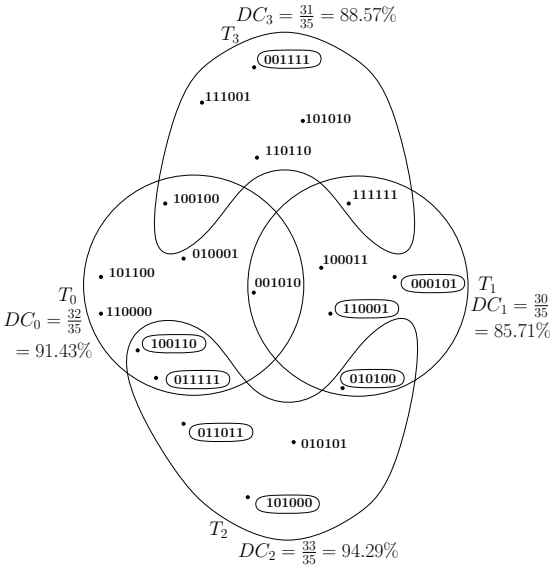


Figure 3.2: Four test sets of the circuit given in Fig. 3.1. The encircled vectors form the required diagnostic test set.

We generate a number of test sets using an ATPG tool successively. Fig. 3.2 shows the four selected test sets. As shown in the diagram, none of these test sets is exclusive i.e., each of them has some test vectors common to other sets. Also, the test sets contain different number of vectors. In this example, each of test set provides 100% detection coverage.

Initially, a test set T_0 with seven vectors is generated. Diagnostic simulation of T_0 resulted in 32 test-equivalent classes, and thus, $DC_0 = \frac{32}{35} = 91.43\%$. Table 3.1 shows the test-equivalent classes. The equivalent classes that contain a single fault are not shown. We keep on adding test sets till the diagnostic coverage increases. The next test set to be selected should increase the diagnostic coverage, i.e., further partition the current test-equivalent classes. The second test set T_1 consists of six test vectors and has $DC_1 = \frac{30}{35} = 85.71\%$.

The third column of Table 3.1 shows the test-equivalent classes for $T = T_0 \cup T_1$ consisting of 12 vectors, it produces 33 test-equivalent classes with $DC = \frac{33}{35} = 94.29\%$. The test set T_1 is responsible for distinguishing the fault-pair $\{4b^1,15^1\}$. Next, test set T_2 that consists of six test vectors and provides $DC_2 = \frac{33}{35} = 94.29\%$ is added, which now distinguishes fault $9a^1$. The test set $T = T_0 \cup T_1 \cup T_2$ consists of 16 vectors and produces 34 test-equivalent classes; thus, $DC = \frac{34}{35} = 97.14\%$. Finally, a test set T_3 with six test vectors and $DC_3 = \frac{31}{35} = 88.57\%$, is added. It distinguishes the pair $\{7a^1,7^1\}$. Now, T, the union of the above four test sets, consists of 19 vectors and provides 100% DC with 35 test-equivalent classes.

Interestingly, out of 19 test vectors thus obtained, we observe that *only eight test vectors* (encircled in Fig. 3.2) are sufficient to provide 100% DC, and that too preserving the same fault-detection coverage of 100% as with each of four test sets. Also, the size of this DTS or "diagnostic test cover" is just one more than the maximum size of these test sets. Thus, a suitably selected cover of a few detection test sets may provide a small-size DTS with high value of DC.

3.5 Proposed Work

We now formulate the combinatorial problem for determining a diagnostic test cover assuming the stuck-at fault model. Our approach is based on the following observations:

Observation 1. *The detection-test set of a circuit obtained by ATPG tools for a particular fault model is, in general, quite diverse in nature.*

Note that for a given fault, there may exist a number of vectors that can detect the fault and this fact is true for most of the faults, excepting the hard-to-detect faults. An ATPG tool experiences various options while selecting the smallest detection test set. Different test sets produced by an ATPG tool, may provide the same fault coverage. Table 3.2 shows the result, where five test sets have been generated using two ATPG tools: ATALANTA [LH93] and Synopsys TetraMAX [J-2] for a few ISCAS'85 [BF85] and ISCAS'89 [BBK89] circuits. By setting an initial random seed, it is possible to generate different test sets in ATALANTA. In TetraMAX, we can obtain different test sets by randomly filling the 'don't cares' in each pattern. In order to demonstrate this variability, we have generated five different test sets. Let T denote the set of these five sets: $T = \{T_1, T_2, T_3, T_4, T_5\}$. For each test set $T_i \in T$, we define $C_i = T_i \cap \sim \{\bigcup_{j=1}^{5} T_j \mid j \neq i\}$, which denotes the set of test vectors that are exclusive in T_i, i.e., they are included in T_i but not in the union of the remaining other test sets. Each entry in Table 3.2 shows the value of $|T_i|/|C_i|$. Note that except for the small circuits c17 and s27, each of the test sets, thus generated, comprise mostly exclusive vectors.

Table 3.2: Analyzing the randomness of ATPG generated test sets for some ISCAS benchmark circuits.

Circuit	ATALANTA [LH93]					TetraMAX [J-2]				
	$\|T_1\|/\|C_1\|$	$\|T_2\|/\|C_2\|$	$\|T_3\|/\|C_3\|$	$\|T_4\|/\|C_4\|$	$\|T_5\|/\|C_5\|$	$\|T_1\|/\|C_1\|$	$\|T_2\|/\|C_2\|$	$\|T_3\|/\|C_3\|$	$\|T_4\|/\|C_4\|$	$\|T_5\|/\|C_5\|$
c17	4/1	6/3	6/5	6/3	6/2	5/1	6/2	5/0	6/3	5/0
c432	49/48	47/46	48/47	51/50	51/50	50/50	49/49	52/52	48/48	50/50
c880	51/51	51/51	53/53	48/48	60/60	36/36	35/35	37/37	34/34	33/33
c1908	120/120	118/118	118/118	117/117	116/116	40/40	42/42	37/37	40/40	37/37
c3540	149/149	155/155	148/148	154/54	149/149	99/99	109/109	107/107	104/104	111/111
c6288	34/34	26/26	35/35	25/25	30/30	30/30	29/29	33/33	40/40	36/36
s27	7/2	7/4	6/3	7/3	6/5	9/6	9/3	7/2	8/3	6/1
s382	34/34	34/34	32/32	36/36	33/33	29/29	30/30	32/32	31/31	31/31
s526	61/61	64/64	65/65	62/62	66/66	61/61	60/60	58/58	56/56	59/59
s832	111/108	114/114	110/110	111/110	112/109	110/110	111/111	101/101	106/106	106/106
s1494	130/109	121/105	126/111	123/101	125/107	117/95	119/92	121/105	116/97	121/101

Observation 2. *The union of two test sets is likely to increase DC.*

A test set containing a larger number of vectors is likely to have a higher DC than the one with fewer vectors. For two different test sets having the same DC, their union often leads to an increase in DC, whenever they have different TE-classes, i.e., when they distinguish different sets of faults. This is demonstrated in the motivational example discussed earlier.

Observation 3. *If two faults in a circuit that are detected independently (one at a time) by a test vector produce different output vectors, they are distinguishable.*

This observation leads to a combinatorial formulation of the diagnosis problem. A distinguishing test vector ought to produce two different output responses for two faults. The TE-classes for a test vector can be computed by comparing fault simulation outcomes.

The response vector (R) for a fault is an m-bit binary string, where m denotes the number of circuit outputs (primary and scan-outputs for sequential circuits). The output bits of R are set to 1 where the error is observed. Thus, if the response vectors of a test pattern are different for two faults, they are distinguishable.

In our experiment, the primary output pins are considered in a fixed sequence such that the corresponding binary string consisting of observed values can be fairly compared. For scan-based circuits, the flip-flop states, after each test cycle, are also observed similarly, following the primary-output pins. Thus, the output vector reflects a fixed sequence of response bits that appear at the primary and secondary outputs.

3.5.1 Data Structure: Response Matrix

For large circuits, especially for scan-based circuits, the value of m is large. Instead of comparing two large binary strings for checking distinguishability of two faults, we may consider the unsigned integer of these binary strings. Hence, instead of constructing a three-dimensional ($|F| \times |T| \times m$) binary response matrix, we consider an $|F| \times |T|$ two-dimensional integer response matrix, where F is a set of detectable structurally-collapsed faults and T is a set of test vectors. However, this would entail a new problem of memory overflow. Typically, a large benchmark circuit may have more than thousand outputs. The unsigned integer representation of such large binary stings would be very large. In order to tackle this problem, we propose a method called matrix relabeling. Note that in order to check whether a test vector distinguishes two faults, their output vectors/ integer values should be different; their actual magnitudes are of no significance. So, for each test vector, the output integer values for all faults can be re-labeled with smaller values preserving their distinguishability. In other words, each entry in a column of the 2D Response Matrix can be relabeled with smaller values (see Fig. 3.3).

We construct a relabeled response (RR) matrix M, of dimension $|F| \times |T|$. Each entry $M[f_i, t_j]$ of the matrix consists of relabeled value of the unsigned integer representation of the output vector when test t_j is applied in the presence of fault f_i. In our experiment, we have relabeled incrementally starting from integer 1, for each test vector, and use the same label if the corresponding integer appears earlier in the column. So, the maximum value in a column is at most $|F|$ (the bound is achieved when all faults are distinguished by the corresponding test vector). Note that during relabelling, the binary strings need to be stored temporarily in a dictionary for comparison.

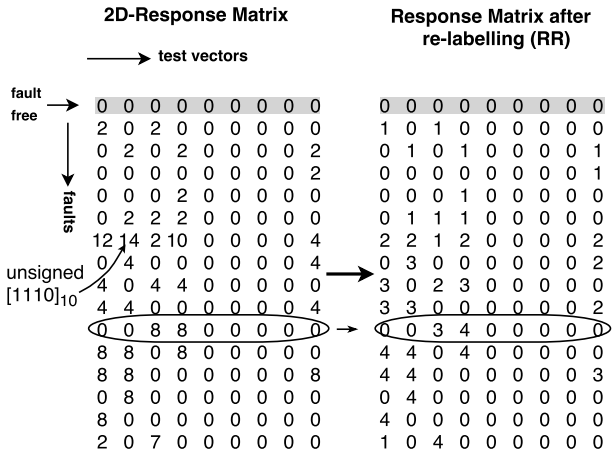


Figure 3.3: An example of matrix re-labeling for s27 circuit

Fig. 3.3 shows an example of 2D-Response Matrix and RR-matrix for s27, an ISCAS'89 circuit [BBK89], corresponding to a detection test set with 9 test vectors generated by TetraMAX. For simplicity, only 15 faults (rows) out of 36 structurally-collapsed faults are shown in the figure. An extra row corresponds to the fault-free response (first row in shade with all 0's) is shown in the figure. Such an additional row is needed in our algorithm to ensure the detectability of all faults; this is explained in the next sub-section. The 16 $\times$ 9 matrix on the left (Fig. 3.3) is a response matrix where the output binary vectors are represented as unsigned integer. An example of an output vector corresponding to the sixth fault for the second test vector is also shown. Circuit s27 has one primary output and three flip-flops, so the first '1' is the value observed at the primary output and the following string '110' represents the states of three flip-flops. The unsigned integer value of [1110], i.e., 14 appears as the corresponding entry of the matrix. The matrix shown on the right is called Re-labeled Response (RR) Matrix. Note that in each column, the maximum value is reduced. For example, the maximum value among the elements in the matrix is 14, whereas in the relabeled matrix it is 4. As shown in the encircled row, the values in each columns have been replaced by smaller values. In general, the maximum value of an element in RR-matrix will be $|F|$, whereas in the output matrix it is $2^m - 1$ where m is the number of output bits. The number of bits representing an element in RR-matrix is, therefore, $\log_2 |F|$. This reduction is more evident for large circuits, especially for scan-based circuits (refer Table. 3.3).

Table 3.3: Number of bits needed for the entries in the original and RR-matrix.

Circuit	#bits in output string	$log_2\|F\|$
s15850	648	14
s35932	2048	16
s38417	1742	16

Observation 4. *Each row in the $|F| \times |T|$ RR-matrix becomes distinct only if all faults are found to be distinguishable by the test set T.*

The distinctness of each row implies that for every fault in F, there exists at least one entry in it, which is different from an entry in the same column of another row, and such an entry in its row exists for every other row. The corresponding test vector is the distinguishing test vector for the fault-pair.

3.5.2 Computing the Equivalence Class of a Test Set T from an $|F| \times |T|$ RR-Matrix

Identification of equivalent classes of faults can be accomplished by reordering the rows such that the faults having identical outputs in the RR-matrix for the test vectors in T are

placed together. Starting from the first test vector in the matrix, all rows are reordered such that the faults having same values in the first column are kept in the same class. This gives us the TE-class for the first vector. Next, each class is considered separately, and further classes found by reordering the rows considering the second column within each class. This is continued till the last vector is processed. The final set of classes gives the TE-classes of T.

3.5.3 Proposed Algorithms

This section describes the techniques for constructing a DTS and two variants of the proposed algorithm. The algorithms consist of two phases:

Phase I. Test Set Selection: The aim of this phase is to augment the current pool of test vectors with those in the new test sets, so as to maximize DC. A number of different test vectors are generated by randomly filling the don't-care bits in a test vector obtained by an ATPG tool. Each time a new test set is generated, it is chosen only if it leads to an increase in the number of TE-classes.

Phase II. Test Vector Selection: The aim of this phase is to obtain DTS, diagnostic test cover of the pool of test vectors selected in Phase I.

Observation 5. *If we add a row corresponding to the fault-free response in an $|F| \times |T|$ RR-matrix then the distinctness of rows will guarantee detection of all faults.*

In other words, if we consider a pseudo-fault f', which has the same output as that of a fault-free circuit, then the existence of a TE-class, which contains only the fault f', guarantees that all other faults are detectable by T. An example is shown in Fig. 3.3 where the RR-matrix of the circuit, s27 is shown. The row corresponding to the fault-free response is shown shaded.

We remove the redundant and aborted faults (reported by ATPG tools) from the list of structurally-collapsed faults before invoking our algorithm. Let F be the total number of detectable structurally-collapsed faults and T_a be its detection test set. We decide the distinguishability of two faults by T_a based on Observation 3. Let the pool of test sets selected be stored in T.

Greedy election of diagnostic test set (DTS)

The pseudocode of the procedure is given in Algorithm 1. The variable E_T stores the TE-classes attained so far. Initially, E_T contains only one class containing all faults in F. In each iteration of the while loop in line 1, a test set T_a for F is generated by calling an ATPG tool. An RR-matrix ($u \times |T_a|$) is constructed to check whether new classes are formed by T_a, where u denotes the number of undiagnosed faults. If so, T_a is chosen and added to

the pool T. The number of allowable trials is controlled by the value of $BOUND$, which is user-specified. The algorithm can terminate earlier if all faults have been diagnosed. At the end of Phase I, the variable E_T stores TE-classes of the pool indicating the maximum number of partitions attained by our algorithm, and provides the numerator component of DC. The test-collapsed faults, i.e, the set of representative faults from each class in E_T of this pool T, is stored in F_{DTS}. It also contains a pseudo-fault $f^{'}$ for the fault-free response (Observation 5).

Algorithm 1

Input:
circuit, $F, T \leftarrow \phi, E_T \leftarrow \{F\}$

▷ Phase I: Test Set Selection
1: **while** $BOUND > 0$ **and** some faults in F are not diagnosed **do**
2: Generate a test set T_a for F by invoking an ATPG tool
3: **if** new test set T_a split any TE-classes in E_T **then**
4: $T \leftarrow T \cup T_a$. Update E_T. Reset $BOUND$.
5: **else**
6: Decrement $BOUND$.
7: **end if**
8: **end while**
9: $F_{DTS} \leftarrow$ the set with a fault from each class in $E_T \cup f^{'}$
10: $DTS \leftarrow$ essential test vectors in the last selected T_a
11: $T \leftarrow T \setminus DTS$

▷ Phase II: Test Vector Selection ▷ F_{DTS}: set of faults undiagnosed by DTS
12: Compute E_{DTS} ▷ Set of TE-classes of F_{DTS} for DTS
13: **while** $F_{DTS}! = \phi$ **do**
14: $C^{'} \leftarrow$ the set of faults in the smallest class of E_{DTS}
15: $T_c \leftarrow \{all\ t | \max_{t \in T}$ (no. of TE-classes of $C^{'}$)$\}$
16: $t_{max} \leftarrow \{t | \max_{t \in T_c}\{$(no. of TE-classes of F_{DTS} for $DTS \cup t$) + (no. of faults in F_{DTS} detected by t and not by DTS)$\}\}$
17: $DTS \leftarrow DTS \cup t_{max}, T \leftarrow T \setminus t_{max}$
18: update E_{DTS} and F_{DTS}
19: **end while**

In Phase II we attempt to find a reduced-sized DTS, which covers the pool, i.e., diagnoses all faults in F_{DTS}. It is updated iteratively, by including one test in each iteration. In each iteration, the set F_{DTS} is updated to contain only the faults undiagnosed by the tests in DTS obtained so far. Similarly, E_{DTS} is updated to contain the set of equivalent classes of F_{DTS}. DTS initially contains essential vectors from the last updated set in Phase I.

Since test vectors are selected from T, the initial test vectors in DTS are removed from T. In each iteration, the test vector that is selected is added to DTS and removed from the set T. An RR-matrix $M(F_{DTS} \times T)$ is used to guide Phase II. It is dynamically constructed while dropping the rows for diagnosed faults in each iteration. This allows us to reduce the space and processing complexity since we do not have to deal with all faults initially contained in F_{DTS}. In order to ensure that all faults are detected, M contains

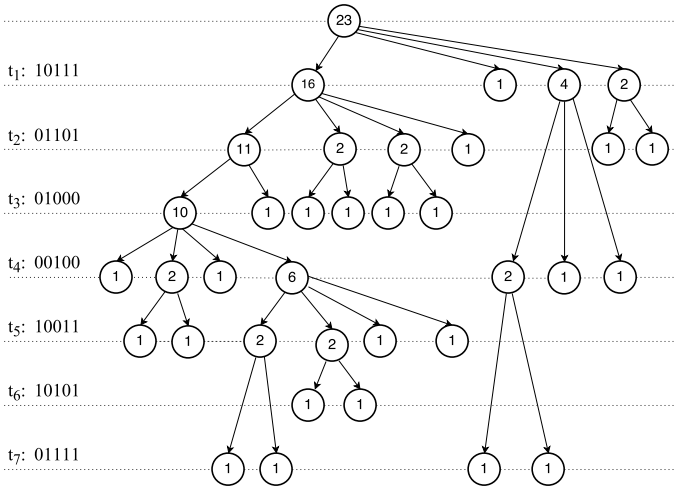


Figure 3.4: Diagnostic tree for DTS of c17. The average depth is 3.65.

a row for the fault-free response corresponding to $f^{'}$ in F_{DTS}. From Observation 5, the element $f^{'}$ is completely distinguished when all faults are detected. A test vector is selected in two steps in order to lessen computational cost. First, a set of candidate test vectors T_c is selected. If a class in E_{DTS} contains exactly two faults, one of the test vectors that splits them should be included in DTS. Hence, T_c contains those test vectors, which can further split the smallest class, namely, $C^{'}$ (usually of size two). If the size of $C^{'}$ is greater than two, then the test vectors that provide maximum splitting of $C^{'}$ are kept in T_c. Next, we evaluate each test vector in T_c. The one that contributes to the maximum increase in detection and diagnostic coverage of the vectors already chosen, is selected. This phase stops when the tests in DTS distinguish all fault-pairs in F_{DTS}.

Since we elect those vectors which are not only essential but also enhance the cumulative diagnostic coverage of the vectors so far chosen, the sequence of vectors thus selected naturally leads to a compact diagnostic tree. This is illustrated for circuit c17 in Fig. 3.4. Each node of the tree at depth i denotes the number of faults in a TE-class for the test vectors t_1 to t_i. In this example, the size of DTS is seven and the average number of test vectors required (average tree-depth) to perform diagnosis is 3.65. Since at each iteration, a test vector has been added to the DTS based on the maximum incremental diagnostic coverage, the same sequence of test vectors can be used to construct a good diagnostic tree.

Complexity Analysis for Algorithm 1: Let τ denote the size of the largest test set T_a, generated by ATPG calls during Phase-1. For the fault set F, let the maximum time required among the calls made to the ATPG tool and fault simulator for simulating a test set, be t_{ATPG} and t_{fsim}, respectively. Let η be the number of test sets selected in Phase-1 that comprise the pool T. Hence, the time for constructing the RR-matrix is $\mathcal{O}(|F| \times \tau)$; the time required for TE-class formation will be $\mathcal{O}(\eta \times |F| ln|F| \times \tau) = \mathcal{O}(|T| \times |F| ln|F|)$. Therefore,

the worst-case running time for Phase I will be: $\{\eta \times BOUND \times (t_{ATPG} + t_{fsim})\} + \mathcal{O}(|T| \times |F|ln|F|)$. The time complexity of Phase II will be $\mathcal{O}(|DTS| \times (|F_{DTS}|(|T| + ln|F_{DTS}|)))$, where DTS is the set of diagnostic test vectors obtained by our algorithm and F_{DTS} is the set of TE-collapsed faults for DTS. Note that because of fault dropping of diagnosed faults in both phases, the actual running time will be much smaller. Additionally, many iterations for selecting test sets cycle through only B times, where $B << BOUND$. The space complexity of the algorithm is $\mathcal{O}(|F| \times |T|)$. Some temporary data structures for storing integers are also required during matrix relabeling, the space and time complexity of which is ignored.

Election of minimal DTS

Note that Algorithm 1 needs full fault simulation for each test vector of the test sets selected in Phase I, and hence, it may take a large amount of CPU-time. In order to improve this, we next propose a variant of selecting diagnostic test vectors. The minimum cover of an RR-matrix can be obtained by dropping the maximum number of columns such that the resulting matrix still have distinct rows. Since this is a hard problem, we aim at finding a minimal cover instead of the minimum. One of the methods to obtain a minimal cover is to start from one end of the matrix and check each vector sequentially while moving to the other end. If the deletion of a vector does not affect the distinctness of each row, then it is dropped. However, checking for distinctness of each row in a large matrix is computationally expensive. The method proposed in Algorithm 2 solves this problem efficiently.

If a column in an RR-matrix distinguishes two rows which could not have been distinguished by any column to its left, then this column is said to dominate all the columns to its left. The RR-matrix is said to hold the *Left-dominance property* if this is true for all columns. The corresponding ordered test set is said to be *Left-dominance (LD) ordered*. Similarly, we can define the *Right-dominance property* of an RR-matrix and *Right-dominance (RD) ordered* test set. A test set which is both LD-ordered and RD-ordered is minimal because each test vector dominates every other test vector in the set. In other words, each vector distinguishes at least one unique fault-pair, which is not distinguished by any other vector in the set. Algorithm 2 is based on the above property.

We can obtain an ordered and minimal DTS using the proposed framework as described below.

Phase I (Test set selection): In this phase, the pool of test sets is maintained as a LD-ordered pool. Thus, once a test set is selected (unlike Algorithm 1), all vectors in the test set are not added to the pool. Only the constituent test vectors that contributed to the selection of the test set are added. Moreover, LD-ordering of such test vectors is maintained while adding them to the pool.

Algorithm 2

Input:
circuit, $F \leftarrow F \cup f^{'}, T \leftarrow \phi, E_T \leftarrow \{F\}$

▷ Phase I: Test Set Selection (LD-ordered test generation)
1: **while** $BOUND > 0$ **and** some fault in F are not diagnosed **do**
2: Generate a test set T_a for F by invoking an ATPG tool
3: **if** new test set T_a split any TE-classes in E_T **then**
4: Add essential LD-ordered tests in T_a to the right of T
5: Update E_T. Reset $BOUND$.
6: **else**
7: Decrement $BOUND$.
8: **end if**
9: **end while**
10: $F_{DTS} \leftarrow$ the set with a fault from each class in E_T (including $f^{'}$)

▷ Phase II: Test Vector Selection (RD-ordered test generation)
11: $E_{DTS} \leftarrow \{F_{DTS}\}$
12: **for** $i = |T|$ to 1 **do**
13: Update E_{DTS} for t_i
14: **if** t_i does not increase classes in E_{DTS} **then**
15: drop t_i
16: **end if**
17: **if** $|E_{DTS}| = |F_{DTS}|$ **then**
18: Drop tests t_{i-1} to t_1
19: Stop
20: **end if**
21: **end for**
22: $DTS \leftarrow T$

Phase II (Test vector selection): In this phase, we drop tests from the pool such that the remaining tests in the pool is RD-ordered as well.

The pseudocode of the proposed method is given in Algorithm 2. The pseudo-fault $f^{'}$ is included in the set F. Initially, E_T contains only one class containing all faults in F. In Phase I, each new test set, T_a is evaluated to check whether it forms new classes in E_T following the method described in Section 3.5.2. We analyze an RR-matrix $M^{'}$ ($\alpha \times |T_a|$), where α is the number of undiagnosed faults in F, so far, starting from the first vector on the left. Thus, if any test vector is found to split some classes in E_T, it is added to pool T, as the right-most vector. Since this vector is able to distinguish at least two rows in the matrix, which could not be distinguished by the vectors already present in the pool, the pool is LD-ordered. Phase II is simpler than the previous algorithm. The set F_{DTS} is a set of test-collapsed faults of E_T (including $f^{'}$) as in Algorithm 1. The set E_{DTS} initially contains a single class containing faults in F_{DTS}. A static RR-matrix, M, of size ($|F_{DTS}| \times |T|$) is used for computation. Starting from the right-most vector (to ensure the right-dominance property), we follow the same method for computing E_{DTS} by re-ordering of the matrix M. We drop the test vectors if they do not contribute to new classes in E_{DTS}. This phase stops when the cover is obtained, that is, E_{DTS} is split into $|F_{DTS}|$ classes by

the test vectors selected so far. The remaining vectors to the left, if present, are dropped. Finally, the vectors in T form the cover, DTS, which is also minimal, i.e., no other test vector can be removed from DTS without degrading DC.

The time complexity of Phase I and the space complexity of Algorithm 2 are of the same order as those for Algorithm 1. However, the actual running time and memory requirement reduce significantly because of smaller size of T. The time complexity of Phase II is $\mathcal{O}(|F_{DTS}|ln(|F_{DTS}|) \times |T|)$.

3.6 Experimental Results

We have implemented the proposed algorithms in C++ programming language to build our diagnostic toolkit COMEDI. The experiments were carried out on Intel Xeon(R) 3.00 GHz $\times$ 4 processor with 8GB memory.

Comparative results for ISCAS'85 [BF85] circuits. The experimental results for Algorithm 1 on ISCAS'85 [BF85] circuits are shown in Table 3.4. We have used ATALANTA [LH93] as the ATPG tool and HOPE [LH96] for fault simulation. Columns 1 and 2 denote the circuit name and the number of collapsed faults, respectively. Column 3 gives the number of equivalent fault-pairs obtained by a DTPG process proposed in [GMK91] and the test-equivalent fault-pairs of the DTS obtained by applying Algorithm 1. Since in [GMK91], the initial fault set is formed by equivalence collapsing of faults in each fan-out free region, the result in Column 3 does not include equivalent fault-pairs in the fan-out free region. This could be the reason that the equivalent fault-pairs reported in [GMK91] is lesser than ours for some circuits. In [GMK91], the authors have proposed a diagnostic ATPG system based on the method of a different ATPG system CONTEST [MGOD90]. Columns 4 and 5 give the number of test vectors and diagnostic coverage, respectively, and the results are compared with [ZA10]. The diagnostic coverage used here is the one define in [ZA10], that if ratio of number of $TE - class$ and the number of structural collapsed faults. Column 6 gives the number of test sets selected, and Column 7 shows the number of ATPG-calls. Columns 8 and 9 show the CPU-time in seconds required in Phase II, and the total running time of algorithm, respectively. Note that we have obtained the same diagnostic coverage as in [ZA10] with a fewer number of vectors. The parameter BOUND provides a mechanism to trade-off DC with CPU-time. We have set its value to 50 in our experiment, as this value is seen to provide a fair trade-off for all the benchmark circuits.

Comparison of results for ISCAS'89 [BBK89] and ITC'99 [CRS00] circuits. The experimental results for ISCAS'89 [BBK89] and ITC'99 [CRS00] circuits obtained using Synopsys TetraMAX ATPG tool, are given in Table 3.5. All results shown in this table are same for both the proposed algorithms. Column 1(7) gives the circuit name, Columns 2(8)

and 3(9) give the number of faults and the number of equivalent fault-pairs, respectively. The result of the two algorithms vary only in the number of test vectors as reported in Table 3.6. In our experiments, we have used the same ATPG Tool, (TetraMAX) as used in [YZHL10], and compared our results with theirs. In Column 3(9), the number of test-equivalent fault-pairs obtained by our algorithm is compared with the number of equal (functionally-equivalent) fault-pairs as reported in [YZHL10]. Note that in [YZHL10], in order to obtain FE fault pairs, the authors analyzed the fault-pairs in two ways: (i) firstly, they aim to distinguish the faults pairs using a miter-based method, and (ii) the fault-pairs that are yet to be distinguished, are targeted using a SAT-solver. Our approach is passive, i.e., we do not target any fault-pair to generate a distinguishing vector for them. We just report the test-equivalent set of our DTS. Although we do not follow any additional exhaustive method or aim to target any fault-pair, for most of the circuits, our results are comparable to those of [YZHL10]; the number TE fault-pairs obtained by DTS is equal to the number of FE fault-pairs. For the majority of cases, we have been able to identify all FE pairs/classes and report the diagnostic test vectors. In some cases, we obtain fewer test-equivalent pairs, which might have arisen because of differences in the synthesized netlist. In Column 4(10), we have reported the number of TE-classes. The information about equivalent classes are more useful for diagnosis rather than the number of fault-pairs. Note that the faults are collapsed on the basis of their classes, and not pair-wise. When the circuit is checked for actual defect location, the class-wise collapsed fault set is considered. Lastly, Columns 5(11) and 6(12) give the number of test sets selected by our algorithm, and the total number of ATPG calls, respectively. Since for most of the circuits, we are able to obtain DTS that distinguishes all faults, which are not functionally-equivalent, we conclude that for the rest of the circuits, the pairs which could not be distinguished are very hard-to-distinguish. As mentioned in [ZA10], FE faults are mostly located in close physical vicinity on the chip regardless of their size. Since the pairs that are missed are very hard-to-distinguish, these faults are likely to have originated from close proximity in the netlist structure, and thus the need for distinguishing them may not be of much concern for the purpose of defect localization.

Table 3.6 compares the number of test vectors; Column 1(3) gives the circuit name. In Column 2(4), the first sub-column gives the total number of test vectors for [YZHL10], the second and third sub-columns show the size of DTS, for Algorithm 1 and 2, respectively. The size of DTS of Algorithm 1 is larger than [YZHL10] for large circuits mostly because we have selected them from random test vectors. For smaller circuits, the results are comparable. For Algorithm 2, the test size of DTS obtained is larger; however it is guaranteed to be a minimal set.

The CPU time in seconds for Phase I, Phase II and total time are given in Table 3.7 for both algorithms. The time taken by Phase II is just a few seconds. The time for Phase I shown in the first sub-column of Column 2(3) includes the time needed for fault simulation and ATPG calls. Since in Algorithm 2, fewer number of test vectors are simulated for all faults, the CPU-time is lesser. This fact is more evident for larger circuits as shown in the table. Previous work [PR07a, WLL14] also dealt with diagnostic test generation; however, we have not compared the size of the DTS obtained by our method with theirs as they used different ATPG tools.

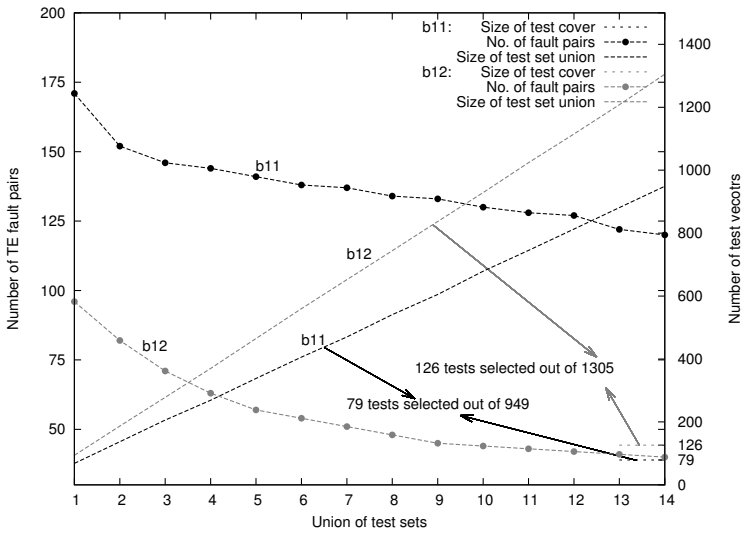


Figure 3.5: Variation of test-equivalent fault-pairs and the number of cumulative test vectors for two ITC'99 benchmark circuits; the size of the diagnostic test set (Algorithm 1) is also shown.

For circuits b11 and b12, the results of Algorithm 1 are shown in Fig. 3.5. The horizontal axis indicates the successive progression of test sets as they are selected. The vertical axis shows the cumulative number of test-equivalent fault-pairs (left side), and the cumulative number of test vectors (right side) corresponding to the successive inclusion of new test sets. It can be seen from the curves marked with spots, that the number of fault-pairs that have been distinguished decreases as more test sets are added. The function saturates when we are left with hard-to-diagnose faults. The smooth lines show the increase in the number of test vectors as new tests are added. Since the plots are almost linear, we conclude that every additional test set contributes exclusive tests vectors to the cumulative DTS. Also, the final size of the diagnostic test cover is shown (126 for b12 and 79 for b11).

Experiment result for large circuits of ITC'99 benchmark suit and a few circuits of IWLS'05 benchmark suit [Alb05] is given in Table 3.8. This experiment was performed on

a 4-core 3GHz Intel Xeon processor with 32GB RAM. Column 1 gives the circuit name. Column 2 reports the number of TE-fault pairs of the detection test set and our DTS. From the smaller value for our DTS, it is clear that a significant number of fault pairs could be distinguished by our DTS compared to the detection test set. Column 3 reports the size of DTS for Algorithm 1 and Algorithm 2. Columns 4 and 5 report the number of test sets selected and number of ATPG calls respectively. Columns 6 and 7 report the CPU-time in seconds taken by Algorithm 1 and Algorithm 2 respectively.

3.7 Conclusion and Future Work

A combinatorial test-selection method for stuck-at fault diagnosis is proposed, which relies solely on the analysis of output responses of the circuit to different test sets. The procedure is simple as no circuit modification or miter-based diagnostic test generation is involved. It utilizes the diversity and randomness of detection-test sets delivered by the standard ATPG-tools, and extracts the diagnostic power hidden in the ensemble. An efficient covering of the response-matrix is proposed to determine a test set with high diagnostic resolution. Experimental results on ISCAS-85, ISCAS-89, ITC-99 and IWLS-05 benchmark circuits show the effectiveness of our approach in terms of various metrics. The technique can be extended seamlessly to handle transition faults as well. Since the selection of test vectors is based only on the output response, in general, it can be applied to any modeled fault if an ATPG and fault simulation tools for the fault model are available. However, like other diagnostic test generation methods, which are based on modeled faults, this method does not target multiple faults. Nonetheless, the test vectors obtained can be used for multiple fault diagnosis following an effect-cause approach [LLC07, Pom14a, KJC+14, TCG+11, YB08]. Our study establishes that a simple ATPG-based procedure is capable of producing a DTS with decent diagnostic coverage. In order to improve DC further, if needed, SAT-based or other targeted methods can be employed to handle the remaining few hard-to-distinguish fault-pairs. As an open problem, it will be interesting to search for properties that may help expedite the algorithm used for RR-matrix covering. Characterization of hard-to-diagnose faults and efficient selection of a few additional test vectors also require further investigation. Moreover, efficient methods for storing the response matrix and performing diagnostic simulation can expedite the process so that it can be effectively applied to large industrial circuits.

Table 3.4: Comparison of results for ISCAS'85 [BF85] benchmark circuits using ATALANTA [LH93] and HOPE [LH96] (BOUND = 50)

Circuit	#Collapsed faults	#Indist. fault-pairs		#Test vectors		Diagnostic coverage		#Test sets	#ATPG calls	Phase 2 time in sec.	Total time in sec.
		[GMK91]	ours	[ZA10]	ours	[ZA10]	ours				
c17	22	0	0	8	7	100	100	2	2	0.2	0.26
c432	520	13	13	69	50	97.51	97.5	12	116	0.33	13.16
c499	750	12	12	53	54	98.40	98.4	2	52	0.26	1.15
c880	942	55	55	70	45	94.16	94.16	3	53	0.27	2.49
c1355	1566	740	740	87	87	59.38	59.39	4	54	1.31	14.89
c1908	1872	295	302	134	121	86.46	86.38	6	61	0.78	16.45
c2670	2632	468	493	150	124	86.42	86.47	10	64	1.27	68.48
c3540	3297	531	540	174	141	89.69	98.69	7	63	1.70	31.54
c5315	5291	447	447	-	126	-	92.2	7	60	2.45	41.00
c6288	7710	1013	1013	137	39	86.87	86.89	5	56	0.80	23.70
c7552	7418	1118	1128	296	188	86.85	86.94	8	67	5.31	183.71

Table 3.5: Experimental result for ISCAS'89 and ITC'99 benchmark circuits using Synopsys TetraMAX [J-2] and comparison with related previous work (BOUND = 50).

Circuit	#Faults		#Equ. fault-pairs		#Equ.	#Test	#ATPG	Circuit	#Faults		#Equ. fault-pairs		#Equ.	#Test	#ATPG
	[YZHL10]	our	[YZHL10]	our	classes	sets	calls		[YZHL10]	our	[YZHL10]	our	classes	sets	calls
ISCAS'89 Benchmark circuits [BBK89]															
s344	342	344	5	5	339	5	56	s349	348	350	10	10	341	7	58
s382	399	391	23	23	368	5	64	s386	384	379	0	0	379	5	14
s420	430	440	13	33	407	5	77	s444	460	452	99	99	369	4	57
s510	564	561	0	2	559	3	53	s526	554	536	34	34	505	4	65
s641	467	487	7	7	480	3	53	s713	543	563	174	174	476	2	52
s820	850	848	42	44	811	12	91	s832	856	854	47	52	811	10	74
s838	857	900	24	65	835	5	85	s953	1079	1095	3	3	1092	4	54
s1196	1242	1247	14	14	1233	9	105	s1238	1286	1289	38	38	1254	7	74
s1423	1501	1452	147	147	1311	8	77	s1488	1486	1483	23	23	1462	8	63
s1494	1494	1491	27	27	1466	10	69	s5378	4563	4518	523	522	4142	14	90
s9234	6473	6401	1229	1229	5401	28	154	s13207	9664	9647	2034	2034	8020	13	66
s15850	11336	10963	2801	2767	9091	9	59	s35932	35110	33705	12893	12893	23783	17	74
s38417	31015	31184	3372	3367	28624	14	98	s38584	34794	33466	2696	2688	31158	19	136
ITC'99 Benchmark circuits [CRS00]															
b03	390	359	2	2	357	6	104	b04	1504	1427	52	52	1383	6	62
b06	157	159	4	4	155	1	51	b07	1132	1084	29	29	1056	2	52
b08	449	424	50	50	400	13	78	b09	419	390	16	24	366	3	53
b10	502	490	6	6	484	6	64	b11	1291	1259	120	120	1170	14	115
b12	2796	2690	40	40	2651	14	75	b13	831	779	44	44	748	5	56
b14	12536	12293	483	610	11814	25	91	b15	23015	22552	1337	1680	21469	15	101

Table 3.6: Total number of diagnostic test vectors for circuits in Table 3.5

Circuit	#Total Vec.			Circuit	#Total Vec.		
	[YZHL10]	Algo1	Algo2		[YZHL10]	Algo1	Algo2
ISCAS'89 Benchmark circuits [BBK89]							
s344	23	23	24	s349	19	25	25
s382	35	33	34	s386	80	69	76
s420	63	78	81	s444	32	29	31
s510	73	62	64	s526	63	59	62
s641	39	36	38	s713	42	35	36
s820	131	121	131	s832	129	120	127
s838	116	159	160	s953	96	87	90
s1196	166	148	162	s1238	176	155	160
s1423	55	50	62	s1488	145	135	138
s1494	144	131	144	s5378	144	150	150
s9234	235	229	294	s13207	278	292	304
s15850	172	171	200	s35932	31	68	83
s38417	137	204	230	s38584	168	265	316
ITC'99 Benchmark circuits [CRS00]							
b03	26	25	26	b04	66	54	66
b06	17	16	16	b07	53	49	48
b08	56	61	62	b09	43	32	34
b10	50	47	50	b11	91	79	91
b12	105	126	135	b13	34	35	39
b14	679	459	521	b15	523	484	567

Table 3.7: CPU time in seconds

Circuit	Algorithm 1			Algorithm 2		
	phase1	phase2	total	phase1	phase2	total
ISCAS'89 Benchmark Circuit [BBK89]						
s344	21.039	0.003	21.042	21	0.001	21.001
s382	24.884	0.005	24.889	24.708	0.001	24.709
s420	33.829	0.012	33.841	32.451	0.001	32.452
s444	29.081	0.005	29.086	23.987	0.001	23.988
s510	20.53	0.011	20.541	20.576	0.001	20.577
s526	27.569	0.009	27.578	26.091	0.001	26.092
s641	20.461	0.003	20.464	20.298	0.001	20.299
s713	23.352	0.003	23.355	23.164	0.001	23.165
s820	47.087	0.082	47.169	42.79	0.003	42.793
s832	39.25	0.064	39.314	35.526	0.003	35.529
s838	53.451	0.053	53.504	50.176	0.004	50.18
s953	26.28	0.029	26.309	23.676	0.003	23.679
s1196	59.641	0.104	59.745	50.967	0.006	50.973
s1238	47.817	0.091	47.908	40.759	0.006	40.765
s1423	50.715	0.033	50.748	45.694	0.003	45.697
s1488	37.979	0.078	38.057	31.687	0.006	31.693
s1494	44.188	0.085	44.273	35.512	0.006	35.518
s5378	464.709	0.566	465.275	184.973	0.02	184.993
s9234	2011.852	1.091	2012.943	973.505	0.045	973.55
s13207	3197.469	1.554	3199.023	2115.646	0.074	2115.72
s15850	1646.117	0.445	1646.562	1268.252	0.067	1268.319
s35932	4507.632	1.003	4508.635	4071.811	0.154	4071.965
s38417	9820.922	4.197	9825.119	6848.716	0.229	6848.945
s38584	9656.091	7.887	9663.978	3400.794	0.401	3401.195
ITC'99 Benchmark Circuit [CRS00]						
b03	36.015	0.002	36.017	36.295	0.001	36.296
b04	39.341	0.022	39.363	32.495	0.003	32.498
b06	17.479	0.001	17.48	18.059	0	18.059
b07	22.437	0.005	22.442	2.827	0.002	2.829
b08	31.684	0.017	31.701	31.819	0	31.819
b09	19.937	0.003	19.94	20.634	0.001	20.635
b10	24.4	0.01	24.41	24.552	0.001	24.553
b11	69.342	0.067	69.409	63.205	0.003	63.208
b12	111.229	0.264	111.493	46.965	0.01	46.975
b13	23.423	0.006	23.429	24.767	0.001	24.768
b14	3513.275	7.74	3521.015	1488.186	0.141	1488.327
b15	9164.734	10.402	9175.136	4269.476	0.192	4269.668

Table 3.8: Results for some large circuit of ITC'99 and few circuits of IWLS'05 benchmark using Synopsys TetraMAX [J-2] (BOUND = 50).

Circuit	#TE-fault pairs		Size of DTS		#Test sets	#ATPG calls	CUT-time in sec. (Algo 1)			CUT-time in sec. (Algo 2)		
	ATPG	DTS	Algo 1	Algo 2			Phase 1	Phase 2	Total	Phase 1	Phase 2	Total
b17	5333	4873	626	696	17	124	82081.66	39.13	82120.79	41754.24	0.65	41754.90
b20s	2400	1528	495	595	32	108	15751.03	18.74	15769.77	4715.77	0.218	4715.9
b21s	2799	2134	500	574	27	141	16678.19	18.14	16696.34	7403.07	0.22	7403.30
b22s	3808	2676	563	668	37	117	42522.52	206.10	42728.63	13029.20	0.55	13029.75
ac97_ctrl	1257	734	164	174	11	105	5503.53	1.73	5505.27	2887.98	0.21	2888.19
mem_ctrl	4124	3595	275	345	14	187	11511.21	4.16	11515.37	7299.58	0.14	7299.72
pci_bridge32	10123	8564	479	570	47	242	179561.86	37.93	179599.79	50322.10	0.53	50322.64

CHAPTER

FOUR

PREDICTING X-SENSITIVITY OF CIRCUIT-INPUTS ON TEST-COVERAGE: A MACHINE-LEARNING APPROACH

4.1 Introduction

During functional or test operation, unknown (X) values may appear at different circuit nodes in a digital system. There are various sources that cause an X to appear such as unspecified inputs or the presence of tri-state elements, IP-cores used as black boxes in the design, unknown states of sequential elements, uncertain timing, clock-domain interface, bus contention, or signal conversion over analog-to-digital boundaries, to name a few [EKR+15]. X-values may also surface out during the post-silicon-validation phase because of underlying design bugs [LHF+12, Goe17].

The presence of X-values strongly impacts the controllability and observability of circuits, severely restraining the capability of automatic test pattern generation (ATPG) systems, fault simulation and test-response compaction. In other words, the testability of faults in the circuit is badly compromised, leading to loss of fault coverage and diagnosability. The unknown values that propagate to the circuit-outputs also influence the design for compactors and logic Built-in-Self-Test (BIST). In order to circumvent the former problem, specially designed ATPG techniques are employed to improve fault coverage in the presence of X's [EKR+15, SEB16], or certain X-masking techniques are employed at the cost of additional hardware and performance degradation, for mitigating the latter class of problems [PKR02, MK02, TWE+06]. In this chapter, we deal with static X-sources as described earlier in the literature [WWN08].

During post-silicon validation [CHJC13] of a manufactured chip, several X-sources

often show up exposing design bugs. In order to reduce full silicon re-spin cost, metal-only engineering-change-order (ECO) processes are performed to correct these bugs. A metal-only ECO can be carried out by changing only a few metal interconnects in the design, which is a very common practice in semiconductor industry now-a-days. Typically, redundant standard cells are sprinkled on the chip-floor as spares, and some of them are allocated to correct the bugs in the design by rewiring a few nets using the spare cells [LHF+12, Goe17, hCMB08, CJC14, CJC13].

From the viewpoint of testability, the presence of X-sources has many implications. As stated earlier, it badly affects the performance of ATPG tools, as X-values invalidate fault excitation and propagation resulting in the degradation of fault coverage. Fault simulation, which has otherwise polynomial-time complexity in a circuit, may become NP-complete in the presence of unknowns [CA87]. Several prior work addressed the problem of efficient test pattern generation [EKR+15, SEB16] and fault simulation [CP89, EKS+14, KSR04, KKM+11] in the presence of X-values. Unknown values at circuit-outputs badly affect the test-compaction ratio since X's corrupt the errors while traveling through the compressor network. In the case of temporal compaction, X's may even corrupt the entire signature. Various methods to overcome the effect of X's on output-data compression have been proposed. The X-blocking technique is used to mask the effects of X-sources by adding extra hardware to the circuit [WWW06]. A few other techniques handle X's when they are captured in scan cells. Spatial X-compaction can be accomplished using a network of XOR-gates [MK02]. Convolutional compactors use memory elements in the XOR-network to reduce aliasing [RTWR03, RT05, AFI06]. X-tolerant test-compaction techniques based on output-bit selection [LLH11] or design of multiple-input signature register (MISR) exploiting X-canceling have also been studied [YT12]. A counter-based circuit was deployed to implement the compactor proposed in [LLH11], [LLH+13]. A method based on scan-chain switching is employed in [WWW+10] to handle unknowns before they feed a compactor. In X-masking, the unknowns in the scan outputs are blocked before forwarding them to the test compactor [WSRW09]. Other techniques for masking and response compaction in a faulty circuit with unknown values when fault-free outputs are known, have also been studied [Pom14b]. Based on the observation that only a subset of scan cells always capture the X-values, the work proposed in [WYL18], groups such scan cells into separate scan chains so as to reduce their effect during output compaction of test responses.

Although various techniques have been developed in the past for the mitigation of X-effects on test-compaction, no systematic attempt has been carried out so far to investigate the effects (sensitivity) of X-inputs on circuit-testability or fault-coverage. More importantly, no efficient and fast method is known that can be used to grade or rank the circuit-inputs with respect to X-sensitivity. Ranking X-sensitive inputs in a logic circuit has a number of applications. In the presence of hardware and other constraints, the most sensitive X-inputs (those causing significant loss of test-coverage) should be prioritized for

X-masking/elimination not only from the viewpoint of enhancing testability but also for re-wiring certain components so that the design bugs causing the X-sources can be rectified during post-silicon validation through the execution of minimal ECO.

Techniques for X-grading will also find some applications to test-time reduction in digital systems. A common practice of handling the X-sources in test environments such as at-speed fault testing [MS08], is to initialize the flip-flops or memory elements to known values so as to prevent the unknowns from entering into the circuit. A good amount of test time is lost in such pre-processing. A possible solution to reducing testing time would be select the flip-flops/memories that have negligible effect on test-coverage, and allow them to remain in unknown states, thereby limiting state-initializations only to a few X-sensitive flip-flops/memory elements.

In this chapter, we propose a method for grading the X-sources in a circuit-under-test in terms of "loss of fault coverage with respect to a test set", using machine-learning techniques. Since X-sources strongly influence testability issues as well as re-spin cost, our analysis would be helpful to a chip designer while making a choice for pre-silicon X-masking (for improving testability or output compaction), or for post-silicon X-correction (for restoring desired functionality and improving reliability of the circuit by ECO).

When an input line of a circuit-under-test (CUT) is set to X, fault coverage reduces and the amount of loss often varies for various inputs. One way to estimate the loss for different X-sources would be to run an ATPG, setting each of the inputs to X. However, this would be computationally time consuming, and more so, because the ATPG needs to be run for every such potential input/X-source. We propose a method here for predicting the effect of X-sources on the loss of fault coverage in negligible time compared to ATPG. Our method is based on a completely different approach: since X-propagation strongly depends on the underlying network structure, we extract certain structural parameters of the circuit and build machine-learning based assessment system using support vector regressor (SVR) that can predict the loss quite effectively for most of the X-inputs. Hence, without the aid of traditional testability measures [DDS92, SPA85, SBA86, GT80] or running any simulation tool, both of which would additionally require the functional features of a circuit, the regressor predicts the detectability loss due to an X-source, solely based on the proposed set of structural features. Note that the objectives of ATPG-tools and X-sensitivity predictors are quite different. The primary purpose of ATPG and fault simulation tools is test-set generation, and we use such tools to train our regressor. Apart from various applications of the proposed regressor listed earlier, it can also be used to improve fault coverage of ATPG-tools as discussed in Section 4.6.4. X-sensitivity prediction is also different from testability analysis [DDS92, SPA85, SBA86, GT80] as the former estimates the loss of detectability in the circuit as a whole with respect to a test-set, whereas the latter provides a measure of controllability/observability of a particular line or a subset of lines. To reiterate, while our regressor uses only the structural information of a CUT, traditional ATPG tools,

random testing, and testability-analyzers require complete structural and functional/gate information of its netlist.

The rest of the chapter is organized as follows. In Section 4.2, the motivation behind the proposed work along with the problem statement is presented. Section 4.3 describes the circuit-parameters that are used for training the regressor and for prediction. Section 4.4 introduces the Support Vector Regressor (SVR) that is used as predictor, and Section 4.5 provides the details of the data analysis and methodology. Section 4.6 reports experimental results. Conclusion and discussions on future work appear in Section 4.7.

4.2 Problem Statement and a Motivating Example

The objective of this work is to investigate the influence of circuit-structure on the loss of detectability in a digital circuit arising due to the presence of one or more X-sources. Furthermore, we would also like to explore whether, instead of deploying traditional and computationally-intensive simulation approaches, a powerful training-based machine-learning regressor (predictor) can be developed to assess the detectability-loss caused by various X-sources, and rank them in the decreasing order of test-coverage regardless of its functionality. Interestingly, our study reveals that a regressor can be designed that relies purely on certain structural features of the circuit. We describe below a set of features (parameters) that capture a number of structural characteristics of the CUT, which influence the testability of stuck-at faults, and consequently, the fault-coverage of a test set produced by ATPG. Selecting a set of features that are minimally-overlapping and representative of various characteristics of the data, always poses a challenge while building a machine-learning system. Also, the total time required for computing the circuit parameters (feature-computation) and predicting the detectability-loss (`DT-Loss`) of test coverage due to X-sources should be negligible compared to exhaustive simulation.

In our analysis, for simplicity, we have considered single stuck-at fault model, though it can be easily extended to include other fault models such as transition faults. Consider a set of faults F, reported as the set of detectable faults by an ATPG tool. On setting a particular X-source s_i to 'X', the ATPG reports a set of faults F_i as detectable. So, the set $UF_i = F \setminus F_i$ is the subset of faults that become undetectable when input s_i is set to X.

The X-sensitivity of an X-source s_i, i.e, the loss of detectability due to the presence of an X-value at s_i is denoted by $\texttt{DT-Loss}(s_i)$.

$$\texttt{DT-Loss}(s_i) = \frac{|UF_i|}{|F|} \times 100\% \tag{4.1}$$

In order to demonstrate the impact of various X-sources on `DT-Loss` in a logic circuit and its dependence on the network structure, we consider an example with ISCAS'89 benchmark s27 [BBK89], which is shown in Fig. 4.1. Assuming each input port as an X-source, `DT-Loss` for each of them is computed by running TetraMAX [J-2] and the results

are shown in Table 4.1. With seven X-sources, the maximum `DT-Loss` is 27.42% for $G5$ while the minimum is 4.84% for $G2$.

It is interesting to note that although both input-ports G5 and G2 have similar-size output cones in the netlist, the `DT-Loss` caused by setting an X-value to each of them alone, differ significantly. The level of the nodes (in the sense of topological ordering [Kah62]) in the output cones as marked in Fig. 4.1, reveals certain structural dissimilarity between G5 from G2. Note that G5 affects the nodes lying at higher levels (which eventually are likely to have bigger-size input cones), compared to those influenced by G2. Furthermore, G5 affects more number of outputs. We observe that for this circuit, DT-loss can be predicted quite satisfactorily based on a few structural features alone. Needless to say, both the functional and structural attributes of the sub-network that is affected by an X-input, determine the DT-Loss in a circuit. In our analysis, we identify certain features of the circuit that are most likely to affect the DT-Loss when an input is set to X. The features we have chosen are purely structural and do not use any functional information. Thereafter, we will build a learning-based regressor and use it for predicting the DT-Loss for an X-input.

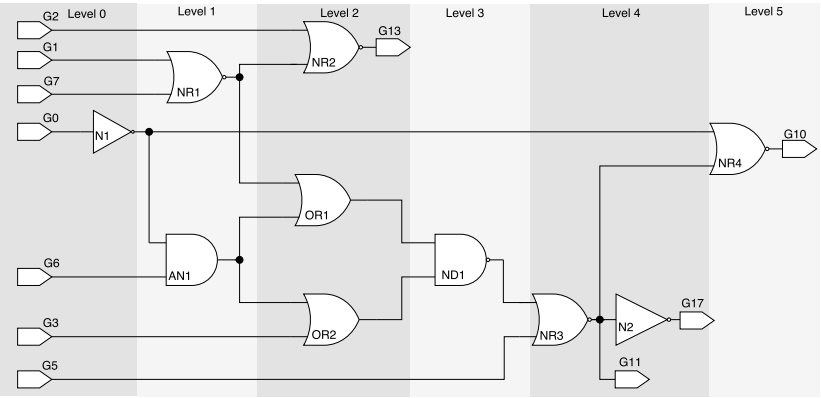


Figure 4.1: A layered, i.e., topologically sorted [Kah62] embedding of ISCAS'89 netlist: s27

Table 4.1: `DT-Loss` for s27

Port	G2	G6	G3	G1	G7	G0	G5
`DT-Loss`	4.84	9.68	9.68	12.9	12.9	16.13	27.42

4.3 Structural Features of Logic Circuits

The training set for the proposed regressor comprises a number of sample points, each representing an X-source, and is drawn from different benchmark circuits. We choose various

sample circuits from the ISCAS'89 [BBK89] and ITC'99 [CRS00] benchmark suites having a wide range of functional and structural diversity. Note that the sizes (i.e., the number of gates, lines) of these circuits also widely vary. Hence, in order to select independent, discriminative, and representative features amidst such variability of circuit-parameters, we need to study not only structural properties of the circuit but also apply certain kind of normalization to obfuscate the relative influence of scale on these characteristic features.

1. A set of structural features that influence the variation of `DT-Loss` for different X-sources in a particular circuit, may not be good enough to explain to the variation of `DT-Loss` for an X-input of another circuit. This is because each circuit may have certain unique structural features. This necessitates the inclusion of several structural characteristics of the circuits in the feature set.
2. Since the circuit-size may vary, each feature of the circuit needs to be normalized so that the features of X-sources drawn from different circuits become comparable. Hence, for each circuit-feature, we need to compute a suitable normalization factor.

Since our objective is to explore the impact of X-sources on test-coverage, we focus on certain structural characteristics of the CUT to serve as potential features that influence the detectability of faults in F_i, and invalidate the detectability of faults that lie in the set UF_i, when an input s_i to the CUT is set to an unknown value. In this chapter, we use the following terminology to define circuit-features:

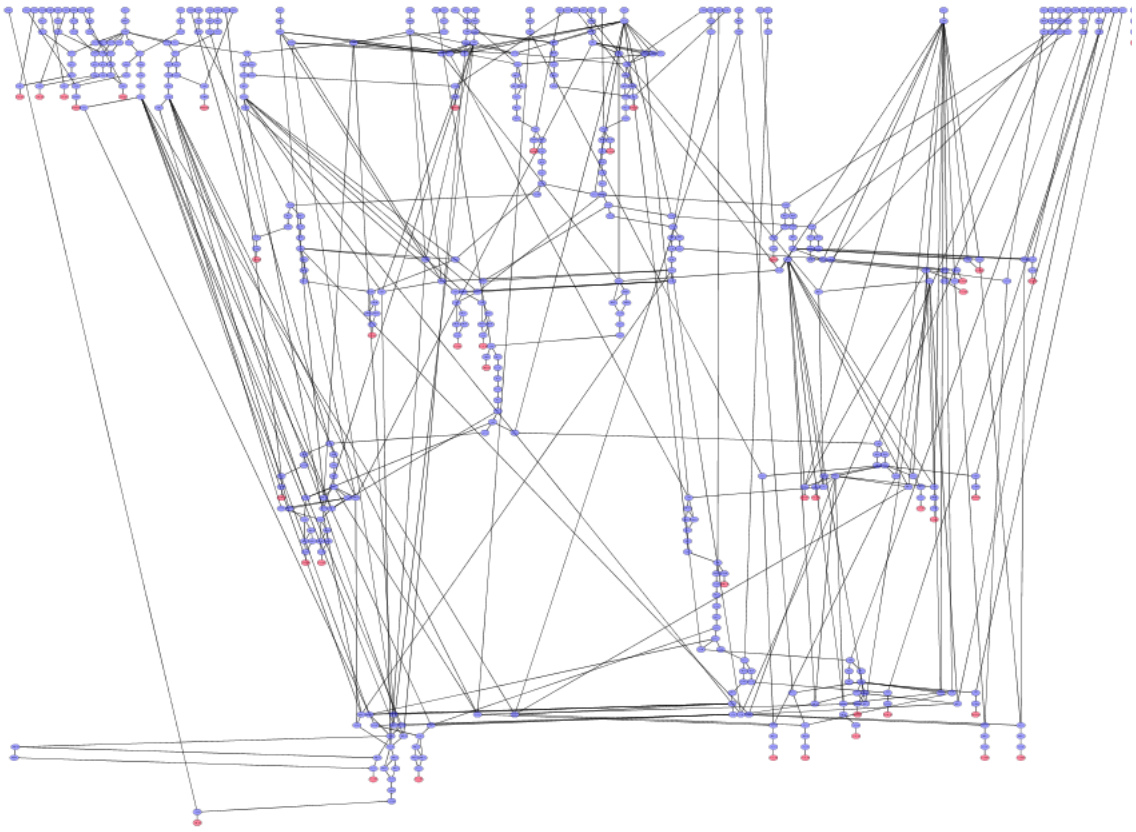
1. *Node.* In a circuit, the inverters, buffers, and the logic gates, 'XOR' and 'XNOR are X-insensitive. That is, they let an X-signal to pass whenever it appears at their input. Hence, such elements may be ignored from the viewpoint of detectability-loss while analyzing a circuit structure. A gate/cell that does not belong to the above category is called a *node*.
2. *Circuit-graph.* Let I, O and LG represent the set of input ports, output ports, and gates of a circuit, respectively. A circuit netlist can be represented by a directed graph $G(E,V)$, where the vertex set $V = \{I \cup LG \cup O\}$ and the edge set E is the set of directed edges representing a signal connection in the circuit between two vertices in V.
3. *X-cone.* The output cone of an X-source in a circuit is called the X-cone of the X-source. Given an X-source of a circuit with circuit-graph G, an X-cone is thus a subgraph of G that can be obtained by exploring G starting from the X-source in the sense of signal reachability.
4. *Level of a vertex.* The level of a vertex v, $level(v)$, of a circuit-graph G is defined as the length of the *longest* path through which it is reachable from any input port. Note that while calculating the path length, only the vertices corresponding to the nodes are considered. The vertex itself is considered in the path if it represents a node. The input ports are assumed to be at level 0. The maximum level of a vertex in

the G is denoted by L. Levelizing a circuit essentially means embedding of the netlist in a topologically sorted order [Kah62], considering the fact that the vertices, which do not represent a node, do not contribute to the level. One such embedding for an ISCAS'89 benchmark circuit, s27, is shown in Fig. 4.1, where the levels are depicted with different shades. Note that in the diagram, two inverters are present and they are in the same level as their preceding vertex.

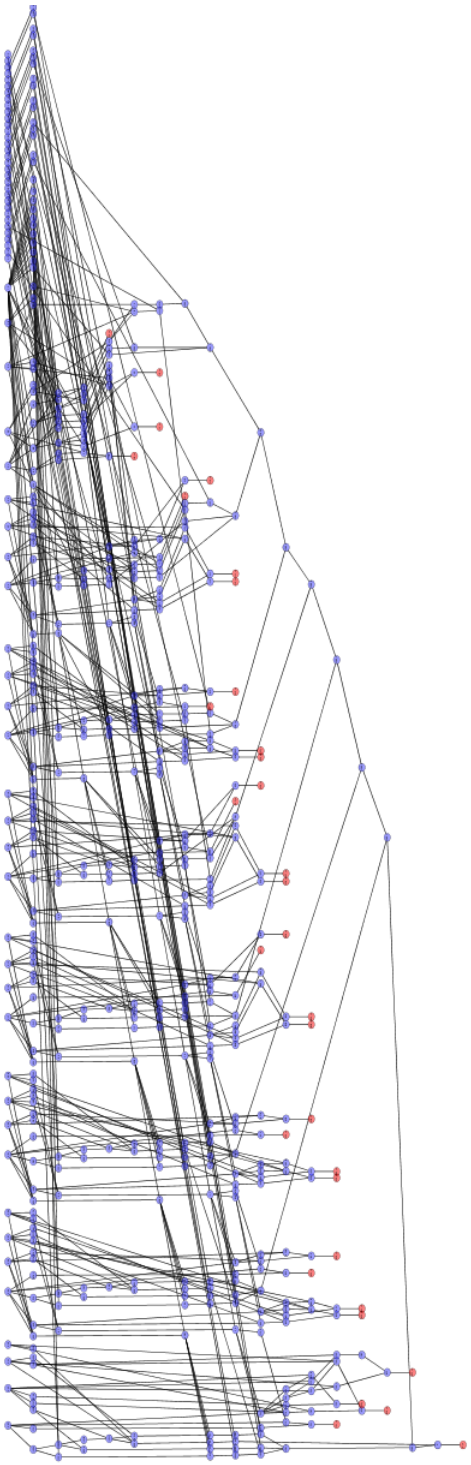
5. *Depth of a vertex.* The depth of a vertex v, $depth(v)$, is defined as the length of the *shortest* path to it from any input port.
6. *X-depth of a vertex.* Quite similar to the above two definitions, the X-depth of a vertex v, $X\text{-}depth(v)$ in an X-cone is defined as the length of the *shortest* path to it from the X-source. While the level and depth of a vertex are defined over all input ports, the X-depth is defined with respect to a particular X-source only.

4.3.1 Illustration of Structural Uniqueness of Circuits and Their Inputs

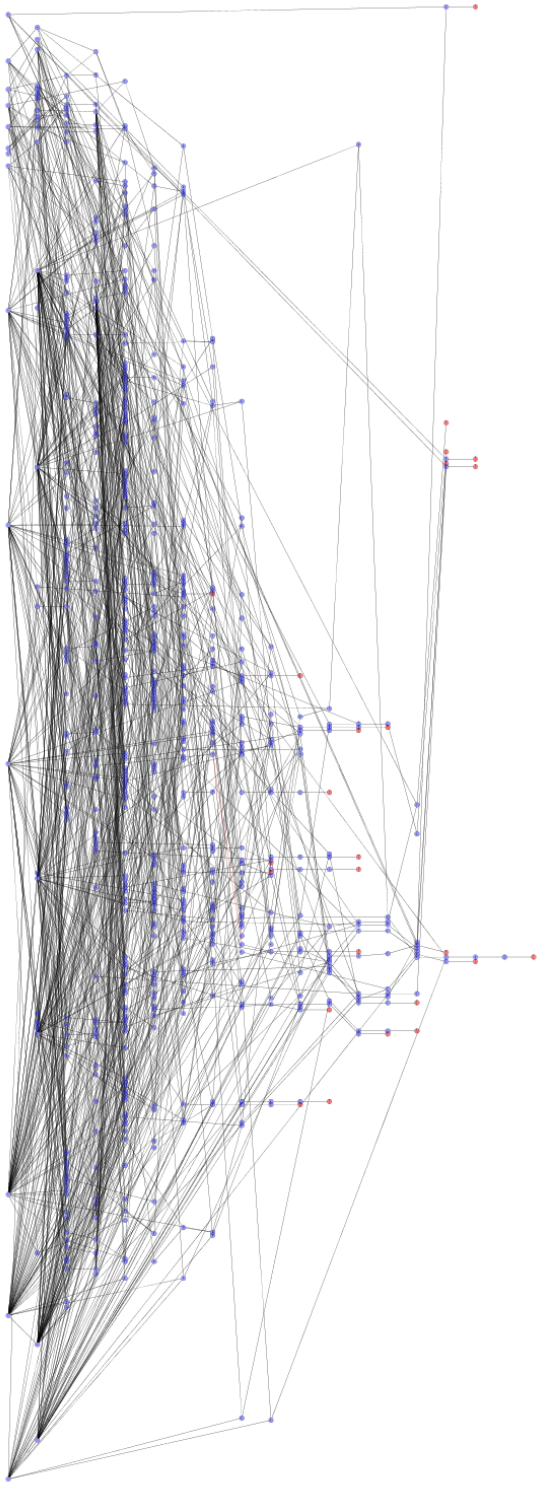
Before we present the proposed structural features, in this section we illustrate a few circuit-graphs of the circuit from ISCAS'89 [BBK89] benchmark suit in order show a visual demonstration the structural uniqueness of the circuits. Figure 4.2 shows the circuit-graphs of s713 (Figure 4.2a), s838 (Figure 4.2b) and s1494 (Figure 4.2c). In order to view the graph we have used a graph visualization tool [SMO+03]. The figures show a hierarchical layout of the circuit-graph, where the vertices belonging to the same graph-level are horizontally aligned. The vertices representing the output ports are marked in red. The rest of the vertices appear in blue. The vertices in the first horizontal alignment are the input vertices (X-sources). Figure 4.2a shows that the circuit-graph of s713 has a heterogeneous structure. The output vertices are scattered in different levels. Each of the X-cone of input vertices are unique. Also, the graph-level seem to be larger compared to the node density of the circuit. As opposed to s713, the circuit-graph of s838, shown in Figure 4.2b, shows a pattern. It has an iterative structure. This affects the X-cone of the inputs also, whose size uniformly varies as we move from the left most to the right most. The third circuit-graph is that of s1494, shown in Figure 4.2c. It can be clearly seen from this graph-layout that this circuit has many high fan-out points, both at the input vertices and at the internal vertices. Thus, we can see that each circuit and each of its X-cones may have a unique structure.



(a) A heterogeneous structure: s713



(b) An iterative structure: s838



(c) High fan-out structure: s1494

Figure 4.2: Circuit-graph of some ISCAS'89 benchmark circuits.

4.3.2 Circuit Features

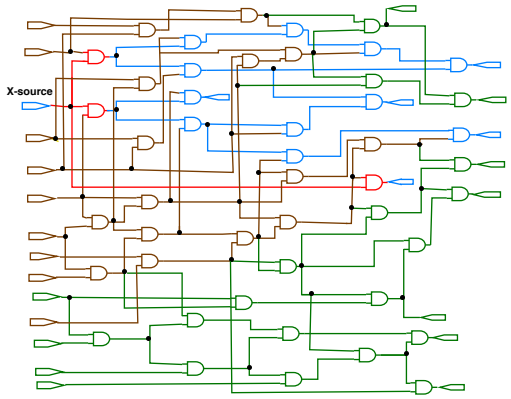


Figure 4.3: A diagram of a hypothetical circuit, where an input port (shown in blue color) is set to an X-value, and the three partitions of the circuit induced by this X-source: $\mathcal{P}_1$ (colored blue), $\mathcal{P}_2$ (colored brown), and $\mathcal{P}_3$ (colored green). The nodes in $\mathcal{P}_1$ that belong to X–depth one ($\mathcal{D}_1$) are shown in red.

Given an X-source s_i in a CUT, we analyze its structural influence by partitioning the circuit into three disjoint sets of vertices with respect to it: (i) Partition $\mathcal{P}_1$ - consists of X-cone of s_i, (ii) Partition $\mathcal{P}_2$ - the sub-circuit that can propagate a signal to the X-cone, and (iii) Partition $\mathcal{P}_3$ - the rest of the circuit. An illustration of partitions of a hypothetical circuit is shown in Fig. 4.3. Note that an X-source can prevent both sensitization and propagation of faults in $\mathcal{P}_1$. On the other hand, it can only prevent propagation of faults in $\mathcal{P}_2$ to the output port in $\mathcal{P}_1$. However, the faults in $\mathcal{P}_2$ can be propagated to the output ports in $\mathcal{P}_3$. Clearly, the X-source will have no effect on $\mathcal{P}_3$. Hence, partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ determine the effect of an X-source on detectability-loss. Moreover, an X-source can degrade the detectability of faults in $\mathcal{P}_1$ in a more predominant fashion than those in $\mathcal{P}_2$. The set of nodes ($\mathcal{D}_1$) in $\mathcal{P}_1$ at X-depth = 1 is shown in red in the figure and is referred to later in this section. For the circuit s27 as shown in Fig. 4.1, we set an X-value to input port G0 and the corresponding three partitions are as follows:

Partition $\mathcal{P}_1$ = {N1, AN1, OR1, OR2, ND1, NR3, N2, NR4, G11, G17, G10};

Partition $\mathcal{P}_2$ = {G1, G7, G6, G3, G5, NR1};

Partition $\mathcal{P}_3$ = {G2, NR2, G13}.

The task of finding suitable circuit parameters involves the discovery of certain structural features that capture the behavioral attributes of $\mathcal{P}_1$ and $\mathcal{P}_2$ as much as possible, particularly for Partition $\mathcal{P}_1$. By analyzing a number of benchmark circuits, we have studied the variation in the partitions within a circuit for different X-sources, and also variations across different circuits, and based on that, we have proposed the following set of empirical

circuit-parameters. Each of these parameters is illustrated with reference to the X-input G0 of s27, although all the proposed parameters may not be applicable for this circuit. Let the set of nodes and the set of output ports in $\mathcal{P}_1$ be denoted by $\mathcal{P}_{1_{nodes}}$ and $\mathcal{P}_{1_{op}}$, respectively.

Number of nodes in $\mathcal{P}_1$ ($n\mathcal{P}_{1_{nodes}}$)

The normalization factor for this parameter is chosen as the total number of nodes in the circuit. Thus, the normalized parameter indicates the proportion of nodes in $\mathcal{P}_1$, i.e, the relative size of $\mathcal{P}_1$ with respect to the size of the CUT. Such normalization also serves an additional purpose of providing a relative measure of the size of $\mathcal{P}_1$ with respect to the different partitions. Note that the size of $\mathcal{P}_1$ may not always increase linearly with `DT-Loss`, because this would mean that each of the X-sources will have similar effect on the nodes in the circuit. Most circuits have X-sources that have many structural variations in their X-cones and will be taken care by other parameters explained below. Nonetheless, this parameter gives us an estimate of the size of partition $\mathcal{P}_1$, and a small size would generally mean small `DT-Loss`.

Example: $n\mathcal{P}_{1_{nodes}}$ for G0: In circuit s27, there are ten logic gates. Out of them, two are 'NOT' gates, so the number of nodes in the circuit is eight, which is the normalization factor. Similarly, the number of nodes in partition $\mathcal{P}_1$ is 6. So, the value of the normalized parameter for input G0 is $6/8 = 0.75$.

Number ($n_{\mathcal{D}_1}$) of nodes in X-depth-1 ($\mathcal{D}_1$)

This gives the number of fanouts from an X-source. If any inverter or 'XOR'/'XNOR' gate or buffer is adjacent to the X-source, then the number fanouts from such a gate is also counted. This gives us a measure of the extent to which an X-source directly affects its X-cone. A large value of $n_{\mathcal{D}_1}$ in general, suggests a large value of `DT-Loss`. In this case, we choose the total number of input ports as the normalization factor. The next two parameters add more information to this feature in terms of the levels of nodes in $n_{\mathcal{D}_1}$.

Example: $n_{\mathcal{D}_1}$ for G0: The nodes in the set $\mathcal{D}_1$ is AN1 and NR4. The total number of input ports in the circuit is 7. So, the normalized parameter for G0 is $2/7 = 0.286$.

Average level ($A_{\mathcal{D}_1}$) of nodes in X-depth-1 ($\mathcal{D}_1$)

The normalization factor for this parameter is chosen as the average level of the output ports in the X-cone.

Example: $A_{\mathcal{D}_1}$ for G0: The level of gate AN1 is 1 and of gate NR4 is 5, so the average level is $(1+5)/2 = 3$. The average level of output ports in the X-cone is $(4+4+5)/3 = 4.33$. So the normalized parameter for G0 is $3/4.33 = 0.69$.

Maximum level ($M_{\mathcal{D}_1}$) of the nodes in X-depth-1 ($\mathcal{D}_1$)

The normalization factor is chosen as the maximum level among the output ports belonging to in the X-cone of the concerned input.

Example: $M_{\mathcal{D}_1}$ for G0: The maximum level among gates AN1 and NR4 is 5 and the maximum level of a gate in s27 is 5. So the normalized parameter for G0 is $5/5 = 1$.

Number of outputs ports ($n\mathcal{P}1_{op}$) in $\mathcal{P}_1$

The normalization factor is chosen as the number of output ports in the circuit. Thus, the normalized parameter gives an idea of the maximum fraction of output ports that could be rendered unobservable because of an X-value at this input.

Example: $n\mathcal{P}1_{op}$ of G0: The total number of output ports in the circuit is 4 and the number of output ports in X-cone(G0) is 3. Thus, $n\mathcal{P}1_{op} = 3/4 = 0.75$.

Normalized sum of levels ($N\mathcal{P}1_{op}$) of the output ports in an X-cone

The levels of different output ports play an important role in determining the loss in detectability. Note that an output port lying at a low level is likely to contribute less in the size of X-cone or Partition $\mathcal{P}_2$. The normalized parameter is defined as,

$$N\mathcal{P}1_{op} = \frac{\sum_{v \in P1_{op}} level(v)}{\sum_{u \in \mathcal{M}} level(u)} \tag{4.2}$$

where $\mathcal{M}$ is a subset of the set of output ports of the circuit such that $|\mathcal{M}| = n\mathcal{P}1_{op}$ and among all the output ports in the circuit, the output ports $\in \mathcal{M}$ have the highest levels.

Example: $N\mathcal{P}1_{op}$ for G0: Among the four output ports in the circuit, the output ports lying at the top 3 levels are same as the output ports in the X-cone. So the parameter is given by $(4+4+5)/(4+4+5) = 1$.

X-depth-level ratio (DLRO) of output ports in an X-cone

$$DLRO = \frac{\sum_{v \in \mathcal{P}1_{op}} X\text{-}depth(v)/level(v)}{n\mathcal{P}1_{op}} \tag{4.3}$$

Empirical evidence shows that this parameter is useful in leveraging the effect of variation in the structure of different circuits when samples from these circuits are drawn to form a training set, particularly the variation observed in the level (L) of the circuits. This parameter is implicitly normalized.

DLRO for G0: The X-depth of the three output ports G11, G17 and G10 is 4, 4, and 1, respectively, and level is 4, 4 and 5, respectively. Hence, $DLRO = (4/4 + 4/4 + 1/5)/3 = 2.2/3 = 0.73$.

Number of input ports ($n_I\mathcal{P}_1$) influencing an X-cone

It is expressed as the total number of input ports in $\mathcal{P}_2$. The normalization factor is the total number of input ports in the circuit (n_I). Hence, it is a reflection of the size of partition $\mathcal{P}_1 \cup \mathcal{P}_2$.

Example: $n_I\mathcal{P}_1$ for G0: Five out of the seven input ports feed X-cone(G0), so the normalized parameter is $5/7 = 0.71$

Average ratio of depth-to-X-depth ($C_o\mathcal{P}_2$)

This parameter is given by the expression

$$C_o\mathcal{P}_2 = \frac{\sum_{v \in \mathcal{P}_{1_{nodes}}} depth(v)/X\text{-}depth(v)}{n\mathcal{P}_{1_{nodes}}}, \tag{4.4}$$

It gives a measure of the controllability due to input ports in $\mathcal{P}_2$ on the nodes in $\mathcal{P}_1$. For a node v, the ratio $depth(v)/X\text{-}depth(v)$ is 1 when an X-source reaches the node through a path shorter than any input in $\mathcal{P}_2$, otherwise the ratio is smaller than 1. $C_o\mathcal{P}_2$ gives the average ratio among all the nodes. A small the value of $C_o\mathcal{P}_2$ implies high controllability of the input ports possibly leading to small `DT-Loss`%. This parameter is implicitly normalized.

Example: $C_o\mathcal{P}_2$ for G0: For the nodes in the X-cone of G0, the depth and X-depth values are given in the Table 4.2 below. From this the value of $C_o\mathcal{P}_2$ is computed to be $(1/1 + 2/2 + 1/2 + 2/3 + 1/4 + 1/1)/6 = 0.736$.

Table 4.2: Depth and X-depth of nodes in X-cone of input G0 of s27

Node	AN1	OR1	OR2	ND1	NR3	NR4
depth	1	2	1	2	1	1
X-depth	1	2	2	3	4	1

Influence of $\mathcal{P}_2$ on the X-cone ($l_{\mathcal{P}_2}$)

Let the total number of gate-input ports of the nodes in $\mathcal{P}_1$ be given by n_{gip}. Among them, let l_{P_2} be the number of lines from $\mathcal{P}_2$ that feed the nodes in $\mathcal{P}_1$. So, the influence of $\mathcal{P}_2$ on the X-cone can be quantified using l_{P_2}, and the normalization factor is chosen as n_{gip}. A small normalized value of l_{P_2} for a large-size $\mathcal{P}_1$ implies that the X-cone has huge reconvergent lines, and the effect of X-signal is reachable up to greater depths of the circuit.

Example: $l_{\mathcal{P}_2}$ for G0: There are six nodes in the X-cone and each of them has two input ports, so the total number of gate-input ports is 12. Among them, one input of AN1,

OR1, OR2 and NR3 each, is fed from outside the X-cone. So, the normalized parameter is $4/12 = 0.33$.

Maximum X-depth ($M\mathcal{P}_1$) among the nodes in an X-cone

The influence of an X-source is likely to decrease with the X-depth of a node. An X-cone with large maximum depth could imply that the `DT-Loss` would be small even if the size of $\mathcal{P}_1$ is large. We do not normalize this parameter because, firstly, this is mostly independent of the size of the circuit in terms of the number of gates in the CUT, and secondly, this parameter does not vary widely for different circuits.

Example: $M\mathcal{P}_1$ for G0: Maximum X-depth among the nodes in the X-cone is that of gate NR3, the value of which is 4.

Categorical feature (CF)

It is observed that some circuits netlist possess an iterative structure. Hence, the X-cones of these circuits might have similar structures even though of different size. The `DT-Loss` due to the presence of an X-source is affected differently due to their iterative structure. Hence, it is important to discriminate such circuits from the rest. We add a categorical binary feature-flag to represent them. Categorical features are generally encoded as dummy variables in the feature set; all sample data that represent X-sources taken from a iterative circuit is encoded as 0 for this feature, and for the rest of sample data, it is encoded as 1. The information of whether or not a circuit has an iterative structure is assumed to be known.

Table 4.3: Summary of the parameters

No.	Name	Description
1	$n\mathcal{P}_{1_{nodes}}$	Number of nodes in $\mathcal{P}_1$.
2	$n_{\mathcal{D}_1}$	Number of nodes in X-depth-1.
3	$A_{\mathcal{D}_1}$	Average level of nodes in X-depth-1.
4	$M_{\mathcal{D}_1}$	Maximum level among nodes in X-depth-1.
5	$n\mathcal{P}_{1_{op}}$	Number of output ports in $\mathcal{P}_1$.
6	$N\mathcal{P}_{1_{op}}$	Normalized sum of levels of output ports in $\mathcal{P}_1$.
7	$DLRO$	Average X-depth-to-level ratio of output ports in $\mathcal{P}_1$.
8	$n_I\mathcal{P}_1$	Number of input ports feeding $\mathcal{P}_1$.
9	$C_o\mathcal{P}_2$	Average depth-to-X-depth ratio of nodes in $\mathcal{P}_1$.
10	l_{P_2}	Number of lines from $\mathcal{P}_2$ feeding the nodes in $\mathcal{P}_1$.
11	$M\mathcal{P}_1$	Maximum X-depth among the nodes in $\mathcal{P}_1$.
12	CF	Categorical binary variable

Example: Since s27 does not have an iterative structure, the value of this parameter is 1 for G0.

A brief description of the parameters (features) is given in Table 4.3. Note that all these features are based exclusively on the structural information of the circuit. No functional information of the netlist such as gate type, is utilized for defining features.

4.3.3 Algorithms for Feature Computation

Algorithm 3 computes the level and depth of the vertices in a circuit-graph. Thereafter, the features for each X-source are computed using Algorithm 4. These algorithms are based on the traversal techniques implicit in Kahns algorithm [Kah62] and breadth-first-search (BFS) [CSRL01] (starting from the X-source) respectively. Slight modifications of these algorithms are needed as in our case, a circuit-graph may contain some vertices (called non-nodes), which do not effectively contribute to the count of level, depth or X-depth of the nodes. The normalization factor of $n\mathcal{P}1_{nodes}$ (the total nodes in the circuit) can also be determined by Algorithm 3.

Algorithm 3 : Computing level and depth of vertices of a circuit-graph *G(E,V)*

```
1: Initialize level(v) ← 0 ∀v ∈ V and depth(v) ← ∞ ∀v ∈ V \ I
2: Initialize depth(v) ← 0 ∀v ∈ I
3: Enqueue(v) on queue Q1 ∀v ∈ I
4: while Q1 not Empty do
5:     v ← Dequeue(Q1)
6:     for Each children w of v do
7:         Remove edge {v → w}
8:         if w is a node then
9:             level(w) ← max(level(w), level(v) + 1)
10:            depth(w) ← min(depth(w), depth(v) + 1)
11:        else
12:            level(w) ← max(level(w), level(v))
13:            depth(w) ← min(depth(w), depth(v))
14:        end if
15:        if indegree of w is 0 then
16:            Enqueue(w)
17:        end if
18:    end for
19: end while
20: Restore edges
```

While computing $n_I\mathcal{P}_1$, for each vertex representing an output port, an input-port-list is maintained. The input-port-list is a list containing those input ports from which the vertex is reachable. This list is also computed during the execution of Algorithm 3. An input-port-lists is maintained for each vertex, and is updated each time it is discovered through a new parent vertex during the traversal. These lists can be deleted once all its

children have been updated. $n_I\mathcal{P}_1$ is computed by finding the union of input-port-lists of the output ports in the X-cone.

For calculation of parameter $l_{\mathcal{P}_2}$, for each node v, let $l(v)$ be the number of inputs from $\mathcal{P}_2$ feeding v, $l(v)_1$ be number of times it is discovered while executing Algorithm 3, and $l(v)_2$ be number of times it is discovered while executing in Algorithm 4. Then $l(v) = l(v)_1 - l(v)_2$ and $l_{\mathcal{P}_2} = \sum_{v \in \mathcal{P}_{1_{nodes}}} l(v)$. The normalization factor is computed as $n_{gip} = \sum_{v \in \mathcal{P}_{1_{nodes}}} l(v)_1$.

Algorithm 4 is based on a variant of BFS and can be used to find the X-depth of each vertex. Note that as the X-cone is traversed by Algorithm 4, the sets $\mathcal{P}_{1_{nodes}}$, $\mathcal{P}_{1_{op}}$ and $\mathcal{D}_1$ are obtained. So, the features can be easily computed.

Algorithm 4 : Computing X-depth of vertices in the X-cone of an X-source s_i

```
1: Enqueue(s_i) on queue Q_2
2: X-depth(s_i) ← 0
3: while Q_2 not Empty do
4:     q ← Dequeue(Q_2)
5:     SUB_BFS(q, Q_2)
6: end while
7: procedure SUB_BFS(q, Q_2)
8:     for Each children r of q do
9:         if r is not visited then
10:            Visit r
11:            if r is a node then
12:                X-depth(r) ← X-depth(q) + 1
13:                Enqueue(r)
14:            else
15:                X-depth(r) ← X-depth(q)
16:                SUB_BFS(r, Q_2)
17:            end if
18:        end if
19:    end for
20: end procedure
```

We modify the BFS-algorithm by incorporating a sub-function called SUB_BFS as given in Algorithm 4. A non-node vertex is never enqueued. Instead, a new SUB_BFS traversal is invoked when a non-node vertex is visited, and stops only if the visited vertex is a node, and the enqueue operation is executed accordingly.

Table 4.4: Features for s27

	parameter \ port	G0	G1	G2	G3	G5	G6	G7
1	$n\mathcal{P}1_{nodes}$	6/8	6/8	1/8	4/8	2/8	6/8	6/8
2	$n_{\mathcal{D}_1}$	2/7	1/7	1/7	1/7	1/7	1/7	1/7
3	$A_{\mathcal{D}_1}$	3/4.33	1/3.75	2/2	2/4.3	4/4.3	1/4.3	1/3.75
4	$M_{\mathcal{D}_1}$	5/5	1/5	2/2	2/5	4/5	1/5	1/5
5	$n\mathcal{P}1_{op}$	3/4	4/4	1/4	3/4	3/4	3/4	4/4
6	$N\mathcal{P}1_{op}$	1	1	0.4	1	1	1	1
7	$DLRO$	0.73	1	0.5	0.77	0.3	1	1
8	$n_I\mathcal{P}_1$	5/7	6/7	2/7	5/7	5/7	5/7	6/7
9	$C_o\mathcal{P}_2$	0.736	0.602	1	0.645	0.75	0.602	0.602
10	l_{P_2}	4/12	6/12	1/2	4/8	2/4	5/12	6/12
11	$M\mathcal{P}_1$	4	5	1	4	2	5	5
12	CF	1	1	1	1	1	1	1

Time Complexity: The time complexity for computing the features from an X-source is $\mathcal{O}(|E| + |V|)$. Note that given a circuit and its X-sources, the full circuit needs to be traversed only once to calculate the level and depth. Once the levels and depths of all the vertices are computed, only the X-cone needs to be traversed, for each X-source.

The various features for the circuit s27 are given in Table 4.4. Each entry is of the form (i/j) where i denotes the feature value and j denotes its normalization factor.

4.4 Support Vector Regression (SVR)

In this section, we set up the problem of predicting the `DT-Loss` of an X-input in a circuit, using the features extracted in the previous section, as a regression problem [DCM12]. To this end, suppose our training data has N inputs (obtained by pooling together all the inputs from the different circuits), and denote by y_i the value of `DT-Loss` of the i-th input, and let $x_i \in \mathbb{R}^p$ denote the vector of features corresponding to that input. In our case, $p = 12$, corresponding to 11 numerical features and 1 categorical binary feature, which is 0 or 1 depending on whether the circuit has an iterative structure or not, respectively (see Table 4.3). Thus, we will have a 12-dimensional feature space. In regression analysis y_i is related to a function of x_i as follows: $y_i = f(x_i) + \varepsilon_i$, where $f : \mathbb{R}^p \to \mathbb{R}$ is some unknown function and $\{\varepsilon_i : 1 \leq i \leq N\}$ are independent mean-zero random variables, which reflect measurement errors [CH06]. In general, the unknown function f is estimated by minimizing some loss-function on the training data $\{(x_i, y_i) : 1 \leq i \leq N\}$, over a class of functions.

The simplest choice is to assume that the function f is linear $f(x) = x^T\beta + \beta_0$. This is the classical multiple regression model, where the parameters β and β_0 are estimated by minimizing the error in sum-of-squares. Even though multiple regression is very easy to

implement and the estimated coefficient $\hat{\beta}$ has a natural interpretability, the relationship between the `DT-Loss` of an input and its associated features is quite complex, and a simple linear function is not enough to capture this dependency. To address this issue and to improve our predictive performance, we use a *support vector regression* (SVR) model, which a powerful machine-learning method commonly used to model non-linear dependencies between variables [SS04]. SVR is an extension of the well-known binary classifier *support vector machines* (SVM) adapted for regression with a qualitative response. In the following, we briefly discuss the popular variant ε-SVR [Vap95, SS04], which fits the function $f(x) = x^T\beta + \beta_0$, where β and β_0 are chosen according to the following optimization problem (given tuning parameters C and ε),

$$\begin{aligned} &\min_{\beta,\beta_0,\zeta,\zeta'} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}(\zeta_i + \zeta_i') \\ &\text{subject to} \begin{cases} y_i - x^T\beta - \beta_0 \leq \varepsilon + \zeta_i, \\ x^T\beta + \beta_0 - y_i \leq \varepsilon + \zeta_i', \\ \zeta_i, \zeta_i' \geq 0, \text{ for } 1 \leq i \leq N. \end{cases} \end{aligned} \tag{4.5}$$

Note that by the introduction of the tuning parameter ε, we ignore the points with residuals $|y_i - x^T\beta - \beta_0|$, which are less than ε. The dual formulation of (4.5) is given by (see [SS04] for details)

$$\begin{aligned} &\max_{\alpha,\alpha'} \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')x_j^T x_i \\ -\varepsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i') + \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i') \end{cases} \\ &\text{subject to} \quad \sum_{i}^{N}(\alpha_i - \alpha_i') = 0 \quad \text{and} \quad \alpha_i, \alpha_i' \in [0, C], \end{aligned} \tag{4.6}$$

where $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_N)$ and $\alpha' = (\alpha_1', \alpha_2', \ldots, \alpha_N')$ are the dual variables.

The SVR procedure described above can be easily extended to include non-linear functions, by enlarging the feature space using basis functions. Once a set of basis functions $h = (h_1, h_2, \ldots, h_m)$ (these correspond to different transformations of the selected features) is chosen, we can fit an SVR using the input features $h(x_1), h(x_2), \ldots, h(x_N)$. This fits a non-linear function $f(x) = h(x)^T\beta + \beta_0$, where the parameters β and β_0 are chosen by solving the optimization problem (4.5) on the transformed data. Note that (4.5) involves the function $h(x)$ only through inner product $h(x)^T h(y)$, that is, to fit a non-linear function using SVR, we only require the knowledge of the kernel function $K(x, y) = h(x)^T h(y)$. Therefore, given a general kernel function $K(x, y)$, the parameters β and β_0 in the SVR

with kernel K, are obtained by solving the following optimization problem:

$$\max_{\alpha,\alpha'} \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \alpha'_i)(\alpha_j - \alpha'_j)K(x_i, x_j) \\ -\varepsilon\sum_{i=1}^{N}(\alpha_i + \alpha'_i) + \sum_{i=1}^{N} y_i(\alpha_i - \alpha'_i) \end{cases} \quad \text{subject to} \quad \sum_{i}^{N}(\alpha_i - \alpha'_i) = 0 \quad \text{and} \quad \alpha_i, \alpha'_i \in [0, C]. \tag{4.7}$$

We will apply (4.7), with the commonly used Gaussian kernel $K(x, y) = \exp(-\gamma||x - y||^2)$, for predicting the `DT-Loss` of an X-input based on the selected features (see Section 4.5 below for details). The Gaussian kernel is translation invariant, and defines a function space, which is much larger than linear or polynomial kernels. The parameter γ can be used to control the influence of a single training sample in the model, with low values meaning 'far', and high values meaning 'close'.

4.5 Data Analysis and Methodology

We consider a number of circuits from the ISCAS'89 and ITC'99 benchmark suites for preparing our dataset. A sample point in the dataset represents an input (primary or pseudo) of a circuit belonging to it. A preliminary analysis of the circuits reveals that they vary widely in their sizes and structural characteristics. In order to maintain some uniformity and avoid issues with outliers and skewness, we prune a few circuits that are very small or very large. Big-size circuits contribute a large number of sample points, and for some of them, most of the inputs cause very small DT-loss when set to X-value; they tend to add only skewness to the data. Following the pruning step, we consider a subset of circuits ranging from s349 (smallest) to b22s (largest), and we divide them into the training set and the test set as follows: the samples in the test set should conform to some representative data in the training set. Conventionally, the sample points in a dataset are randomly split into training and test sets to ensure fair representation. In our case, we need to grade the X-sources of a circuit, and hence, we need to select circuits rather than the X-sources. To ensure fair representation of the circuit that are chosen for the SVR test set, in the training set, we perform the following classification based on the functionality of ISCAS89 circuits: (a) Category 1 (multiplier): s344, s349, s420, s838, (b) Category 2 (controllers): s382, s386, s400, s444, s510, s526, s953, s1488, s1494, (c) Category 3 (PLD devices): s641, s713, s820, s832, (d) Category 4 (combinational circuits with randomly inserted flip-flops): s1196, s1238, (e) Category 5 (real chip-based circuits): s9234, s13207, s15850, s38417, s38584, and (f) Category 6: (function not classified): s35932. Among them, some circuits (s344, s382, s641, s820, s1196, s1488) are re-synthesized versions of some other circuits (s349, s400, s713, s832, s1238, s1494) obtained after removing redundancy. Finally, we consider some of the circuits in the ITC99 benchmark suite as Category 7, the largest among the data set being

b22s. From each category, we consider a representative circuit of large size in our test set. To summarize, the training and test set consist of inputs from the following circuits:

Training set: {s344, s349, s382, s386, s400, s444, s420, s510, s526, s641, s820, s832, s953, s1196, s1423, s1488, s5378, s9234, s13207, s15850, s38417, b03, b04s, b05s, b06, b07s, b08, b09, b10, b11s, b12, b13s, b14s, b15s, b20s, b21s}

Test set: {s713, s838, s1238, s1494, s35932, s38584, b22s}.

The `DT-Loss` for large circuits is found to be small for most of the X-sources. As a result, the distribution of `DT-Loss` in the population is observed to be very skewed, and thus the performance of the regression model becomes sensitive to the choice of the function f. For example, multiple linear regression was unable to accurately predict the `DT-Loss` for most of the test circuits (see Table 4.5 in the next section). This motivates us to use non-linear regression functions based on more sophisticated machine-learning techniques. In our dataset, we use the support vector regression (SVR) method [SS04], using the Gaussian kernel, as explained in the previous section. Recall, in the notation of Section 4.4, the training data is of the form $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$, where y_i is the value of the `DT-Loss` of an X-input, and $x_i \in \mathbb{R}^{12}$ is the corresponding feature vector in the 12-dimensional feature space. Note that this model has three tuning parameters C, ε, γ, which control the generalizability of SVR [CM04]; the parameter ε determines the maximum deviation that is allowed between the predicted and true value of any point, the parameter C controls the trade-off between the amount to which deviations greater than ε is permitted and the model complexity in (4.5), and the kernel parameter γ controls the reach of a single point in the training set. The selection of these parameters based is on the minimum error in the training set leads to the well-known problem of *over-fitting*. This is avoided by adopting a technique called K-fold *cross-validation* [HTF09, Chapter 7], where the training data is divided into K equal parts/folds, and then training is performed on $K - 1$ folds, whereas testing is executed on the remaining fold. This process is repeated K times where each fold is considered in the test set once, and the parameters are chosen by minimizing the average test error of the K test errors. In our data set, we have used 3-fold cross-validation to choose the parameters, using the mean-squared-error to measure test errors. The parameters for the SVR model in our dataset are selected as: $\varepsilon = 0.01$, $C = 100$ and $\gamma = 0.1$.

4.6 Experimental Results

The experiments were carried out on an Intel Xeon 3.00-GHz $\times$ 4 processor with 8GB memory. We used Synopsys TetraMAX [J-2] as the ATPG tool for computing the `DT-Loss`. For the construction and manipulation of graphs, we used the Python module, graph-tool [Pei14], which implemented in C++. We used scikit-learn [ea11] as the machine learning tool-kit. For each circuit in the training set as well as in the test set, all the input ports are

considered as X-sources. However, in training set, the input ports that have similar values for true `DT-Loss` and parameter values are removed to make sure that each sample point is unique.

4.6.1 Goodness-of-Fit

The first step to assessing the performance of the SVR model is to check whether it fits the training data well. To this end, let $\hat{f}(x)$ be the regression function estimated by the SVR method, and $\hat{y}_i = \hat{f}(x_i)$ the `DT-Loss` predicted by the model for the i^{th} input point in the training set. Therefore, $y_i - \hat{y}_i$ is the difference between the actual and the predicted `DT-Loss`. It is common to use the *coefficient of determination* (CD), which is a relative measure of goodness-of-fit obtained by comparing the squared-RMSE (Root Mean Squared Error) with total variance of the training data: $CD = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$, where $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$ is the mean of the `DT-Loss` in the training set. Note that $CD \leq 1$ by definition (in case of multiple regression CD is equal to the well-known multiple R^2 [CH06]), and a value of CD close to 1 indicates a good fit.

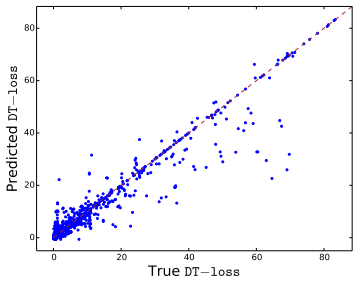


Figure 4.4: Scatter plot showing SVR-predicted against true `DT-Loss` for X-inputs in the training set.. The number of sample points is 3898.

The CD value for the SVR is 0.96, which indicates that this fits the training data well, and far better than multiple regression, which has CD = 0.62. Fig. 4.4 shows a scatter plot of the predicted values of the SVR against the actual values of the `DT-loss` in the training set. The horizontal axis represents the actual and the vertical axis represents the predicted `DT-loss`, the points plotted are $(y_1, \hat{y}_1), (y_1, \hat{y}_2), \ldots, (y_N, \hat{y}_N)$, and the dotted red diagonal line represents the equation $y = x$, that is, the points where predicted values are same as the true values. We observe that most of the points are in the neighborhood of the diagonal line, indicating that the SVR model fits well to the training data. The CPU-times for training multiple regression and SVR are 0.08 seconds and 11.22 seconds, respectively.

4.6.2 Predictive Performance

Recall that we have $n = 7$ circuits in the test set. Suppose the j-th circuit (where $1 \leq j \leq 7$) has n_j possible inputs, with `DT-Loss` $y_{j1}, \ldots, y_{jn_j}$, and the corresponding feature vectors $x_{j1}, \ldots, x_{jn_j}$. To measure how well our SVR model predicts the `DT-Loss` of the j-th circuit we define the (predictive) CD of the j-th circuit as follows:

$$CD_j = 1 - \frac{\sum_{i=1}^{n_j}(y_{ji} - \hat{y}_{ji})^2}{\sum_{i=1}^{n_j}(y_{ji} - \bar{y}_j)^2},$$

where $y_{ji} = \hat{f}(x_{ji})$ is the predicted value, and $\bar{y}_j = \frac{1}{n_j}\sum_{i=1}^{n_j} y_{ji}$ the mean of the actual `DT-Loss` of the inputs of the j-th circuit. As before, the predictive CD is scale independent, and, therefore, can be used to compare the results for different test circuits regardless of the range of the `DT-Loss` for each circuit.

Table 4.5: Predictive performance of the regressor

Circuit	Linear Regressor	SVR
s713	0.28	0.42
s838	0.48	0.94
s1238	0.64	0.90
s1494	0.21	0.99
s35932	0.63	0.85
s38584	0.75	0.83
b22s	-0.16	0.81

The CD values for the different circuits are shown in Table 4.5. As expected, the SVR has better CD than the multiple regression for all the circuits. In particular, for circuits s1494 and b22s we see marked improvement of the CD values from 0.21 to 0.99 and -0.16 to 0.81, respectively. Moreover, the CD values for SVR of all circuits (expect s713) is more than 0.8 which indicates that SVR is able to effectively predict the `DT-Loss` values in these circuits. (Possible reasons for inaccurate predictions in circuit s713 are discussed in Section 4.6.4.) The scatter plot of the values predicted by the SVR model versus the actual values, for all the 7 circuits in the test set, are shown in Fig. 4.5. We will discuss these results, together with how well the SVR model predicts the ordering of the `DT-Loss`, in detail in Section 4.6.4.

4.6.3 A Metric for Evaluating X-Source Grading

As discussed in section 4.1, grading of the X-sources based on their X-sensitivity i.e, ranking them in the decreasing order of `DT-loss`, has many applications. However, the `DT-loss`

for some X-sources may be very close to each other and hence grading them may not be so useful compared to other X-sources of the same circuit with high `DT-Loss`. Hence, we consider the top 10 X-sources in terms of `DT-Loss` to demonstrate the efficiency of our ranking scheme. The true `DT-Loss` of a set of X-sources yield a particular grading (a ranking from the largest to the smallest based on `DT-Loss`) of the X-sources. Similarly, the predicted `DT-Loss` yield another grading of the X-sources. If these two gradings are the same, then we say that the model could grade all the X-sources correctly, even though the model might not predict the actual value of the `DT-loss` so accurately. In general, these two gradings will not be identical, and to measure how well a grading is correctly predicted by the model, we define a notion of distance between gradings. We estimate this based on the well-known *Kendall's tau distance* [FKS03, DG77], which is a measure of disarray between two permutations, as defined below: Given two rankings/permutations σ_1 and σ_2 of a set of elements $S = \{l_1, l_2, \ldots, l_m\}$, the Kendall's tau distance is the fraction of pairs, $(l_i, l_j), i < j$, of elements of S that appear in opposite orders in the two rankings, i.e., either $\sigma_1(l_i) > \sigma_1(l_j) \wedge \sigma_2(l_i) < \sigma_2(l_j)$ or $\sigma_1(l_i) < \sigma_1(l_j) \wedge \sigma_2(l_i) > \sigma_2(l_j)$.

Given a circuit $j \in \{1, 2, \ldots, 7\}$ in the test set with b_j different X-sources, denote by $S_j = \{s_1, s_2, \ldots, s_{b_j}\}$ the set of its X-sources. Let $\eta(s_i)$ and $\hat{\eta}(s_i)$ be the true and predicted (using SVR) values of the `DT-Loss` for the X-source $s_i \in S_j$, respectively. To evaluate the how well the model predicts grading of the X-sources among the highest `DT-Loss`, we consider the top M (we use $M = 10$ in our experiments) values from each of the sets $\{\eta(s_1), \eta(s_2), \ldots, \eta(s_{b_j})\}$ and $\{\hat{\eta}(s_1), \hat{\eta}(s_2), \ldots, \hat{\eta}(s_{b_j})\}$. However, the `DT-Loss` in these two sets may correspond to different X-sources, so we cannot directly compare the rankings using the *Kendall's tau distance*. To circumvent this issue, we combine the X-sources from these two sets to make them identical, and then compute the distance between the actual and predicted grading. This is explained in the steps below:

1. Compute the set $S_{(M)}$ of X-sources with the top M actual `DT-Loss`, that is, X-sources corresponding to the largest M values from the set $\{\eta(s_1), \eta(s_2), \ldots, \eta(s_{b_j})\}$.
2. Compute the set $\hat{S}_{(M)}$ of X-sources with top M predicted `DT-Loss`.
3. Define $S_M = S_{(M)} \bigcup \hat{S}_{(M)}$.
4. Compute set $\mathcal{T}$ of all pairs (s_i, s_j), with $s_i, s_j \in S_M$, such that
   - $\eta(s_i) > \eta(s_j)$ and $\hat{\eta}(s_i) < \hat{\eta}(s_j)$ or $\eta(s_i) < \eta(s_j)$ and $\hat{\eta}(s_i) > \hat{\eta}(s_j)$, and
   - $|\eta(s_i) - \eta(s_j)| > \delta$.
5. Define the *δ-Kendall's tau distance* (between the actual and the predictive orderings) as the proportion of such pairs, that is, $\tau_\delta := \frac{|\mathcal{T}|}{\binom{|S_M|}{2}}$.

Choosing $\delta = 0$ corresponds to the actual Kendall's tau distance (which simply counts the number of pairwise mismatches) between the actual and predicted gradings of the X-sources in the set S_M defined above. We introduce the tuning parameter δ, which ignores a pairwise mis-grading when the difference between the actual `DT-Loss` of the pair is close (less

than δ). Therefore, by definition, τ_δ lies between 0 and 1, and τ_δ decreases as δ increases. In Fig. 4.5 we show a *grade plot* for the each circuit in the test set, which plots the values τ_δ, as δ is varied from 0 to 10. Finally, note that the presence of one or a few X-sources in S_M which has large under-prediction or over-prediction, can significantly increase the value of τ_δ. This can happen when the model predicts extreme points inaccurately, and inclusion of these X-sources might not reflect the quality of the predicted grading for the rest of the inputs. In order to deal with this, we can remove $r \ll M$ elements from S_M which has the highest difference between the actual and predicted `DT-Loss`%, and then compute the τ_δ. We call this r-*eliminated* δ-Kendall's tau distance.

4.6.4 Interpreting the Prediction Results

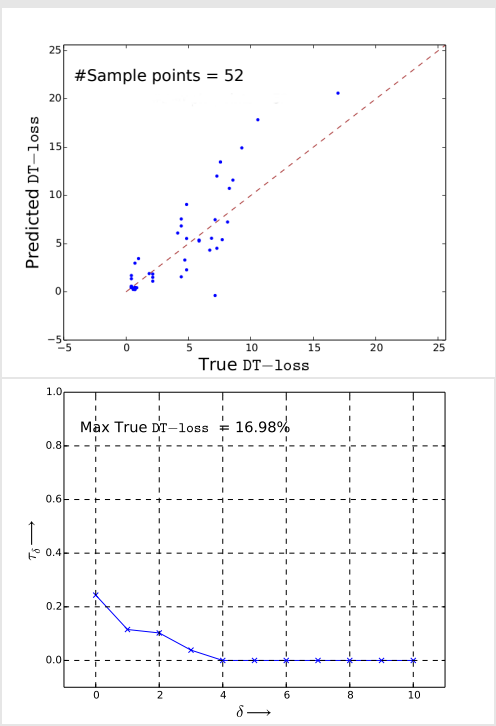
Fig. 4.5 shows two plots for each circuit in the test set. The first (top) figure shows the scatter plot and the second (bottom) figure shows the grade plot. The number of inputs for each circuit is given in the scatter plot. The maximum value of true `DT-Loss` is given in the grade plot.

Among the circuits in the test set, circuit s713 (Fig. 4.5a) is least effectively predicted by the SVR model, as seen from the CD value in Table 4.5. One reason for this, can be because that each X-source of this circuit has a large structural variation in the partition $\mathcal{P}_2$, unlike most of the other circuits. Though the feature $n_I\mathcal{P}_1$ is able to capture such variation, since the number of such circuits in the population is small, the predictive performance is not good. However, the scatter plot show that many inputs are accurately predicted and the grading plot shows that the grading of the X-sources in S_M is quite fair even for $\delta = 0$.

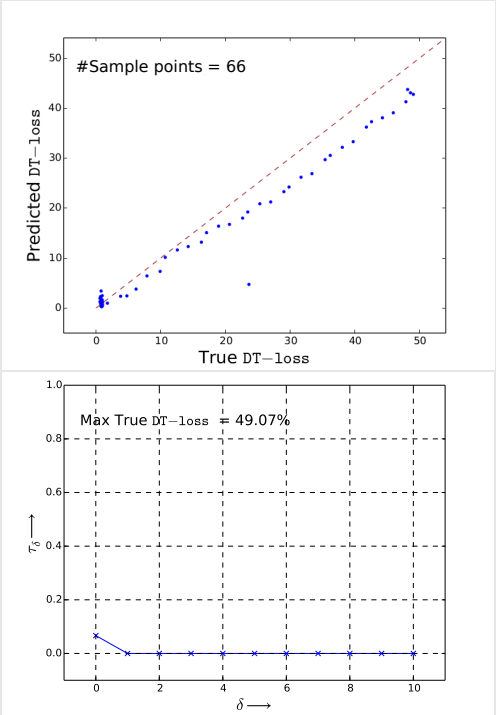
Circuit s838 has an iterative structure and so the scatter graph shows that the X-sources have uniformly varying `DT-Loss`. Though nearly all the sample points are under predicted (Fig. 4.5b), the points are quite close to the diagonal line. Only one X-source of this circuit is seen to have large under prediction. From the grade plot, it can be seen that τ_δ is very small for $\delta = 0$ and becomes 0 for $\delta = 1$, indicating that the grading is very well predicted by the SVR model.

The scatter plot for circuit s1238 (Fig. 4.5c) shows that the sample points with small `DT-Loss` are predicted well. However, there are a number of sample points with true `DT-Loss` between 10% to 50%, which have not been very accurately predicted. This effects grading of the X-sources as seen from the large values of τ_δ (the grade plot and the 1-eliminated and the 2-eliminated are also given in Fig. 4.5c).

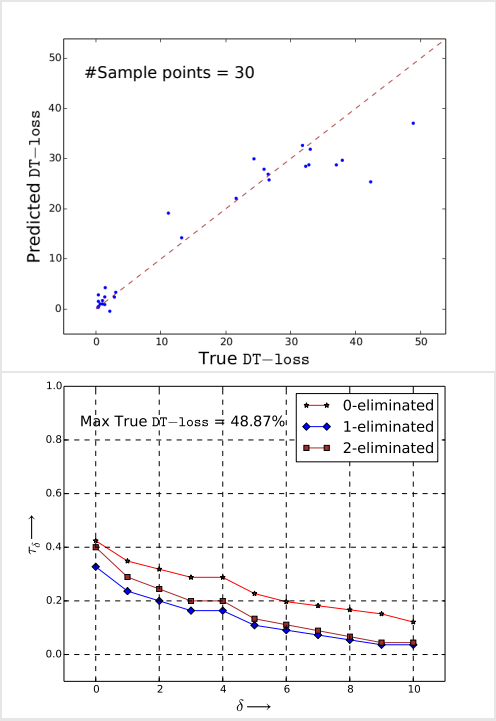
For circuit s1494 (Fig. 4.5d), the scatter plot shows that the true `DT-Loss` values vary in a wide range with the maximum being 78.35%. Almost all of them have been accurately predicted. Consequently, the grading is good, with τ_δ nearly 0 when $\delta < 3$ and τ_δ becoming 0 when $\delta = 4$ (which is a very small threshold given that the sample points are spread between 0% to 80%).



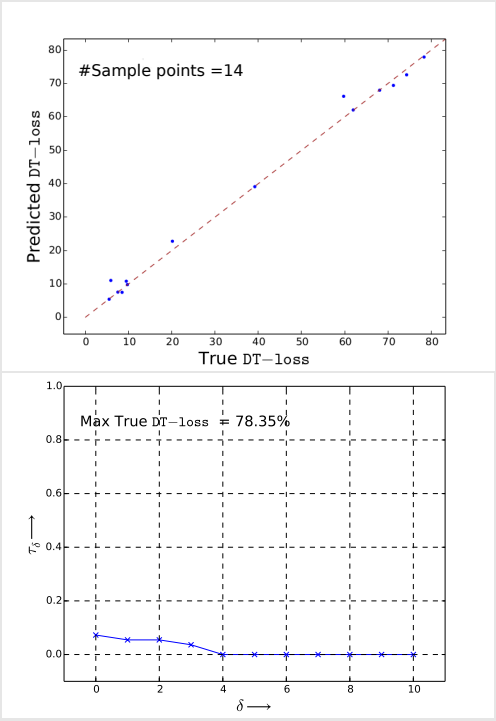
(a) s713



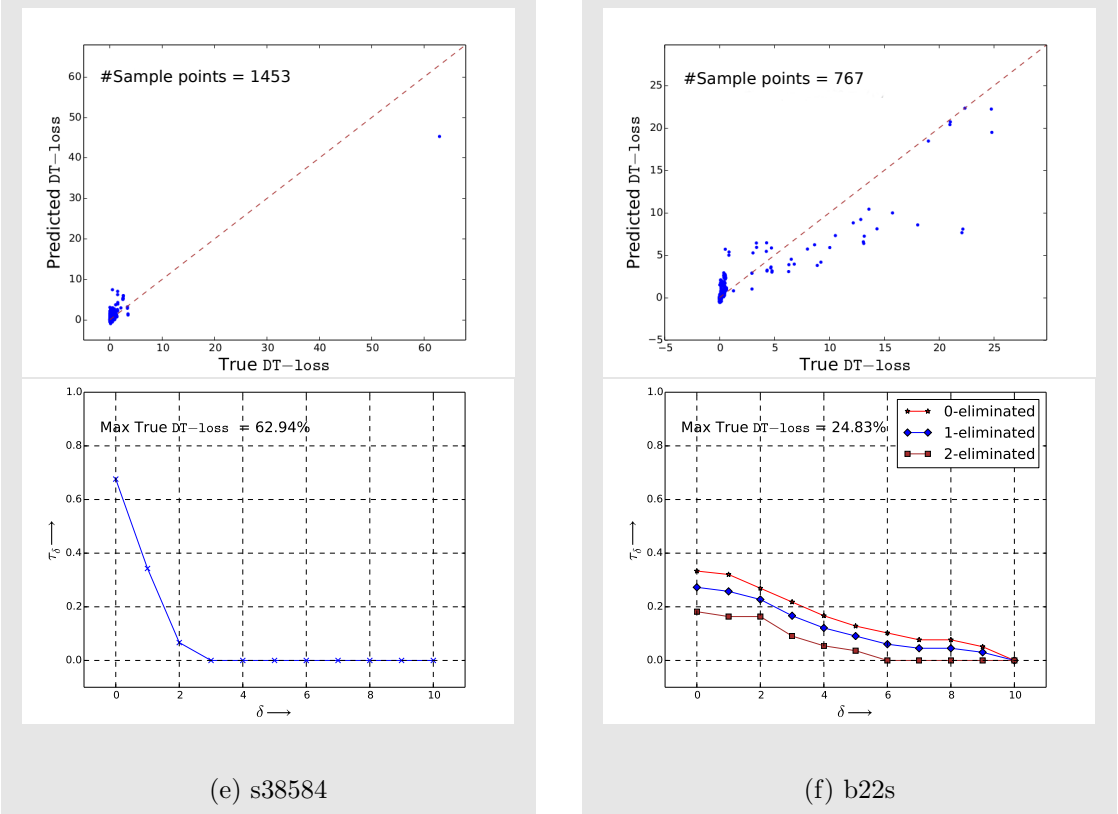
(b) s838



(c) s1238



(d) s1494



(e) s38584 (f) b22s

Figure 4.5: Scatter plots and grade plots for the circuits in the SVR test set.

Circuit s38584 (Fig. 4.5e) is a large circuit of the benchmark. It is has one X-source with large `DT-Loss`. The remaining X-sources of this circuit have very small `DT-Loss`. Even though the prediction of the `DT-Loss` is not accurate for the X-source with large `DT-Loss`, it can be seen that the prediction result could clearly distinguish it. Also, since the rest of the X-sources have very close true `DT-Loss` values, τ_δ becomes 0 for $\delta = 3$.

Lastly, in circuit b22s (Fig. 4.5f), most of the X-sources have negligible `DT-Loss`, all of which have been successfully predicted. A number of X-sources have `DT-Loss` which lie in a small range (less than 15%). It is seen that the structural difference between these X-sources are very small and so it is quite difficult to grade them minutely by prediction. A group of X-sources have `DT-Loss` more than 20%. These could be distinguished from the rest of X-sources expect for two cases. This is due to inadequacy of distinguishing features in our model. The predictions for these inputs can potentially be improved if we can find a structural uniqueness which distinguish these two X-sources. Moreover, we see from the scatter plot that these two X-sources have high under-prediction. The grade plots for the 1-eliminated and 2-eliminated of the τ_δ are shown in Fig. 4.5f.

In the following we summarize our observations from Fig. 4.5 and the above discussion:

- For all the circuits in the test set, most input points are near the diagonal line.

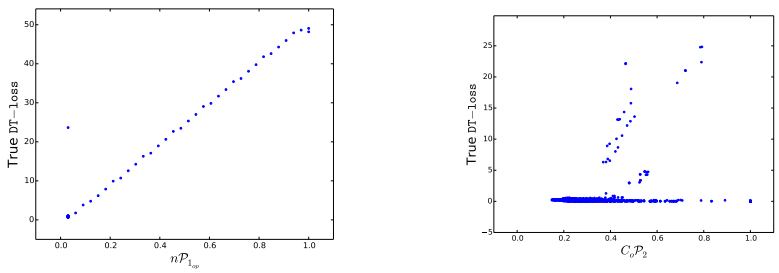
This implies that the features selected and the SVR model effectively captures the relationship between the circuit features and the detectability loss.

- For most of the circuits, a large fraction of sample points have true `DT-Loss` very close to 0. For all such sample points, the predicted values are quite accurate. Therefore, the proposed model can successfully identify the X-sources in a circuit which have small effect on the detectability loss.
- The grade plots show that for 4 out of the 6 test circuits, τ_δ is 0 for $\delta \leq 4$. This means, for these circuits each pair of the X-sources, could be graded successfully assuming that a maximum of 4% difference in true `DT-Loss` is overlooked.

The regressor may also find applications in augmenting an ATPG tool. For an illustration, we consider an ISCAS'89 benchmark circuit (s1494) and run ATPG to generate a test set and its fault-coverage is noted. The regressor predicts high X-sensitivity for one of its input ports with a DT-loss of 69% (its real `DT-loss` is 71%). Once this information is known, we set a constant value (say logic 0) to this input. When we run ATPG again on this circuit, we observe that fault-coverage of the test set produced by it increases significantly. Incidentally, `DT-loss` reduces to 17% when a constant value is set to the sensitive input.

4.6.5 Relationship of the Features with `DT-loss`

The proposed features have been envisioned in order to intuitively capture the structural diversity of the circuits in the dataset. As discussed in Section V, the circuits in the test data have been chosen such that each of them is a representative of some functional circuits that constitute the training set. This is important because, we are primarily interested in discovering the relation between DT-loss and structural features, and this can only be reflected by choosing the circuits in the test set following the criterion stated above. On plotting the standardized values of individual feature against true DT-loss for the circuit in the test set, it is seen that a unique subset of features often plays a crucial role for a given circuit. A few of the plots have been shown in the Fig. 4.6.



(a) s838 (b) b22s

Figure 4.6: Dependence of true `DT-loss` on various features

- s838: Many features vary almost linearly with `DT-loss`. One such plot for $n\mathcal{P}1_{op}$ is shown in Fig. 4.6(a).
- b22s: Two features $n\mathcal{P}1_{nodes}$ and $M\mathcal{P}_1$ are seen to distinguish the samples with small `DT-loss`. For the rest of the sample points, feature l_{P_2} and in particular, $C_o\mathcal{P}_2$ are seen to discern minute `DT-loss` variations. The plot for $C_o\mathcal{P}_2$ is shown in Fig. 4.6(b). Feature $n_{\mathcal{D}_1}$ provides good explanation for high `DT-loss`. While the importance of $DLRO$ is not apparent from the its plots, it helps improving the prediction quality for several circuits as discussed earlier in Section 4.3.

4.6.6 CPU-Time

We present here, data for CPU-time required by the SVR method for grading the DT-Loss of X-inputs of a circuit. The prediction time is mostly spent on feature computation for the X-sources, so we report the feature extraction time. As mentioned earlier, our primary aim is to grade all X-sources of a given circuit within a short time. In order to justify the efficacy of the proposed method, we report the total prediction time for all inputs of a circuit. Note that an ATPG tool can also be used for computing the DT-Loss for each input port when it is set to an X-value. We compare below, the performance of our regressor with that of an ATPG-based tool.

We report the CPU-time for some large circuits from ISCAS'89 and ITC'99 suites in Table 4.6: Column 1(4) gives the circuit name, columns 2(5) and 3(6) report the total CPU-time required for running ATPG and for feature computation, respectively. From the table we can see that the overall procedure runs within a few minutes for all the circuits, and the time required for feature computation is almost negligible compared to the time required to run ATPG.

Table 4.6: CPU-time taken by ATPG tool [J-2] and for feature computation for some ISCAS'89 and ITC'99 benchmark circuits.

Circuit	CPU-time (sec)		Circuit	CPU-time (sec)	
	ATPG	Feature computation		ATPG	Feature computation
ISCAS'89 Benchmark circuits [BBK89]					
s15850	296.96	25.6	s35932	1015.36	22.38
s38417	1341.46	54.34	s38584	1338.92	37.52
ITC'99 Benchmark circuits [CRS00]					
b14s	485.07	42.044	b15s	1987.53	145.51
b21s	2002.49	112.22	b22s	4521.06	169.58

4.6.7 Error Bars

To understand how the predicted `DT-Loss` estimate the actual values, we need to understand the variability of the estimates with respect to the measurement error. Error bars are graphical representations of the variability of estimates, which show the precision of estimates and how far away are the estimates from the values. We use the method of non-parametric bootstrap [ET93] for computing the 2-standard deviation error bar as described below:

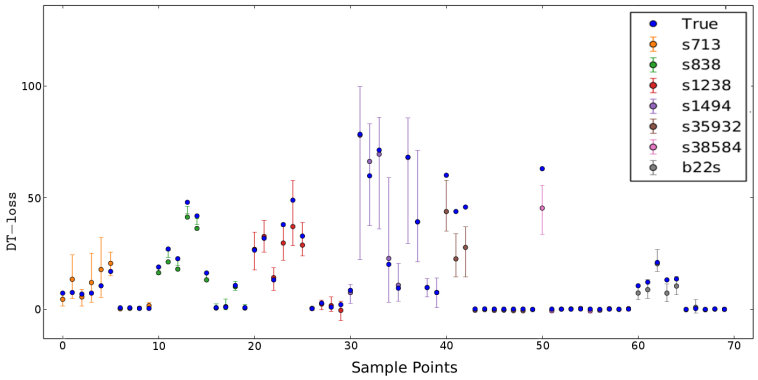


Figure 4.7: 2-Standard deviation prediction-error bars for inputs from circuits in the test set.

1. Sample N points uniformly with replacement from the training set $(x_1, y_1), \ldots, (x_N, y_N)$. Denote this sample by $(x'_1, y'_1), \ldots, (x'_N, y'_N)$.
2. Estimate the regression function f using the data $(x'_1, y'_1), \ldots, (x'_N, y'_N)$ as the new training set. Denote the function estimated by $\hat{f}'$, and the new predicted value by $\hat{y}'_0 = \hat{f}'(x_0)$.
3. Repeat steps 1 and 2, B times, which gives B different predicted values: $\hat{y}'_{01}, \hat{y}'_{02}, \ldots, \hat{y}'_{0B}$.
4. The 2-standard deviation prediction error bar, for predicting y_0, is $[\hat{\mu}_{x_0} - 2\hat{\sigma}_{x_0}, \hat{\mu}_{x_0} + 2\hat{\sigma}_{x_0}]$, where $\hat{\mu}_{x_0} = \frac{1}{B}\sum_{j=1}^{B} \hat{y}'_{0j}$ is the mean, and $\hat{\sigma}^2_{x_0} = \frac{1}{B}\sum_{j=1}^{B}(\hat{y}'_{0j} - \hat{\mu}_{x_0})^2$ is the variance of the B bootstrap estimates.

In our experiments we have taken $B = 100$ bootstrap resample. Fig. 4.7 shows the 2-standard deviation error bars of 10 randomly selected inputs from each circuit in the test set. For every circuit, we randomly choose inputs, 5 with `DT-Loss` greater than 5% and 5 inputs with `DT-Loss` less than 5%. However, few circuits have less than 5 inputs with `DT-Loss` greater than 5%, in which case, the rest of the inputs are chosen from those having `DT-Loss` less than 5%. Fig. 4.7 shows that for most of the inputs the actual values (blue points) are within the error bars, which means that the actual `DT-Loss` is always within 2-standard deviations of the predicted value. Moreover, the lengths of the bars are quite small for most of the inputs, showing accuracy of prediction. There are a few cases, particularly for inputs from circuit s838, where the true values are outside the 2-standard

deviation error bars. This is expected because our model under-predicts the `DT-Loss` for nearly all the sample points in that circuit (see Fig. 4.5b). This might be due to the lack of inputs with similar features in the training set.

4.7 Conclusion and Future Work

In this work, we have proposed, for the first time, a technique for predicting the sensitivity of X-sources in a circuit with regard to the loss of test-coverage. Instead of using ATPG and fault simulation tools, which may fare badly in the presence of unknowns, we have used a machine-learning predictor based on linear regression and support vector regression. The proposed circuit features used for prediction use only the structural information of the circuit and no functional information. Experimental results show that even based on structural information alone, the detectability-loss for the most influential X-sources in a circuit can be fairly graded using a support vector regressor. The CPU-time for feature computation is found to be very less compared to that needed by ATPG and fault simulator tools.

Another popular method for learning non-linear regression functions is artificial neural networks (ANN). This has been applied extremely successfully in various disciplines, especially when there is a large amount of data. However, when the data is limited or skewed, ANN gives limited performance, because it involves a highly non-convex optimization which might get stuck at a local optimum (see [BM17, ABF07] for recent work on these issues). On the other hand, SVM/SVR solves a convex (quadratic) optimization problem, which always returns the global optima and can be easily implemented. In our case, where the size of the data set is moderate and the dimension is small, we have chosen the SVR method to design our predictor.

In this work, we have considered the case where only one X-source is present at a time. This work can be extended to predict detectability-loss when multiple X-sources are simultaneously present. For such a case, we may consider, equivalently, a single virtual X-source that feeds the given subset of multiple X-sources. However, in order to obtain good prediction results, more elaborate training experiments need to be designed. Also, for a given k, finding the k-subset of X-sources that causes maximum detectability-loss, could be a future direction of research. In this work, we did not perform prediction for IWLS'05 benchmarks, since setting a single input to an unknown value may not affect the `DT-Loss` significantly as the circuits are large. So for such circuits, a subset of inputs need to be set to X. This may be studied as a future research problem.

CHAPTER

FIVE

ENCODING LARGE GRAPHS FOR REPRESENTATION OF LOGIC NETWORKS

5.1 Introduction

Given the pervasiveness of X-sources in modern electronic chips, the prediction of X-sensitivity of a circuit will have potential applications in circuit testing and test generation. We discussed them in the previous chapter (Chapter 4). We also noted that a logic circuit network can be represented by an acyclic directed graph, and one can predict the X-sensitivity based on the structural properties of this graph. The results on the prediction accuracy for the benchmark circuits have led to the conviction that the structural features of the circuit play a crucial role in determining the effect of X-sources on the testability of the circuit. Feature engineering, that is, the formulation of features, however, demands huge time and effort. Indeed, selecting and extracting suitable structural features for X-sensitivity prediction involves in-depth analysis of the circuit structure. This is one of major overheads that limits graph-based predictions. One solution to this problem is to deploy an automated feature-learning mechanism. In fact, there exist a number of machine-learning tools, such as artificial neural network (ANN) [Ben09], where such automated learning have been successfully incorporated. However, for graph-based data, it is still a challenge; especially, learning those features which are capable of capturing the intricacies of the full structure. In this chapter, we look at the challenges and present a method to represent a logic network such the structure of the entire circuit is preserved.

In fact, the present trend in ML research community is towards the exploitation of graph-based data. This is no wonder because graphs are found in a wide range of real-

world environments ranging from social and distribution networks to biological networks, such as neuronal networks, to VLSI circuits. In such environments, either the data naturally occur as graphs, or the interactions are modelled through graphs. The structure of the underlying graph is a repository of the properties of the system it represents. Graphs are being explored as a potential data source in machine-learning frameworks to predict various parameters of the system or classify different systems. Studies show that the prime challenge here is to design an efficient representation. Note that such graphs are usually large and hence computationally expensive to handle and secondly, they represent highly unstructured data. Data in ML applications are considered in the form of vectors called feature vectors [FM12]. Structured data such as those representing images can also be considered as vectors. However, graphs do not naturally correspond to vectors. Most of the graph-based ML approaches aim at capturing maximum relevant information in a structured format. Research in graph based-data can be classified based on (i) the approaches and (ii) the problem setting. A number of surveys are available in the literature which focus on three basic methods: (i) graph embedding [CZC18, FM12], (ii) representation learning [HYL17], and (iii) geometric deep learning [BBL+17]. Graph embedding aims at representing the graph as a vector [FM12] or as a set of vectors [CZC18]. The properties are based on local proximity, which are either manually defined or extracted in a automated fashion. The latter belongs to the class of representation learning frameworks where feature formulation is automated using ML or matrix factorization [HYL17]. Geometric deep-learning tools for graphs are based on convolutional neural nets (CNN) or recurrent neural network (RNN). Vector representations have mostly been used in a node-centric setting where a node is represented by a low-dimensional vector. They are mostly used for node classification or link prediction [GL16, TQW+15]. A few works aim at capturing full graph embedding/learning, e.g., graph kernel or graph neural network (GNN) [SGT+09]. However, none of these preserve the graph structure, or are applicable to large graphs.

Most of the published literature in geometric deep learning [BBL+17] are based on one of the two techniques, spectral [DBV16] or spatial approach [NAK16]. Tools such as CNNs that use spectral analysis have the following major drawbacks: (i) they are valid only for undirected graphs, (ii) applicable only to graphs with similar size and structure, and (iii) learning is mostly based on the weights of the graphs. Structural features that are embedded in a graph cannot be learned so easily as far as the current state-of-the-art in graph-representation is concerned. Although CNNs based on spatial analysis have been used to handle arbitrary graphs, complete structure preservation is still a concern [NAK16].

Recently, in the field of integrated circuits, various methods have been employed for efficient application ML and deep learning tools [Wan17a, HCS+18, DB17, MRK+19]. In this chapter, we present a new representation of circuit-graphs (which are directed acyclic graphs (DAG) for combinational or scan-based logic circuits). A major contribution is that the entire structure of the graph is preserved in the proposed representation. We introduce

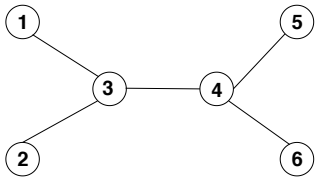
a novel encoding technique for this purpose. The encoding is lossless and is based on a very old but not fully explored, graph theoretic concept known as Prüfer sequence [Pru18]. This sequence was first used in 1918 to prove Cayley's formula, which was used to count the number of possible spanning trees in a graph with a given number of vertices. The classical Prüfer code can be used for encoding trees only; however we are concerned with graphs. For encoding of a graph, we need to represent it with a tree. In this chapter, we propose a technique called *graph-to-tree enhancement* ($\mathcal{GT}$*-enhancement*), for this purpose. We call such a tree, which represents a graph through $\mathcal{GT}$-enhancement, a *g-tree*. We present two approaches for $\mathcal{GT}$-enhancement. Further, we report new properties of Prüfer codes and discuss methods for improving interpretability and preserving the edge directions. We also discuss Prüfer codes in the light of making them learnable.

The rest of the chapter is organized as follows. The motivation and methodology appear in Section 5.2 and Section 5.3, respectively. Section 5.4 and Section 5.5 report the two methods that can be used to obtain a *g-tree*. Section 5.6 provides a discussion on the selection of Prüfer code. Conclusions and future work appear in Section 5.7.

5.2 Motivation

We will discuss the motivation highlighting three aspects. First, effectiveness of Prüfer codes in representing large graphs. Second, the efficiency of $\mathcal{GT}$-enhancement for representing a graph as a single tree. Third, the capability of a Prüfer sequence to preserve the entire structure of a circuit-graph.

As mentioned before, the graphs that appear in real world are large, and typically, they are sparse graphs. Hence, for a graph $G(V, E)$, where V is the vertex set and E is the edge set of G, $|E| \ll |V|^2$. In general, in sparse graphs $|E| = \mathcal{O}(|V|)$. In fact, for most of the digital test benchmark circuits [BBK89, CRS00], $|E|$ is not more that $2 \times |V|$. Prüfer codes can be exploited to store a graph in terms of its edges as follows. The Prüfer encoding represents a labeled tree of n nodes (and hence having $n-1$ edges) by a string of vertex-labels whose length is $n-2$. So, the size of the code is same as the number of edges (neglecting the difference of one). Figure 5.1 shows an example of a Prüfer code of a tree.



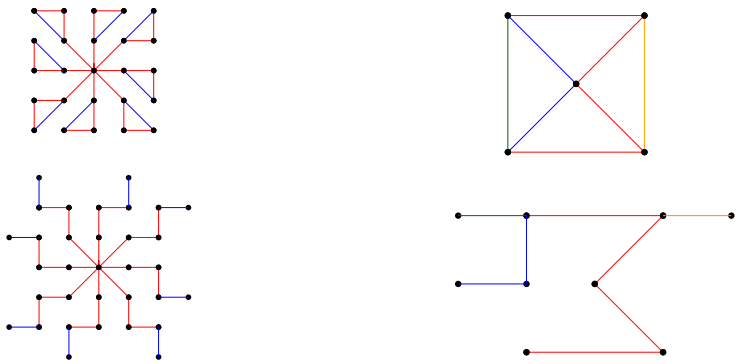
(a) Example labeled tree.

3344

(b) Prüfer code of the tree in Figure 5.1a

Figure 5.1: Example of Prüfer code of a tree.

A labeled tree with six vertices is shown in Figure 5.1a and its corresponding Prüfer code is shown in Figure 5.1b. The size of the code is four. The algorithm for encoding the tree and decoding it is discussed in the next section. The $\mathcal{GT}$-*enhancement* of G preserves the number of edges of G in *g-tree*. Thus, the size of the Prüfer code of a *g-tree* is $|E-1|$.



(a) (b)

Figure 5.2: Graphs with large number of tree-partitions. Top: (a) Sparse graph with 25 vertices 32 edges and 9 tree partitions ; (b) Dense graph with 5 vertices 8 edges and 4 tree partitions. Bottom: the corresponding *g-trees*.

A simple way of representing a graph G using Prüfer codes would be to represent it as a union of trees so that the graph can be represented as the corresponding set of Prüfer codes for individual trees. However, the problem of partitioning the edge set E of a graph G into minimum number (k) of trees is known to be NP-hard [BB07]. Moreover for sparse graphs, even k can be very large (Figure 5.2(a)). Also if the (sub) graphs are dense, the number of trees could be large (Figure. 5.2(b)). For a complete graph K_n, the minimum number of tree-partitions is $\lceil n/2 \rceil$. So, if the number of tree-partitions is comparable to $|V|$, then the method would not be beneficial. In contrast, the proposed $\mathcal{GT}$-enhancement provides a nice solution to this problem, in which the graph can be represented by just a single tree at the cost of adding $E+1-V$ vertices thus forming the *g-tree*.

An illustration on structure preservation and reconstruction of a circuit-graph is given in Figure 5.3 with the help of an ISCAS'89 [BBK89] benchmark circuit s27. The netlist and the corresponding circuit diagram is shown in Figure 5.3(a) and Figure 5.3(b), respectively. The circuit-graph generated from the netlist is given in Figure 5.3(c). Note that the graph has two cycles. $\mathcal{GT}$-enhancement, which adds two vertices 8^1 and 10^1, creates the *g-tree* shown in Figure 5.3(d). The tree is then encoded by a Prüfer sequence as shown in Figure 5.3(e). The tree structure can be completely reconstructed as shown in Figure 5.3(f).

In summary, we propose a new encoding scheme for digital networks based on Prüfer sequence which has the following useful properties:

1. it is lossless, i.e., captures the structure of the entire graph;
2. it is memory efficient; the size of encoding is $\mathcal{O}(|V|)$;

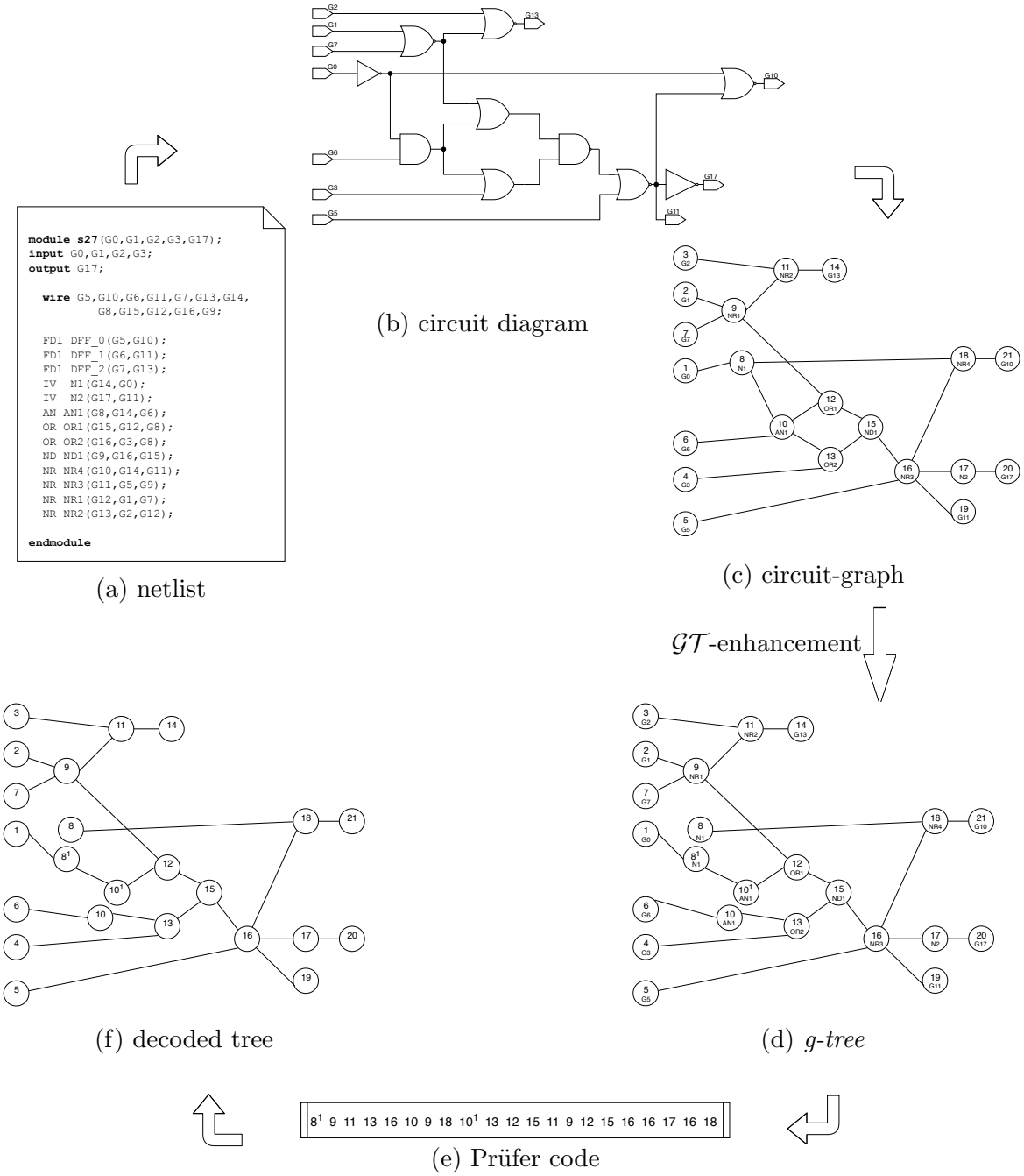


Figure 5.3: An illustration of Prüfer code for the circuit-graph representing the benchmark circuit s27, and reconstructing it from the Prüfer code.

3. the representation can be expressed as a single string of vertex labels;
4. it provides an efficient 1-D representation suitable for machine-learning tools;
5. the encoding can be computed in time linear in the size of the circuit.

6. the directions on edges, especially for DAGs, are preserved.

5.3 Methodology

In this section we will briefly look at the Prüfer encoding of a tree. We will also introduce the *g-tree* and show how Prüfer encoding can be applied to it.

5.3.1 Prüfer-Code

Consider a tree, T, with n vertices. Prüfer encoding is based on the following assumptions concerning the labels attached to the nodes:

Assumption 1. *The vertices of T are labelled sequentially as $\{\text{`1'}, \text{`2'}, \cdots \text{`n'}\}$.*

Assumption 2. *The vertices of T are comparable such that $\{\text{`1'} < \text{`2'} < \cdots < \text{`n'}\}$.*

The steps to encode T is given in Procedure 5.

Procedure 5 Encoding: tree-to-code

1: Fetch a single degree vertex, v, with the smallest label. Delete v.
2: Write the label of the neighbour of v to the right of the code.
3: Repeat Step 2 until only two vertices remain in the tree.

Consider a Prüfer code of T: $c_1, c_2, \cdots, c_m$; we make the following observation:

Observation 6. $n = m + 2$.

Observation 7. *The degree of a vertex is one more than the number of times the label of a vertex appears in the code.*

Corollary 1. *Labels of single-degree vertices do not appear in the code.*

Corollary 2. *Labels of non-pendant vertices appear in the code.*

The steps for decoding Prüfer code are given in Procedure 6.

Procedure 6 Decoding: code to tree [GJRR01]

1: Initialize a variable k to 1.
2: Compute: $n \leftarrow m + 2$; labels $\leftarrow \{1 \cdots n\}$. ▷ Observation 6.
3: Compute the degree of each node. ▷ Observation 7.
4: Fetch a single-degree vertex, v, with the smallest label. Set (v, c_k) as an edge of the tree.
5: Decrement the degree of v and c_k; increment k.
6: Repeat Step 4 and Step 5 until all vertices have degree 0, except a pair with degree 1. ▷ These form the last edge of the tree.

A linear-time encoding and a decoding algorithm for Prüfer sequence are given in [WWW09]. During decoding, the labels of the vertices can be easily obtained from the value of n (Step 2, Procedure 6) on the basis of Assumption 1. However, when the labels of a tree do not hold this assumption, they cannot be similarly extracted. The decoding of labels is the most critical step for analyzing such trees. In order to handle these cases, we decode the labels as follows. The labels of non-pendant vertices can be obtained from the code (Observation 7). Unfortunately, since the labels of the rest of the vertices do not appear in the code (Corollary 1), they can not be inferred and so an additional list L of extra vertices will be required. Next, we will introduce the concept of *g-tree* and its encoding.

5.3.2 $\mathcal{GT}$-Enhancement and Encoding of g-tree

$\mathcal{GT}$-enhancement of a graph $G(V, E)$ produces a tree called *g-tree*, denoted as $T_g(V^T, E^T)$. $\mathcal{GT}$-enhancement adds a set R of new vertices, in order to remove the cycles in G. This is done by splitting the vertices which are a part of a cycle in the graph. So, if vertex v is split/replicated by adding one more vertex v^1, then a sub-set of edges that were incident on v are now incident on v^1, such that these edges no longer form a cycle. This method is elaborated further in Section 5.4. So, $V^T = V \cup R$ and $|E^T|$ is equal to $|E|$. Since, T_g has $|E|$ edges, so the number of vertices, $|V^T| = n = |E| + 1$. Let $n_1 = |V|$. Thus $|R| = n_2 = |E| + 1 - |V|$. So, $n = n_1 + n_2$. In T_g, the vertices in V are labelled as {'1' $\cdots$ 'n_1'}. The vertices in R are added to G by replicating some vertices in V (explained in the next section). When a vertex v with label l, where l in an integer, is replicated into k additional vertices, they are labelled as $l^1, l^2, \cdots, l^k$. While encoding T_g with a Prüfer code, the labels of these vertices are considered in the order $l < l^1 < l^2 < \cdots < l^k < L + 1$ (Assumption 2). Assumption 1 partially holds for T_g since it is true only for n_1 vertices. If n_1 is known, the labels of these vertices can be computed. For decoding T_g, in the list L, only the single degree vertices from R need to be stored, and n_1 can be computed as follows. From the code, we compute n (Observation 6). With the help of L, we can obtain the labels of the vertices in R. Once we obtain n_2 and n, n_1 can be directly computed.

Next, we propose two methods to generate a *g-tree*. The two methods differ in the way the graph is traversed and the way the vertices in the graph is split. The first method aims at reducing the list L such that $|L| \ll |E|$. The second method produces the *g-tree* where all the additional vertices are of degree two and so no extra list is required; the Prüfer code thus obtained completely represents the graph.

5.4 Tree-Partition Based $\mathcal{GT}$-Enhancement

In this method, representing an undirected graph by a tree is realized by partitioning the edges of the graph into trees. Thereafter, the individual trees are joined to from a single

tree.

5.4.1 Proposed Approach

Since the problem of finding a minimum tree-partition of a graph G is hard [BB07], we follow a greedy approach based on depth-first search (DFS) [CSRL01] traversal of G. DFS of an undirected graph produces a spanning tree T_{DFS}; the edges of this tree called are tree-edges and they form the primary partition. The rest of the edges of the subgraph $G^c = G \setminus T_{DFS}$ are called back edges. While the DFS spanning tree is implicitly constructed during the traversal, the residual graph G^c is often found to be disconnected. The secondary partitions, which are formed by the back edges of G^c, are called *be-trees* and belong to one of the following:

1. Class-1 back-edge tree (*be-tree-1*): These are trees formed by the back edges between the vertices, which have already been visited once.
2. Class-2 back-edge tree (*be-tree-2*): These are trees formed by the back edges between the vertices, which have already been visited twice: by the DFS-tree and also by a be-tree-1.
3. q^{th} class back-edge tree (*be-tree-q*): The trees formed by the back edges between the vertices that have already been revisited $q-1$ times, are called *be-tree-q*.

Note that the DFS-traversal usually yields a number trees for each class. For a given class i, all trees in *be-tree-i* are independent to each other since they have no vertex in common. The *be-trees* that are non-independent to a given k^{th} be-tree-1, are said belong to the same *family*, f_k. Thus, the edges of the graph are partitioned into a spanning DFS-tree and a set of families of non-independent trees.

Example: Figure 5.4(a) shows an example graph, where the DFS-tree is shown in red. The decomposition consists of one family of *be-trees* with two classes: *be-tree-1* (blue) and *be-tree-2* (green). Also note that in Figure 5.2(a), the graph has eight families of be-trees.

Reducing the number of be-tree classes

Consider a pair of vertices, u and v, of G such that they are connected by an edge e and also by two paths p_{dfs} and p_{be1}. Suppose DFS exploration, traverses the path p_{dfs}, including it in the DFS-tree, such that v is visited before u. Let p_{be1} be assigned to *be-tree-1*, b_1. We define two special edges:

Cycle-edge: At some instant during the traversal when vertex u is being processed, let edge e be a back edge from u to v. Since it will introduce a cycle in b_1, it is assigned to *be-tree-2*. Such an edge is called a cycle-edge.

Swap-edge: Now, suppose there exists a tree-edge e_s, in the path p_{dfs}, which does not form a cycle with any *be-tree-1*. Then the edge e_s is swapped with e; e_s is then called a swap-edge for the cycle-edge e.

Edge-swap: An edge-swap between the cycle-edge e and its swap-edge e_s enables e being assigned to a tree-edge and e_s being assigned to *be-tree-1*. A new class of *be-tree* for the cycle-edge e is thus avoided by an edge-swap operation. Note that an edge-swap, modifies the DFS-tree, without disconnecting it.

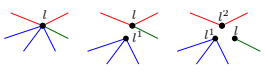
Example: Consider the graph in Figure 5.4(a); edge (7-4) and edge(5-6) would be a cycle-edge and swap-edge, respectively. The resulting graph is shown in Figure 5.4(b).

(a) Partitions with *be-trees* (b) Edge-swap

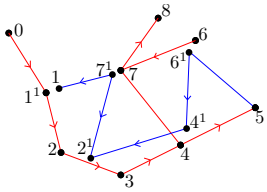
Figure 5.4: Example graph

$\mathcal{GT}$-enhancement

Given a graph $G(V, E)$ and the set P of secondary partitions, for each partition in P, a *join vertex* (v_{join}) is specified. G is decomposed into a *g-tree*, T_g, based on P and their join vertices. Each vertex in V, belongs to the primary partition. For every vertex v that belongs to some secondary partition P_j, a *vertex-split* operation (refer to Figure 5.5(a)) is performed. The vertex v is split into a replica v^{P_j} for every partition P_j where it belongs to, such that the edges of G that belong to P_j and were incident on v, become incident on v^{P_j}. However, if this vertex is a join-vertex of a partition, its replica for that partition is not created. Thus, the tree T_g is formed by the trees for each partition in P, each tree sharing a common vertex with the spanning tree at its join vertex. We called this tree *DFS-Partition-tree* and denote it as T_G^{DP}. The labeling methodology for the new vertices was described before in Section 5.3.



(a) Vertex split with label-swap



1^1 7^1 2 3 4 7 2^1 4^1 6^1 5 4 7

(b) Example tree for the graph in Figure 5.4(b) & its Prüfer sequence

Figure 5.5: *g-tree* and label-swap operation

Example: On performing tree decomposition on the graph in Figure 5.4(b), we obtain the resulting graph as shown in Figure 5.5(b).

Once we obtain the partitions though DFS-traversal, there is a scope of further reducing the list L by taking care of how we form *g-tree*. Reducing L means transforming as many labels in R (the set of new vertices) that represent pendant vertices, to point to non-pendant vertices. In order to do such transformation, we perform the following operations:

1. *Choice of join-vertex*: It is so chosen that it becomes a pendant vertex of the *be-tree* and the DFS-tree (if possible);
2. *Label-swap*: The replicas of a vertex in v have similar labels; so exchanging the labels does not change the structure of G. The operation label-swap is applicable to a pair of vertices (v, v^r) with corresponding label (l, l^r), where v^r is a replica vertex of v, v is a non-pendant vertex and v^r is a pendant vertex. In such case their labels are swapped. So, the label l^r now points to a non-pendant vertex and hence it does not need to be stored (Corollary 2). So, this operation decreases the size of L.
3. *Edge-swap*: The number of replicas for a vertex is equal to the number of *be-tree* classes that it belongs to. Edge-swap reduces the number of replicas of a vertex. Also, the number of labels that needs to be possibly stored, is reduced.

Example: The Figure 5.5(a) depicts label-swap. Also in Figure 5.5(b), label-swap is performed for the vertex with label "1". The vertex with label "5" is chosen as join-vertex since it is pendant in both the trees it belongs to. Hence, after all operations, the list L reduces to ϕ. The corresponding Prüfer sequence is shown in the figure.

5.4.2 Implementation

In this section, we discuss the outline of our implementation on benchmark integrated circuits [BBK89, CRS00]. Since scan-based circuits are envisaged as directed acyclic graphs, we first label their nodes in topologically-sorted order [Kah62] and run a single DFS-traversal, considering it as an undirected graph (directions can be retrieved from the labels). Also, the successors of each vertex are visited in the increasing order of their labels. When an edge $e(u, v)$ is discovered, it is (i) assigned to a particular partition by an operation called add-edge (described later), or (ii) added to the list of cycle-edges, or (iii) assigned to a partition (add-edge) if it is already marked as a cycle-edge and has not yet been swapped. When an edge is backtracked, we check whether it is suitable for swapping with a swap-edge in the list. Lastly, once all the edges emanating from a vertex are processed, vertex-splitting along with label-swap is performed on it if needed.

During each successive iteration of the DFS-traversal, search is made to explore an unvisited vertex, and if found, the connecting edge is added to the DFS-tree. Similar implicit exploration cannot, however, be performed to extract *be-trees*. A pair of edges in a *be-tree*, or the edges of the successive classes in a family of *be-trees*, could be discovered

during two independent iterations. Thus, to keep track of *be-trees*, we record the following entities, which are updated whenever a new back-edge is discovered: (i) *tree index*: every back-edge that does not share a common vertex with any other *be-tree* formed so far, or form a cycle with the adjacent *be-trees*, is considered a new *be-tree* with one-higher index; (ii) vertex-attribute for each vertex v (tree-index list $\mathcal{T}_v$): a ordered tuple of indices of different classes of *be-trees* to which the vertex belongs, and (iii) edge-attribute for each edge e (c_e): the *be-tree* class index. $\mathcal{T}_v[c_e]$ gives the tree-index of the c_e^{th} class.

add-edge: When a unvisited back edge $e(u, v)$ is encountered with tree-index lists $\mathcal{T}_v$ and $\mathcal{T}_u$, c_e is computed from the two lists such that it does not form a cycle with any of the trees to which v and u belong. Thereafter, it is assigned to a *be-tree* by updating the lists depending on the following three cases; (i) union of two *be-trees*: Suppose e connects two back-edges $e_1(v, w_1)$ and $e_1(w_2, u)$, each belonging to distinct *be-trees* with tree-indexes i_1 and i_2, respectively, but both having the same tree-class. In such a case, the three edges are assigned to a single *be-tree* having tree-index i_1. This is done by treating the tree-indices as disjoint sets. Using the disjoint-set data structure, we perform union(i_1, i_2); (ii) if only one of the two vertices belongs to a unique class of *be-tree*, the edge e is assigned to that *be-tree*; (iii) lastly, if it does not fall into the above two cases, it creates a new *be-tree* as explained above.

The time complexity of the algorithm is $\mathcal{O}(|V| + |E|)$.

5.4.3 Results on Benchmark Circuits

Results for ISCAS'89 and ITC'99 benchmark-suites are shown in Table 5.1. The experiments were carried out on an Intel Xeon 3.00-GHz $\times$ 4 processor with 8GB memory. Each of the benchmark circuits corresponds to the netlist of a digital logic circuit, where the input-ports/output-ports, logic gates, and memory cells are represented as vertices, and the interconnections among them, as edges, of a directed acyclic graph. The circuit-name, the number of vertices, and edges are given in Columns 1, 2 and 3, respectively. These graphs are very sparse as the edge-count is just around two times of the vertex-count. The length of Prüfer code is given in Column 4. The required number of extra-labels (L) is given in Column 5, which is much less than the number of edges. The highest-index of *be-trees* is given in Column 6, and the number of edge-swaps is given in Column 7. The number of vertex-splits is shown in Column 8, and out of it the number of vertices whose labels are swapped, is shown in Column 9. The CPU-time needed to encode the circuit using Prüfer-sequence is reported in Column 10, in seconds.

Table 5.1: Results on logic circuits in ISCAS'89 and ITC'99 benchmark-suites.

Circuit	#vertices	#edges	Prüfer code length	#extra_ labels ($\|L\|$)	#be_ trees	#edge_ swap	#vertex _split	#label_ swap	CPU-time in sec.
ISCAS'89 benchmark circuits [BBK89]									
s713	489	635	634	3	2	7	147	62	1.07
s820	336	781	780	142	3	2	446	193	4.06
s832	334	793	792	173	4	1	460	173	4.25
s838	545	820	819	11	1	8	276	139	1.5
s953	492	818	817	12	2	6	327	165	1.6
s1196	593	1043	1042	38	2	18	451	225	2.36
s1238	572	1075	1074	41	2	19	504	264	2.56
s1423	827	1245	1244	57	2	5	419	202	2.68
s1488	692	1412	1411	242	4	1	721	302	7.82
s1494	686	1418	1417	209	4	6	733	354	8.16
s5378	3206	4435	4434	27	2	1	1230	606	7.76
s9234	6094	8235	8234	292	4	25	2142	1048	15.29
s13207	9441	12048	12047	450	4	14	2608	1184	24.56
s15850	11067	14380	14379	568	4	10	3314	1594	28.74
s38417	25585	33969	33968	1202	4	4	8385	4535	77.78
s38584	22447	34497	34496	2325	4	5	12051	5628	125.77
ITC'99 benchmark circuits [CRS00]									
b11s	512	972	971	39	3	4	461	185	2.23
b12	1155	2022	2021	65	2	10	868	434	4.71
b13s	392	601	600	0	2	7	210	111	1.12
b14s	5020	9862	9861	1324	3	26	4843	1881	40.88
b15s	9343	19068	19067	2534	3	39	9726	3909	87.38
b17s	25615	52447	52446	7282	3	42	26833	10175	307.35
b20s	9909	19555	19554	2739	3	21	9647	3620	98.39
b21s	10293	20300	20299	2387	3	46	10008	3871	104.73
b22s	15836	31304	31303	4105	3	11	15469	5856	158.33

5.5 Improved $\mathcal{GT}$-Enhancement

The above tree-partition based method for constructing $\mathcal{GT}$-enhancement has the following drawbacks:

1. The list L needed in addition to the Prüfer code in order to decode the graph. A single string would be more convienient for good representation of data from ML point of view.
2. Many operations need to be handled in order to reduce the size of L.
3. In the cases where a vertex has high degree and that of the most of its neighbors are two, then the number of single-degree nodes which cannot be label-swapped could be large. An instance of this is shown in the Figure 5.6 which shows a sub-graph. The edges belonging to the DFS-tree is shown in red and the edges belonging to a

be-tree are shown in blue. The vertex v_a has high degree with four (v_a, v_b, v_c, v_d) of its adjacent vertices of degree two. Considering that the v_e is the join-vertex, the rest of the three vertices, v_a, v_b, v_c, are split into a pairs of pendant vertices during $\mathcal{GT}$-enhancement, and hence cannot be label-swapped. However, there exists another *g-tree* (shown in dotted-green) for this sub-graph, where the three vertices need not be split.

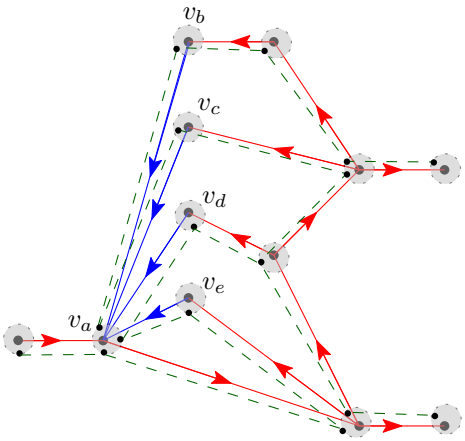


Figure 5.6: An example of an instance showing a sub-graph where a large number of pendant vertices can not be label-swapped.

In the light of above scenario, a second method of $\mathcal{GT}$- enhancement is proposed here, which overcomes these shortcomings. We call this method Seek-Edge-aNd-Split-On-Revisit (SENSOR) $\mathcal{GT}$-enhancement. This method implicitly maintains that any new vertex that is added during $\mathcal{GT}$-enhancement is of degree two. Thus, this method obviates the need for the list L. Thus, *g-tree* can be represented by a single Prüfer code. Futhermore, the method is much simpler to implement. The SENSOR $\mathcal{GT}$-enhancement follows two operations.

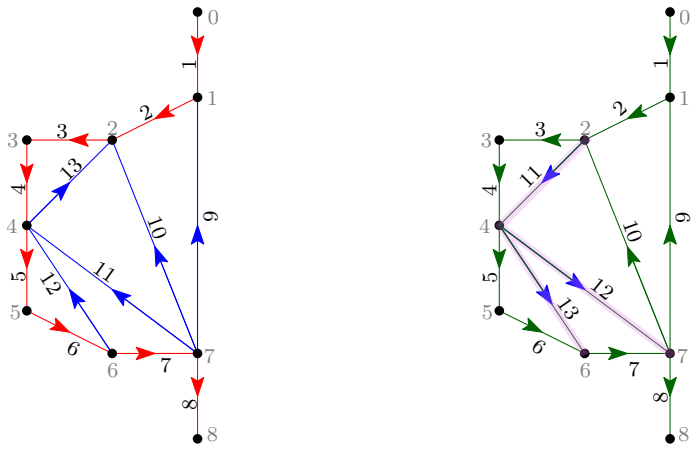
1. Seek-edge graph traversal
2. Vertex-split-on-revisit during traversal.

5.5.1 Seek-Edge (SE) Traversal

Conventional graph-traversal algorithms such as DFS and BFS (breadth-first- search) follow an exploration method that is vertex-centric; the term "search" refers to visiting the vertices sequentially, and the search technique is either "depth-first" or "breadth-first". In contrast, our purpose is to arrive at a graph-encoding scheme that captures its information in terms of edges. We therefore require a traversal scheme which is rather edge-centric; where the exploration will be guided by edges rather than vertices. Towards this end, we propose an edge-centric graph-traversal method called *Seek-Edge* (SE) traversal. The SE-algorithm of

a graph G starting from a vertex $v \in G$ is as follows. An unvisited edge adjacent to v, $e^1(v, w)$ is visited (flagged). Iteratively, the next unvisited edge adjacent to w is visited. During the traversal, if no unvisited adjacent-edge is available from the current vertex u, then the algorithm backtracks to its parent vertex u_p. If there is any unvisited edge adjacent to u_p, it is iteratively visited, and the process is continued until all edges in G are flagged.

The central idea of this traversal algorithm is that whether a vertex is discovered for the first time or is revisited, in both the cases, the vertex is processed in the same manner. So, the traversal is controlled by the edges and not by the vertices. An example of the traversal on the graph in Fig. 5.4 is given in Fig. 5.7. The graph is redrawn for clarity. The edge-labels depict the sequence number in which the edges are traversed. The arrow heads show the direction of traversal. Fig. 5.7(a) shows the DFS traversal sequence and Fig. 5.7(b) shows the SE-sequence. Since the adjacent edges of a vertex can be chosen for traversal in any order, we assume that this order is same for both traversals for the sake of comparison. So, the edge sequence is identical up to 10. The 10^{th} edge is (7,2), which is incident on vertex 2. The next unvisited edge adjacent to 2 is (2,4.) So, in SE, the 11th edge traversed is (2,4). Note that, although the edge (2,4) is readily traversable, DFS does not allow traversal of this edge, and instead backtracks. In fact, DFS processes the edge (2,4) in the end after processing (4,6) and (4,7). Thus, SE is more convienient for systematic exploration of edges.



(a) DFS (b) SE

Figure 5.7: The graph in Fig. 5.4 is redrawn here to show the difference in the sequence of edge-traversal. Three edges are shaded in (b) to highlight their difference.

5.5.2 Split-On-Revisit (SOR)

In this section, we explain the mechanism for vertwx-splitting during the SE traversal of the graph employing a technique similar to the previous approach. During the traversal, if a

vertex v with label l is revisited through an edge e_{cycle}, it implies that it is a part of a cycle. To break the cycle, v is split as follows: a new a vertex v^1, labeled l^1, is created. During the previous visit to v, let e_{in} be the edge through which it was visited and let e_{out} be its adjacent edge which was next traversed. Obviously, e_{out} is also a part of the cycle. So, the pair of edges (e_{in}, e_{out}) is now connected to v^1 instead of v. The edge e_{in} maintains the connectivity to the edges traversed up to it while the edge e_{out} contributes to disconnecting the cycle and the maintenance of connectivity to the rest of the edges. We call (e_{in}, e_{out}) the *split-pair* edges of v. On every successive revisits to v, it is split similarly, and the i^{th} revisit will produce a replica which is labeled as l^i. Thus, the degree of v is reduced by two in every revisit. Such a traversal breaks all the cycles while maintaining the connectivity of the graph turning it into a tree. We call this method *Split-On-Revisit* or *SOR* and the *g-tree* obtained is called *SENSOR-tree*. The SENSOR-tree of the graph in Fig. 5.7(b) is shown in Fig. 5.8. The five instances of vertex-splits are circled in pink shade in the figure. For vertex 1, it is split into vertex 1^1 with the split-pair being ((0,1),(1,2)). Similarly, we can observe other split-pairs which are marked red in the figure.

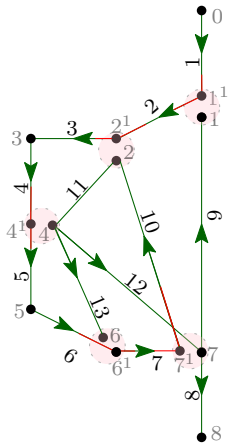


Figure 5.8: SENSOR-tree of the graph in Fig. 5.7(b).

The pseudocode for the algorithm is given in Algorithm 7.

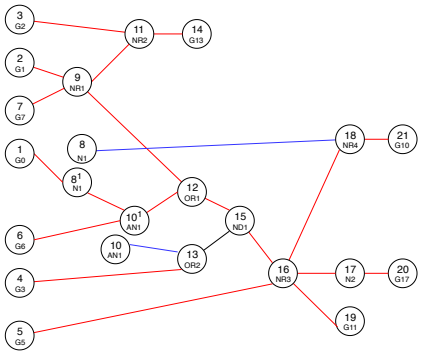
Algorithm 7 SENSOR(v)

```
1: if v is visited earlier then
2:     split v                                  } SOR
3: end if
4: visit v
5: while edge(w, v) in unvisited adjacent edge-list of v do   } SE
6:     SENSOR(w)
7: end while
```

A SENSOR transformation of a graph $G(E, V)$ splits a subset of vertices in V. In the

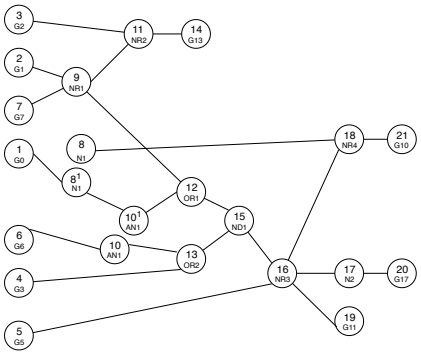
SENSOR-tree thus obtained, the vertices fall into three categories, where a vertex is given a denotation reflecting its category. A vertex $\in V$ that is split, leading to its degree being decremented, is denoted as s-vertex. A vertex belonging to the set of remaining vertices $\in V$, which are not split, is denoted as g-vertex. A replica vertex created by vertex-split is denoted as r-vertex. We make the following observations about a SENSOR-tree. (i) An r-vertex is of degree two, (ii) An s-vertex may be of degree one or more. (iii) If an s-vertex is pendant then its degree in G is odd.

So, only the r-vertices carry the new labels. From the first observation, we conclude that the new labels always appear in the Prüfer code of the SENSOR-tree for any given graph. Hence, the Prüfer code alone is enough to infer the labels of the vertices following the method described in Section 5.3.2, thus decoding the tree.



Prüfer code:
8^1 9 11 13 16 10 9 18 10^1 13 12 15 11 9 12 15 16 16 17 16 18

(a) Tree-partition based method



Prüfer code:
8^1 9 11 13 16 10^1 9 18 10^1 13 12 15 11 9 12 15 16 16 17 16 18

(b) SENSOR-based method

Figure 5.9: An example of *g-tree* and Prüfer code of s27 using both the methods

The *g-tree*s and the corresponding Prüfer sequences for s27 obtained using the two approaches is given in Figure 5.9. The two new replica vertices added are 8^1 and 10^1. Figure 5.9a shows the *g-tree* for the tree-partition-based method. There are two be-trees, marked in blue. The new vertices are pendant and so the vertex-label is swapped for both of them, making the list of extra vertices L empty. However this is not the case with most of the benchmark circuits as depicted in Table 5.1. The SENSOR-tree for s27 is given in Figure 5.9b. It can be seen that both the r-vertices are of degree two. As discussed, this method implicitly produces r-vertices of degree two for any given circuit-graph.

In the next section, we will discuss a few methods that lead to further improvement of the attributes of the Prüfer code.

5.6 Prüfer Code Selection

We observed that the structure of a tree can be fully preserved by its Prüfer code. However, we will show that the Prüfer code for a tree may not be unique. In fact, a tree can be encoded with several Prüfer codes. To elucidate this, we discuss some of their properties. One has an option to choose a suitable code to represent a *g-tree* such that the chosen code (i) exhibits good interpretability, (ii) preserves the attributes of the graph such as the directions of edges, and (iii) it is learnable. We present three such codes for *g-tree*. Lastly, we discuss about the learnability of such codes.

5.6.1 Properties of Prüfer Code

Consider an unlabelled tree T with n vertices. A Prüfer code of T has the following properties:

1. *A Prüfer encoding induces an edge sequence.*

   A Prüfer encoder monitors the list of pendant vertices. Let us call this list *Pen_List*. In each iteration, *Pen_List* is updated. From Procedure 5, we recall that the basic operation in each iteration of encoding (decoding) T into a Prüfer sequence is: "choose the pendant vertex (u) with the smallest label, encode the label l_v of its adjacent vertex (v) as the next element of the code, and remove the vertex u from *Pen_List*. If v is now a pendant vertex, add it to *Pen_List*. Thus, in each iteration, an edge (u, v) is encoded. Although Prüfer code comprises a string of vertex labels, each label l_v in the code actually represents an edge (u, v). Hence, a Prüfer code represents a *sequence of edges* rather than vertices. Out of the set, *Edge_Seq*, of all possible edge-sequences in a tree, a Prüfer encoding induces a *sub-set*, *Edge_Seq_Prüfer*, of the set of edge-sequences. we next look at this sub-set, *Edge_Seq_Prüfer*.

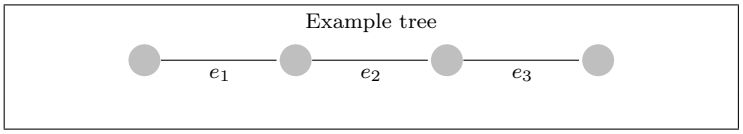
   For an edge to be selected in a particular iteration, one of its end-vertices should satisfy two conditions:

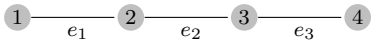
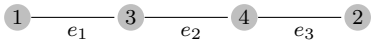
   a) it should belong to *Pen_List*, and

   b) it should have the smallest label among them.

   Hence, in order to obtain a desired edge sequence, we need to take care of the above two conditions. The first condition is mostly affected by the structure of tree and hence, we have little control over it. However, the second condition depends on the labeling of vertices. *Edge_Seq_Prüfer* is the subset of *Edge_Seq*, where an edge is prioritized on the basis of creating a pendant vertex. An example to demonstrate this is given in Table 5.2. Edges e_1 and e_3 of the tree can be named interchangeably because of the symmetry of the tree structure. Note that, the size of the set *Edge_Seq* is three, and its elements are listed in the first column; first two sequences can be encoded with Prüfer code and so belong to *Edge_Seq_Prüfer*. The third sequence cannot be encoded by a Prüfer code.

Table 5.2: Edge sequence example.



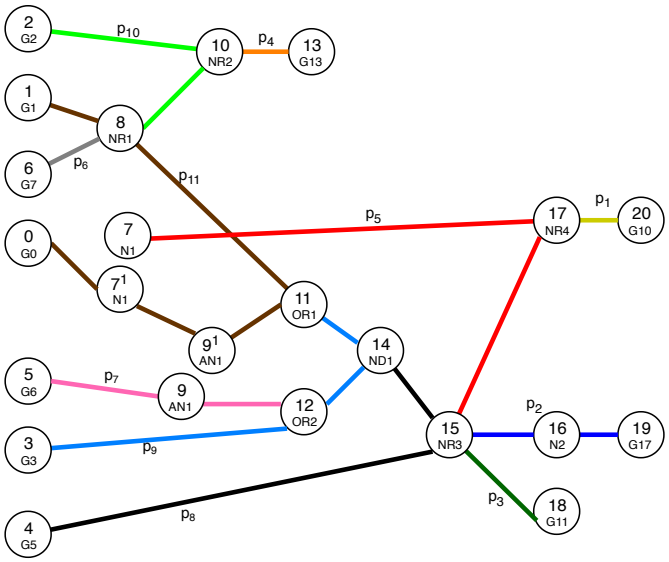
Edge-Sequence	Vertex labelling	Prüfer code
$e_1 - e_2 - e_3$		2 3
$e_1 - e_3 - e_2$		3 4
$e_2 - e_1 - e_3$	does not exists	Not possible

2. *The vertex-labels of T determine the nature of the Prüfer code.*
   This is an important property and follows from the above discussion of Property 1.
3. *T can be encoded to a set, S_T, of distinct Prüfer codes.*
   From Property 2, every relabeling of T furnishes a Prüfer code unique to that labeling. Thus, given an unlabeled T, it can be represented by several Prüfer codes that make up the set S_T. The number of such codes is determined by the number of unique labeling of the vertices of T.
4. *Every Prüfer code in S_T decodes to the tree structure of T.*
   On applying the decoding algorithm on a Prüfer code in S_T, the structure of T is reconstructed.
5. *Prüfer sequences induce a partition on S which is the set of Prüfer codes of labeled trees with n vertices.*
   The Cayley's formula, n^{n-2}, gives the number of labeled trees with n vertices [Sho95]. Prüfer codes provide a bijective proof of Cayley's formula and so the size of $|S|$ is exactly equal to the number of labeled trees with n vertices. For a given n, the number of unlabeled trees is less than the size of S because there exist several labeled trees for a given unlabeled tree. From Property 3, each unlabeled tree T, of size n can be represented by a Prüfer code in S_T. Also, from Property 4, each of them uniquely reconstructs the original tree T. Hence, they are mutually exclusive. Furthermore, since every Prüfer code among the n^{n-2} codes reconstructs to some tree of size n. The union of the sets S_T for all unlabeled trees is thus collectively exhaustive. Therefore, they induce a partition.

5.6.2 Encoding Methods

We have seen how the tree $T_g(V^T, E^T)$ can be obtained by invoking by SENSOR on the graph $G(V, E)$. We select any code (Property 3 and Property 4) from the partition S_T (Property 5) to represent the structure of T_g. Here we present three codes that are generated based on graph relabeling, label reordering, and tree relabeling.

1. **Direction-centric code (DCC).** For directed acyclic graphs (DAG), a topological ordering of vertices preserves the direction of the edges. So, in this method, the vertices of a graph are labeled in a topological order. Each pair of vertices (v_1, v_2), with the edge directed from v_1 to v_2, is labeled such that label of v_1 is smaller than that of v_2. Since combinatorial logic networks are represented as DAGs, DCC completely preserves the structure of logic networks.
2. **Path-centric code (PCC).** In PCC, the labeling scheme is same as DCC, so it preserves the direction of the edges. While in DCC, the edges are not directly interpretable, in PCC, we make the edges interpretable by introducing a change in the ordering of vertices keeping their labels intact. We assume that the vertex-labels also determine



(a) Path partitions of the graph for circuit s27.

20 19 18 13 7 6 5 4 3 2 1 **0**

(b) Sequence of single pendant vertices derived from the above code

p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}
17 16	**15**	**15**	**10**	17	**15**	**8**	9	**12** 15	**14** 12 14	**11** 10	**8** 8 11	9^1	7^1

(c) Prüfer code

Figure 5.10: Example of PCC for *g-tree* of s27.

their ordering, smaller label meaning higher order. We relax the assumption only for the leaf nodes of T_g. Let the set of leaf nodes of T_g be V^T_{leaf}. Here, we assume that the order of vertices in V^T_{leaf} is higher than those of non-leaf vertices ($V^T \setminus V^T_{leaf}$). When this constraint in ordering is satisfied, the vertices in the set $V^T \setminus V^T_{leaf}$ that appear in *Pen_List* are processed before those in V^T_{leaf} during encoding/decoding (Refer Section 5.6.1 under Property 1). This has the following implication: For a leaf vertex u_{leaf} of T_g we define two terms: (i) *junction point* (u_m), the first vertex, of degree greater than two, reachable from u_{leaf}, (ii) *path-vertices*, the set of two-degree vertices that lie between u_{leaf} and its junction point. In the Prüfer code, the path-vertices appear consecutively, in the same sequence as in the path, $u_{leaf} \rightsquigarrow u_m$, followed by the junction vertex u_m. Thus, each of the leaf node, except the last one, induces a path. Also, the order of paths follows that of the corresponding leaf-nodes.

The edges can be easily reconstructed from their labels if we mark the vertices corresponding to the junction points. They can be marked by traversing the code from right-to-left, and marking those vertex labels which have already appeared (Observation 7).

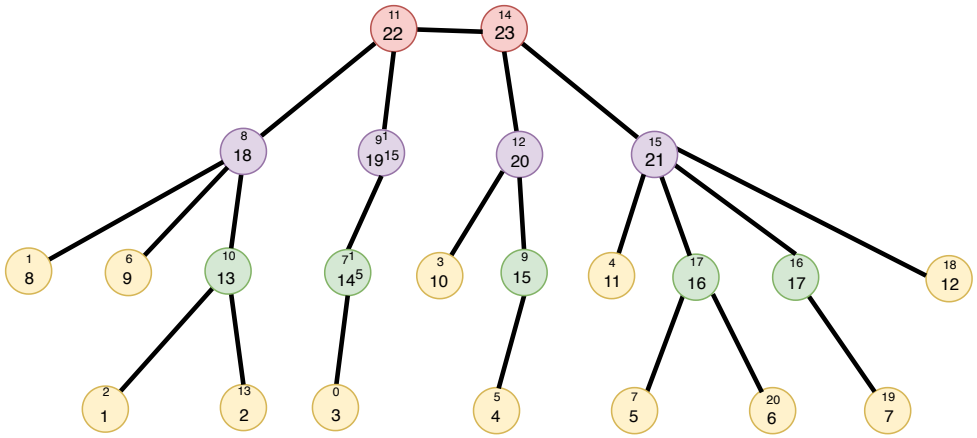
An example of PCC for circuit s27 is given in Figure 5.10. The tree along with eleven paths is shown in Figure 5.10a. Note that the vertices in V^T_{leaf} are considered in reverse order; higher label means higher order (Figure 5.10b). The Prüfer code with marked junction points is shown in Figure 5.10c. For example, the leaf-node 3 induces Path p_9, whose path-vertices {12,14} and vertex at junction point, vertex labeled 11 appear sequentially in the code.

3. **Leaf-centric code (LCC).** Although the edges can be reconstructed directly from PCC, the leaf-node needs to be computed beforehand. LCC is a fully edge-interpretable code. The index of a label gives the label of its adjacent vertex. This is possible because each vertex-label in the code represents an edge (Property 1). This is accomplished by complete relabeling of T_g, denoted by T^1_{lcc}, which is done as follows: The nodes are partitioned into sets called *leaf-stage*. The leaves of T_g are assigned to the set representing the first stage, $LS1$. Next, on removing the nodes in leaf-stage-1, the leaves of the new tree T^2_{lcc} are assigned to $LS2$. Iteratively, the partitions are formed until the entire tree is processed. The final leaf-stage, LSk, consists of either a node or a pair of nodes, and is called the *tree center* [Har71]. The labeling is done iteratively as follows. Vertices of $LS1$ are labeled from 1 to $|LS1|$. Iteratively, the vertices in k^{th} set is labeled $\sum_1^k |LS(i-1)|+1$ to $\sum_1^k |LSi|$. Consider a set $REP \subset V^T$, where the vertices in REP were formed by splitting a particular vertex in G. Let their labels be $l_1 \cdots l_m$. Let $v \in REP$ have the smallest label (l'_m). For each vertex u in REP, with label l_u, their label is modified as $l_u^{l'_m}$. The additional index is used to store the replica information and does not affect the ordering of vertices while encoding. Such labeling strategy will enforce the label and its index to preserve an edge. Also, the edges belonging to a leaf-stage appear consecutively

in LCC.

There are scope for further improving the interpretability of LCC by adjusting the labels within each leaf-stage. The labels in a leaf-stage subsequence can be made to appear in an ascending order. Considering the tree to be rooted at its center, the vertices can be assigned to different levels progressively. The labeling starts from the vertex (vertices) in the root (level 0). The first vertex is labeled n and subsequent labeling is processed in a decremental order. The labeling is done while moving from lower-level to higher-level nodes, while prioritizing them based on the leaf-stage (higher leaf-stage first). The vertices within a level and belonging to the same leaf-stage are prioritized based on the label of their parents (vertex with higher-labeled parent first).

Lastly, the direction of the edges can be encoded by marking the labels which represent the edges that are directed one way (converging or diverging) and leaving the edges in the opposite direction unmarked. Thus, besides DAG any directed graph can be encoded.



(a) Relabeled tree of s27 for LCC. The original label is shown in small font. Leaf-stage of vertices is color coded. Vertices with the same level appear horizontally.

13	13	14^5	15	16	16	17	18	18	20	21	21	18	19^{15}	20	21	21	22	22	23	23
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

(b) LCC and the index

Figure 5.11: Example of LCC for *g-tree* of s27.

An example of LCC for s27 is given in Figure 5.11. Figure 5.11a shows the tree where the root is the edge with vertices marked in red. The new labels are shown in the Figure 5.11a. Also, the labeling of vertices within each stage is taken care of so as to improve interpretability. This is reflected in the LCC given in Figure 5.11b. Here, the label of

the vertex with the corresponding edge incident on it, is marked in orange and one with corresponding edge divergent from it is marked in blue.

5.6.3 Learnable Representation

Data samples are commonly represented as vectors not only in statistical inference but also in ML, where they are called feature vector. Image data can be viewed as vectors since they are regularly structured on a rectangular lattice. Such representation offers computational ease and lends strong mathematical foundation [FM12] where huge repository of operations have been defined for vectors. Thus, there are numerous algorithms available that can be used for analyzing vector data in ML. Graphs, however, comprise highly unstructured data and hence cannot naturally be represented as vectors. Although an adjacency matrix can be viewed as structured data, it has several issues which render it difficult to be used as feature vectors. Firstly, for large graphs, the size ($|V|^2$) is too large for implementation. Secondly, in scenarios such as graph classification where a single graph is a data sample, it required that the graphs be of similar size. Lastly, even if they are of same size, we need a method to order the nodes so as to induce some correspondence among the nodes of different sample graphs.

Graph embedding has presently emerged as a common solution to above mentioned problems. It captures a certain properties of the graph in the form of vectors. There are of two kinds: (i) node embedding, where each node is represented by a vector. They are used mostly for node classification and link prediction of a graph. They are mostly dependent on the neighborhood information and first/second degree proximity. Some of the examples are Node2Vec [GL16], LINE [TQW+15], DeepWalk [PARS14]; (ii) whole graph embedding. Here the entire graph is represented by a vector which captures some of its properties. They are mostly applicable for graph classification. Such related work includes graph kernels [YV15] and Subgraph2Vec [NCC+16].

Since the aim of our representation is to capture the structural properties of the entire circuit-graph, it needs whole-graph type embedding. However, the existing methods capture only abstracted properties of graph. For example, graphlet-based kernels represent the counts of different kinds of graphlets as a vector. More than just graph classification, we aim to learn structural features from the graph. The proposed Prüfer code captures the structure of the entire graph, and it can handle the scalability issue mentioned earlier. However, the other two issues need to be addressed to make it suitable for graph embedding. The second issue can be handled if the assumption on the size of the graphs (the number of edges in this case), holds, i.e., they are of same order. For graphs representing logical circuits, we can synthesize similar-sized circuits to form the training data. Any difference in their sizes can be handled augmenting them with pseudo-vertices/edges as in [NAK16]. For node correspondence, ordering of the nodes can be accomplished using some special

properties [NAK16] such as node-degree. Besides that, other properties of the circuit such as level or depth (discussed in Chapter 4) can also be used.

Once we obtain an ordering of the vertices of the graph $G(V, E)$, we can follow this ordering while traversing the graph following SENSOR in order to obtain the tree $T_g(V^T, E^T)$. Hence, T_g will be a unique tree of the graph based on vertex ordering. Since the topological ordering of a DAG is not unique, in the case of DCC or PCC, the independent vertices that have the same level in the graph can be labeled following the ordering of nodes. Also, in the case of LCC, the ordering can be followed for the vertices, which belong to the same tree-level, leaf-stage, or have the same parent.

5.7 Conclusion and Future Work

In this work, we have demonstrated a proof-of-concept for lossless and compact encoding of large graphs using just a linear-size sequence of vertex labels. In order to encode graphs with Prüfer codes, we have proposed a method called $\mathcal{GT}$-enhancement, to represent a graph by a tree. We have proposed two techniques for $\mathcal{GT}$-enhancement, and the second technique called SENSOR, allows such tree to be represented by a single code. The focus of the encoding is to capture the structural property of a graph representing a digital circuit, which is essentially a DAG. Among various graph-representation methods used for machine-learning framework, a major issue is to ensure lossless encoding of the structure of the graph. The method based on Prüfer codes completely overcomes this issue. Moreover, we have proposed a labeling technique that preserves the direction of the edges and improves their interpretability. We have discussed the learnability of the code and how they can be used for graph embedding. Application of this code to facilitate the prediction of X-sensitivity in logic circuits is left as future work. In general, a learnable representation of the code can potentially be applied for geometric deep learning. Additionally Prüfer code offers a 1-D representation of the graph, which is often a requirement for geometric deep learning [BBL+17]. Besides, in the case of *g-trees* generated from graphs with ordered vertices, the interpretable codes such as LCC bring certain structuredness in the representation, which is required for such applications. Use of such codes in deep learning framework like RNN or CNN can be explored in the future.

CHAPTER

SIX

CONCLUSIONS AND FUTURE WORK

In this thesis, we report combinatorial and machine learning based techniques to address some of the challenges in the field of logic test and diagnosis. These problems arise as outcome of large scale integration and process complexity. While this has led to new levels of complex integration on the product end, it demands intricate process technology at the manufacturing end.

The first problem we studied is related to fault diagnosis. This refers to the process of defect localization in failed chips, which, in turn, is needed to improve the manufacturing process. A critical factor that determines the accuracy of fault diagnosis is the power of diagnostic test patterns. The aim of diagnostic test pattern generation is to distinguish as many faults pairs as possible. We presented a combinatorial solution to this problem. While the previous approaches employed circuit analysis or circuit modification techniques, our approach is based on a novel concept which only relies on ATPG and fault simulation tools and do not require any explicit interaction with the circuit. Based on the experimental observation that the test sets produced by ATPG tools are diverse, we proposed a combinatorial test-selection method for growing the diagnostic test set. Experimental results show that such a method is equally good at distinguishing the fault pairs like previous methods and the diagnostic test set size is also comparable.

Our second problem is concerned with the study of X-sensitivity of circuit-inputs. An inevitable consequence of the complexity of integration in IC-chips is the presence of unknown logic values (X) in the circuit. Unknown values severely hamper the fault coverage and test cost. The present solutions to this problem incur various hardware and time overhead. We have built a predictor, based on support vector regressor, that estimates the impact of the X-sources on the fault coverage of the test set of any given circuit. We showed that such ML-based predictor can be used to grade the X-sources of a circuit conveniently so that the X-sources can be prioritized before handling them in order to avoid unnecessary

overhead. While an ATPG tool could also fulfill this, it would have taken huge time. In contrast, the proposed predictor just needs a function evaluation for such estimate, and so the grading can be obtained instantly. A noteworthy aspect of our predictor is that, it is based solely on the structural features of the circuit network. These have been formulated so as to take care of the variations in circuit size and structural diversity of different circuits. The predictor can thus be generalized for a large class of circuits.

While circuit networks provide a rich source of data, the need to mine features from such graphs has to be addressed first before they may be fed to machine learning tools. In our third problem, we present a compact and lossless encoding of graphs in order to make them readable by machine learning tools. Our method is based on Prüfer sequences which encode the structure of a tree. In order to apply them to general graphs, we have proposed a technique to modify a graph to a tree (*g-tree*), called $\mathcal{GT}$-enhancement. We have proposed two methods based on this technique. The second method, SENSOR, forms a *g-tree* which can be fully encoded into a single Prüfer sequence. We have shown that it is possible to preserve the direction (signal flow) of the graph.

As a future work, it would be interesting to study the Prüfer encoding of graphs in the setting of predicting the impact of X-sources. While only single X-source has been considered in our work, its extension to multiple X-sources using the graph-encoding as input, can be explored. From the survey reported in Chapter 2, it was observed that machine learning is being adopted as a potential solution in this field. This is because of the availability of data in profusion and variety and also because the errors due to new defects are becoming more and more probabilistic. So, the application of encoding to other problem settings can be explored. Fault diagnosis, especially multiple fault diagnosis, would also be a potential area of future ML-research since algorithmic approaches are difficult. Lastly, each of the methods proposed in this thesis can be applied to large industrial circuits by more efficient implementation of the response matrix for DTS generation and by considering multiple X-sources for X-sensitivity prediction.

BIBLIOGRAPHY

[AA13] Chidambaram Alagappan and Vishwani D. Agrawal. Defect diagnosis of digital circuits using surrogate faults. In *Proc. VDAT*, pages 376–386, 2013.

[ABF02] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman. *Digital Systems Testing and Testable Design*. IEEE Press, NJ, 2002.

[ABF07] H. Altun, A. Bilgil, and B.C. Fidan. Treatment of multi-dimensional data to enhance neural network estimators in regression problems. *Expert Systems with Applications*, 32(2):599 – 605, 2007.

[ABKS03] V.D. Agrawal, Dong Hyun Baik, Yong Chang Kim, and K.K. Saluja. Exclusive test and its applications to fault diagnosis. In *Proc. VLSI Design*, pages 143–148, 2003.

[AFI06] M. Arai, S. Fukumoto, and K. Iwasaki. Expansion of convolutional compactors over Galois field. In *Proc. ATS*, pages 401–408, Nov 2006.

[AFPB03] M.E. Amyeen, W.K. Fuchs, I. Pomeranz, and V. Boppana. Fault equivalence identification in combinational circuits using implication and evaluation techniques. *IEEE Trans. CAD*, 22(7):922–936, July 2003.

[Ait12] R. Aitken. Yield learning perspectives. *IEEE DTC*, 29(1):59–62, Feb 2012.

[Alb05] C. Albrecht. IWLS 2005 benchmarks. In *International Workshop on Logic Synthesis*, June 2005.

[APA03] V. D. Agrawal, A. V. S. S. Prasad, and M. V. Atre. Fault collapsing via functional dominance. In *Proc. ITC*, pages 274–280, Sept 2003.

[BA05] M. Bushnell and Vishwani Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Publishing Company, Incorporated, Dordrecht, Netherlands, 2005.

[BB07] Therese Biedl and Franz J. Brandenburg. Partitions of graphs into trees. In *Proc. Graph Drawing*, pages 430–439, 2007.

[BBK89] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Proc. ISCAS*, pages 1929–1934, 1989.

[BBL+17] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, July 2017.

[BCV13] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE PAMI*, 35(8):1798–1828, Aug 2013.

[Ben09] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[BF85] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *Proc. ISCAS*, 1985.

[BM17] Mohammad Bataineh and Timothy Marler. Neural network for regression problems with reduced training sets. *Neural Networks*, 95:1 – 9, 2017.

[BND16] K. M. Butler, A. Nahar, and W. R. Daasch. What we know after twelve years developing and deploying test data analytics solutions. In *Proc. ITC*, pages 1–8, Nov 2016.

[BS13] Vijay Bhargava and Harkaran Singh. Handling X-bounding in LBIST designs. https://www.edn.com/design/integrated-circuit-design/4418773/Handling-X-bounding-in-LBIST-designs, July 2013.

[CA87] H. P. Chang and J. A. Abraham. The complexity of accurate logic simulation. In *Proc. ICCAD*, pages 404–407, 1987.

[CH06] Samprit Chatterjee and Ali S. Hadi. *Regression Analysis by Example*. Wiley, Hoboken, New Jersey, 2006.

[CHJC13] S. Y. Chen, M. Y. Hsiao, W. B. Jone, and T. F. Chen. A configurable bus-tracer for error reproduction in post-silicon validation. In *Proc. VLSI-DAT*, pages 1–4, April 2013.

[CJC13] H. Y. Chang, I. H. R. Jiang, and Y. W. Chang. ECO optimization using metal-configurable gate-array spare cells. *IEEE Trans. CAD*, 32(11):1722–1733, Nov 2013.

[CJC14] H. Y. Chang, I. H. R. Jiang, and Y. W. Chang. Functional ECO using metal-configurable gate-array spare cells. In *Proc. DAC*, pages 1–6, 2014.

[CKSF05] T. Clouqueur, Kamran Zarrineh, K. K. Saluja, and H. Fujiwara. Design and analysis of multiple weight linear compactors of responses containing unknown values. In *Proc. ITC*, pages 10 pp.–1108, Nov 2005.

[CLH+19] Mason Chern, Shih-Wei Lee, Shi-Yu Huang, Yu Huang, Gaurav Veda, Kun-Han (Hans) Tsai, and Wu-Tung Cheng. Improving scan chain diagnostic accuracy using multi-stage artificial neural networks. In *Proc. ASPDAC*, pages 341–346, New York, NY, USA, 2019. ACM.

[CM04] Vladimir Cherkassky and Yunqian Ma. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, 17(1):113 – 126, 2004.

[CP89] S. J. Chandra and J. H. Patel. Accurate logic simulation in the presence of unknowns. In *Proc. ICCAD*, pages 34–37, Nov 1989.

[CRS00] F. Corno, M.S. Reorda, and G. Squillero. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Design and Test of Computers*, 17(3):44–53, Jul 2000.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, Cambridge, 2nd edition, 2001.

[CTR17] W. Cheng, Yue Tian, and S. M. Reddy. Volume diagnosis data mining. In *Proc. ETS*, pages 1–10, May 2017.

[CZC18] H. Cai, V. W. Zheng, and K. C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, Sept 2018.

[DB17] Y. Dai and R. K. Braytont. Circuit recognition with deep learning. In *Proc. HOST*, pages 162–162, 2017.

[DBV16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proc. NIPS*, pages 3844–3852. Curran Associates, Inc., 2016.

[DCM12] G. Geoffrey Vining Douglas C. Montgomery, Elizabeth A. Peck. *Introduction to Linear Regression Analysis*. Wiley, New York, NY, USA, 5th edition, 2012.

[DDS92] Xiang Dong, Wei Daozheng, and Chen Shisong. Probabilistic models for estimation of random and pseudo-random test length. *Journal of Computer Science and Technology*, 7(2):164–174, Apr. 1992.

[DED+17] H. Dhotre, S. Eggersgl, M. Dehbashi, U. Pfannkuchen, and R. Drechsler. Machine learning based test pattern analysis for localizing critical power activity areas. In *Proc. International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, Oct 2017.

[DED19] H. Dhotre, S. Eggersgl, and R. Drechsler. Cluster-based localization of ir-drop in test application considering parasitic elements. In *Proc. LATS*, pages 1–4, March 2019.

[DG77] Persi Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society, Series B (Methodological)*, 39(2):262–268, 1977.

[DG84] William R. Dillon and Matthew Goldstein. *Multivariate Analysis Methods and Applications*. John Wiley & Son, New York, 1984.

[Dom12] Pedro Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, October 2012.

[ea11] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[EKR+15] D. Erb, M. A. Kochte, S. Reimer, M. Sauer, H. J. Wunderlich, and B. Becker. Accurate QBF-based test pattern generation in presence of unknown values. *IEEE Trans. CAD*, 34(12):2025–2038, Dec 2015.

[EKS+13] D. Erb, M. A. Kochte, M. Sauer, H. Wunderlich, and B. Becker. Accurate multi-cycle atpg in presence of x-values. In *Proc. ATS*, pages 245–250, Nov 2013.

[EKS+14] Dominik Erb, Michael A. Kochte, Matthias Sauer, Stefan Hillebrecht, Tobias Schubert, Hans-Joachim Wunderlich, and Bernd Becker. Exact logic and fault simulation in presence of unknowns. *ACM Trans. Des. Aut. Electr. Sys.*, 19(3):28:1–28:17, June 2014.

[ET93] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, New York, 1993.

[FKS03] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top k lists. In *Proc. SODA*, pages 28–36, 2003.

[FM12] Yun Fu and Yunqian Ma. *Graph Embedding for Pattern Analysis*. Springer Publishing Company, Incorporated, New York, 2012.

[FS83] Fujiwara and Shimono. On the acceleration of test generation algorithms. *IEEE TOC*, C-32(12):1137–1144, Dec 1983.

[GCI+17] L. Rodrguez Gomez, A. Cook, T. Indlekofer, S. Hellebrand, and H.J. Wunderlich. Adaptive bayesian diagnosis of intermittent faults. *JETTA*, 30(5):527–540, 2017.

[GJRR01] Jens Gottlieb, Bryant A. Julstrom, Günther R. Raidl, and Franz Rothlauf. Prüfer numbers: A poor representation of spanning trees for evolutionary search. In *Proc. GECCO*, pages 343–350, 2001.

[GL16] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proc. KDD*, pages 855–864. ACM, 2016.

[GMK91] T. Gruning, U. Mahlstedt, and H. Koopmeiners. DIATEST: A fast diagnostic test pattern generator for combinational circuits. In *Proc. ICCAD*, pages 194–197, Nov 1991.

[Goe17] Richard Goering. How metal-only ECOs save full silicon respins. https://community.cadence.com/cadence_blogs_8/b/ii/archive/2010/11/23/user-interview-how-metal-only-ecos-save-full-silicon-respins, April 2017.

[Gro06] Ian A. Grout. *Automatic Test Equipment (ATE) and Production Test*, pages 257–266. Springer London, London, 2006.

[GT80] L. H. Goldstein and E. L. Thigpen. SCOAP: Sandia controllability/observability analysis program. In *Proc. DAC*, pages 190–196, June 1980.

[GW16] L. R. Gmez and H. Wunderlich. A neural-network-based fault classifier. In *Proc. ATS*, pages 144–149, Nov 2016.

[Har71] F. Harary. *Graph Theory*. Addison Wesley Series in Mathematics. Addison-Wesley, Boston, 1971.

[Hat17] A. A. Hatzopoulos. Analog circuit testing. In *Proc. IMSTW*, pages 1–6, July 2017.

[HBK+17] Y. Huang, B. Benware, R. Klingenberg, H. Tang, J. Dsouza, and W. Cheng. Scan chain diagnosis based on unsupervised machine learning. In *Proc. ATS*, pages 225–230, Nov 2017.

[hCMB08] Kai hui Chang, Igor L. Markov, and Valeria Bertacco. Reap what you sow: Spare cells for post-silicon metal fix. In *Proc. ISPD*, pages 103–110, 2008.

[HCS+18] W. Haaswijk, E. Collins, B. Seguin, M. Soeken, F. Kaplan, S. Ssstrunk, and G. De Micheli. Deep learning for logic optimization algorithms. In *Proc ISCAS*, pages 1–4, 2018.

[HFMB18] Q. Huang, C. Fang, S. Mittal, and R. D. S. Blanton. Improving diagnosis efficiency via machine learning. In *Proc. ITC*, pages 1–10, Oct 2018.

[HGCL08] Y. Huang, R. Guo, W. Cheng, and J. C. Li. Survey of scan chain diagnosis. *IEEE DTC*, 25(3):240–248, May 2008.

[HKP04] L. M. Huisman, M. Kassab, and L. Pastel. Data mining integrated circuit fails with fail commonalities. In *Proc. ITC*, Oct 2004.

[HKWB12] S. Hillebrecht, M. A. Kochte, H. Wunderlich, and B. Becker. Exact stuck-at fault classification in presence of unknowns. In *Proc. ETS*, pages 1–6, May 2012.

[HSEL02] C. Hora, R. Segers, S. Eichenberger, and M. Lousberg. An effective diagnosis method to support yield improvement. In *Proc. ITC*, pages 260–269, Oct 2002.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, New York, 2nd edition, 2009.

[HYL17] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.

[J-2] TetraMAX (TM) J-2014. Synopsys advanced pattern generation. https://www.synopsys.com/implementation-and-signoff/test-automation/testmax-atpg.html.

[JM15] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[JYZ+16] S. Jin, F. Ye, Z. Zhang, K. Chakrabarty, and X. Gu. Efficient board-level functional fault diagnosis with missing syndromes. *IEEE TCAD*, 35(6):985–998, June 2016.

[Kah62] A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, Nov. 1962.

[KJC+14] Subhadip Kundu, Aniket Jha, Santanu Chattopadhyay, Indranil Sengupta, and Rohit Kapur. Framework for multiple-fault diagnosis based on multiple fault simulation using particle swarm optimization. *IEEE Trans. VLSI Syst.*, 22(3):696–700, 2014.

[KKH11] Mouna Karmani, Chiraz Khedhiri, and Belgacem Hamdi. Design and test challenges in nano-scale analog and mixed cmos technology. *VLSICS*, 2, June 2011.

[KKM+11] M. A. Kochte, S. Kundu, K. Miyase, X. Wen, and H. Wunderlich. Efficient BDD-based fault simulation in presence of unknown values. In *Proc. ATS*, pages 383–388, Nov 2011.

[KMG04] S. Kundu, T. M. Mak, and R. Galivanche. Trends in manufacturing test methods and their implications. In *Proc. ITC*, pages 679–687, Oct 2004.

[KPC+13] S. Kundu, S. Pal, S. Chattopadhyay, I. Sengupta, and R. Kapur. A metric for test set characterization and customization toward fault diagnosis. *IEEE TCAD*, 32(11):1824–1828, Nov 2013.

[KSR04] Seiji Kajihara, Kewal K. Saluja, and Sudhakar M. Reddy. Enhanced 3-valued logic/fault simulation for full scan circuits using implicit logic values. In *Proc. ETS*, pages 108–113, 2004.

[LAP19] MARK LAPEDUS. Finding defects in chips with machine learning. https://semiengineering.com/finding-defects-with-machine-learning/, MARCH 2019.

[LCP+17] 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. VTS*, pages 1–6, April 2017.

[LGEC17] A. LHeureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz. Machine learning with big data: Challenges and approaches. *IEEE Access*, 5:7776–7797, 2017.

[LH93] H.K. Lee and D.S. Ha. ATALANTA: An efficient ATPG for combinational circuits. In *Technical Report*, pages 93–12, Dept of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1993.

[LH96] Hyung Ki Lee and Dong Sam Ha. HOPE: an efficient parallel fault simulator for synchronous sequential circuits. *IEEE Trans. CAD*, 15(9):1048–1058, Sep 1996.

[LHCL13] Shih-Yuan Liu, Ying-Chin Hou, Chih-Chung Chang, and Jian-Chang Lin. Sige profile inspection by using dual beam fib system in physical failure analysis. In *Proc. IPFA*, pages 490–492, July 2013.

[LHF+12] D. Lin, T. Hong, F. Fallah, N. Hakim, and S. Mitra. Quick detection of difficult bugs for effective post-silicon validation. In *Proc. DAC*, pages 561–566, 2012.

[LHLL17] Y. Liu, C. Han, S. Lin, and J. C. Li. PSN-aware circuit test timing prediction using machine learning. *IET Computers Digital Techniques*, 11(2):60–67, 2017.

[LLC07] Yung-Chieh Lin, Feng Lu, and Kwang-Ting Cheng. Multiple-fault diagnosis based on adaptive diagnostic test pattern generation. *IEEE Trans. CAD*, 26(5):932–942, May 2007.

[LLEP07] A. Larsson, E. Larsson, P. Eles, and Z. Peng. Optimized integration of test compression and sharing for soc testing. In *Proc. DATE*, pages 1–6, April 2007.

[LLH11] Kuen-Jong Lee, Wei-Cheng Lien, and Tong-Yu Hsieh. Test response compaction via output bit selection. *IEEE Trans. CAD*, 30(10):1534–1544, Oct 2011.

[LLH+13] Wei-Cheng Lien, Kuen-Jong Lee, Tong-Yu Hsieh, K. Chakrabarty, and Yu-Hua Wu. Counter-based output selection for test response compaction. *IEEE Trans. CAD*, 32(1):152–164, Jan 2013.

[MC71] E. J. McCluskey and F. W. Clegg. Fault equivalence in combinational logic networks. *IEEE Transactions on Computers*, C-20(11):1286–1293, Nov 1971.

[MGBK12] Deepak Mahajan, Aniruddha Gupta, Deepak Kumar Behera, and Narender Kaushik. Understanding the concept of X in SOC design flow. https://www.edn.com/design/integrated-circuit-design/4392245/2/Understanding-the-concept-of-X-in-SOC-design-flow-, August 2012.

[MGOD90] U. Mahlstedt, T. Gruning, C. Ozcan, and W. Daehn. CONTEST: A fast ATPG tool for very large combinational circuits. In *Proc. ICCAD*, pages 222–225, Nov 1990.

[Mil98] L. S. Milor. A tutorial introduction to research on analog and mixed-signal circuit testing. *IEEE TCAD*, 45(10):1389–1407, Oct 1998.

[MK02] S. Mitra and Kee Sup Kim. *X*-compact: an efficient response compaction technique for test cost reduction. In *Proc. ITC*, pages 311–320, 2002.

[MRK+19] Yuzhe Ma, Haoxing Ren, Brucek Khailany, Harbinder Sikka, Lijuan Luo, Karthikeyan Natarajan, and Bei Yu. High performance graph convolutional networks with applications in testability analysis. In *Proc. DAC*, pages 1–6, Jun 2019.

[MS08] V.C. Mushirabad and R. Shettigara. Automatic fault-testing of logic blocks using internal at-speed logic-BIST, July 2008. *US Patent* 7,398,443.

[Mut76] Peter Muth. A nine-valued circuit model for test generation. *IEEE TOC*, C-25(6):630–636, June 1976.

[Mut14] Ann Steffora Mutschler. Yield ramp challenges increase. https://semiengineering.com/yield-ramp-challenges-increase/, DECEMBER 2014.

[NAK16] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proc. ICML*, pages 2014–2023, 2016.

[NCC+16] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *CoRR*, abs/1606.08928, 2016.

[NPRK03] M. Naruse, I. Porneranz, S. M. Reddy, and S. Kundu. On-chip compression of output responses with unknown values using lfsr reseeding. In *Proc. ITC*, volume 1, pages 1060–1068, Sep. 2003.

[NTB10] J. E. Nelson, W. C. Tam, and R. D. Blanton. Automatic classification of bridge defects. In *Proc. ITC*, pages 1–10, Nov 2010.

[NZD+06] J. E. Nelson, T. Zanon, R. Desineni, J. G. Brown, N. Patil, W. Maly, and R. D. Blanton. Extraction of defect density and size distributions from wafer sort test results. In *Proc. DATE*, volume 1, pages 1–6, March 2006.

[OME05] C. O'Farrill, M. Moakil-Chbany, and B. Eklow. Optimized reasoning-based diagnosis for non-random, board-level, production defects. In *Proc. ITC*, pages 7–179, Nov 2005.

[OSK+10] M. P. Ooi, E. Kwang Joo Sim, Y. C. Kuang, L. Kleeman, C. Chan, and S. Demidenko. Automatic defect cluster extraction for semiconductor wafers. In *Proc. IMTCP*, pages 1024–1029, May 2010.

[OY18] Ankush Oberai and Jiann-Shiun Yuan. Efficient fault localization and failure analysis techniques for improving ic yield. *Electronics*, 7(3), 2018.

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proc. KDD*, pages 701–710. ACM, 2014.

[PBCB18] 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 TCAD*, page (Accepted), 2018.

[Pei14] Tiago P. Peixoto. The graph-tool python library. http://figshare.com/articles/graph_tool/1164194, 2014.

[PKR02] I. Pomeranz, S. Kundu, and S. M. Reddy. On output response compression in the presence of unknown output values. In *Proc. DAC*, pages 255–258, 2002.

[Pom14a] I. Pomeranz. OBO: An output-by-output scoring algorithm for fault diagnosis. In *Proc. IEEE Computer Society Annual Symposium on VLSI*, pages 314–319, July 2014.

[Pom14b] I. Pomeranz. Unknown output values of faulty circuits and output response compaction. *IEEE Trans. CAD*, 33(2):323–327, 2014.

[Pom15] I. Pomeranz. Improving the accuracy of defect diagnosis by considering reduced diagnostic information. In *Proc. VTS*, pages 1–6, 2015.

[Pom16] Irith Pomeranz. A test selection procedure for improving the accuracy of defect diagnosis. *IEEE Trans. VLSI Syst.*, 24(8):2759–2767, 2016.

[Pom19] I. Pomeranz. Diagnostic test generation that addresses diagnostic holes. *IEEE TCAD*, 38(2):335–344, Feb 2019.

[PR98] I. Pomeranz and S.M. Reddy. A diagnostic test generation procedure for synchronous sequential circuits based on test elimination. In *Proc. ITC*, pages 1074–1083, Oct 1998.

[PR07a] I. Pomeranz and S.M. Reddy. Diagnostic test generation based on subsets of faults. In *Proc. ETS*, pages 151–158, 2007.

[PR07b] I. Pomeranz and S.M. Reddy. Equivalence and dominance relations between fault pairs and their use in fault pair collapsing for fault diagnosis. In *Proc. VLSI Design*, pages 498–503, Jan 2007.

[Pru18] H. Prufer. Neuer beweis eines satzes uber permutationen. *Arch. Math. Phys.*, 27:742–744, 1918.

[PVRS04] I. Pomeranz, S. Venkataraman, S.M. Reddy, and B. Seshadri. Z-sets and z-detections: circuit characteristics that simplify fault diagnosis. In *Proc. DATE*, pages 68–73, 2004.

[RM14] Lior Rokach and Oded Maimon. *Data Mining With Decision Trees: Theory and Applications*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2nd edition, 2014.

[Rot66] J. P. Roth. Diagnosis of automata failures: A calculus and a method. *IBM Journal of Research and Development*, 10(4):278–291, July 1966.

[RSRB15] Andreas Riefert, Matthias Sauer, Sudhakar Reddy, and Bernd Becker. Improving diagnosis resolution of a fault detection test set. In *Proc. VTS*, pages 1–6, 2015.

[RT05] J. Rajski and J. Tyszer. Synthesis of X-tolerant convolutional compactors. In *Proc. VTS*, pages 114–119, May 2005.

[RTWR03] J. Rajski, J. Tyszer, Chen Wang, and S. M. Reddy. Convolutional compaction of test responses. In *Proc. ITC*, volume 1, pages 745–754, Sept 2003.

[SA88] M. H. Schulz and E. Auth. Advanced automatic test pattern generation and redundancy identification techniques. In *Proc. ISFTC*, pages 30–35, June 1988.

[SBA86] S. C. Seth, B. B. Bhattacharya, and V. D. Agrawal. An exact analysis for efficient computation of random pattern testability in combinational circuits. In *Digest of Papers, International Symposium on Fault-Tolerant Computing Systems (FTCS)*, pages 318–323, July 1986.

[SBP+17] C. Shan, P. Babighian, Y. Pan, J. Carulli, and L. Wang. Systematic defect detection methodology for volume diagnosis: A data mining perspective. In *Proc. ITC*, pages 1–10, Oct 2017.

[SEB16] K. Scheibler, D. Erb, and B. Becker. Accurate CEGAR-based ATPG in presence of unknown values for large industrial designs. In *Proc. DATE*, pages 972–977, March 2016.

[SGT+09] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Trans. NN*, 20(1):61–80, Jan 2009.

[Sho95] Peter W Shor. A new proof of cayley's formula for counting labeled trees. *Journal of Combinatorial Theory, Series A*, 71(1):154 – 158, 1995.

[SJX+13] 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. ITC*, pages 1–8, Sep. 2013.

[SJX+15] 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. ASP-DAC*, pages 737–742, Jan 2015.

[SK17] Jyotirmoy Saikia and Rohit Kapur. Scheme for masking output of scan chains in test circuit, March 2017. *US Patent* 9,588,179 B2.

[SMO+03] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, Jonathan T. Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res.*, 13(11):2498–2504, Nov 2003.

[SNW17] N. Sumikawa, M. Nero, and L. Wang. Kernel based clustering for quality improvement and excursion detection. In *Proc. ITC*, pages 1–10, Oct 2017.

[SPA85] Sharad Seth, Lilu Pan, and Vishwani Agrawal. PREDICT–Probabilistic estimation of digital circuit testability. In *Digest of Papers, International Symposium on Fault-Tolerant Computing Systems (FTCS)*, pages 220 – 225, June 1985.

[SS04] Alexander J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[Str18] H. Stratigopoulos. Machine learning applications in ic testing. In *Proc. ETS*, pages 1–10, May 2018.

[TCG+11] X. Tang, W. T. Cheng, R. Guo, H. Tang, and S. M. Reddy. Diagnosis of multiple faults based on fault-tuple equivalence tree. In *Proc. DFTS*, pages 217–225, Oct 2011.

[TGP17] D. Tille, B. Gottinger, and U. Pfannkuchen. A lightweight x-masking scheme for iot designs. In *Proc. ITC-Asia*, pages 77–82, Sep. 2017.

[Tip04] Michael E. Tipping. *Bayesian Inference: An Introduction to Principles and Practice in Machine Learning*, pages 41–62. Springer Berlin Heidelberg, 2004.

[TMR+07a] H. Tang, S. Manish, J. Rajski, M. Keim, and B. Benware. Analyzing volume diagnosis results with statistical learning for yield improvement. In *Proc. ETS*, pages 145–150, May 2007.

[TMR+07b] Huaxing Tang, S. Manish, J. Rajski, M. Keim, and B. Benware. Analyzing volume diagnosis results with statistical learning for yield improvement. In *Proc. ETS*, pages 145–150, 2007.

[TQW+15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: large-scale information network embedding. *CoRR*, abs/1503.03578, 2015.

[TSS+14] J. Tikkanen, S. Siatkowski, N. Sumikawa, L. Wang, and M. S. Abadir. Yield optimization using advanced statistical correlation methods. In *Proc. ITC*, pages 1–10, Oct 2014.

[TWE+06] Yuyi Tang, H. J. Wunderlich, Piet Engelke, I. Polian, B. Becker, J. Schloffel, F. Hapke, and M. Wittke. *X*-masking during logic bist and its impact on defect coverage. *IEEE Tran. VLSI*, 14(2):193–202, Feb 2006.

[Vap95] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1st edition, 1995.

[VCAA04] A. Veneris, R. Chang, M.S. Abadir, and M. Amiri. Fault equivalence and diagnostic test generation using ATPG. In *Proc. ISCAS*, pages V–221–V–224, 2004.

[Wan17a] Gang Wang. A novel neural network model specified for representing logical relations. *CoRR*, abs/1708.00580, 2017.

[Wan17b] L. Wang. Experience of data analytics in eda and testprinciples, promises, and challenges. *IEEE TCAD*, 36(6):885–898, June 2017.

[WLL14] Cheng-Hung Wu, Kuen-Jong Lee, and Wei-Cheng Lien. An efficient diagnosis method to deal with multiple fault-pairs simultaneously using a single circuit model. In *Proc. VTS*, pages 1–6, 2014.

[WPY+12] H. Wang, O. Poku, X. Yu, S. Liu, I. Komara, and R. D. Blanton. Test-data volume optimization for diagnosis. In *Proc. DAC*, pages 567–572, June 2012.

[WSRW09] Wei-Che Wang, James C.-M. Lim Yi-Chih Sung, Amy Rao, and Laung-Terng Wang. Test response compaction in the presence of many unknowns. In *Proc. VTTW*, 2009.

[WW09] S. Wang and W. Wei. Machine learning-based volume diagnosis. In *Proc. DATE*, pages 902–905, April 2009.

[WWN08] P. Wohl, J. A. Waicukauski, and F. Neuveux. Increasing scan compression by using X-chains. In *Proc. ITC*, pages 1–10, Oct 2008.

[WWW06] Laung-Terng Wang, Cheng-Wen Wu, and Xiaoqing Wen. *VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, July 2006.

[WWW09] Xiaodong Wang, Lei Wang, and Yingjie Wu. An optimal algorithm for prufer codes. *JSEA*, 2:111–115, 01 2009.

[WWW+10] Zhigang Wang, Laung-Terng Wang, Shianling Wu, Xiaoqing Wen, Boryau Sheu, and Zhigang Jiang. Compacting test responses using X-driven compactor, Aug. 2010. *US Patent* 7,779,322 B1.

[WYL18] Sying-Jyan Wang, Kuan-Ting Yeh, and Katherine Shu-Min Li. Exploiting distribution of unknown values in test responses to optimize test output compactors. *Integration, the VLSI journal*, 2 2018.

[XPLB13] Y. Xue, O. Poku, X. Li, and R. D. Blanton. Padre: Physically-aware diagnostic resolution enhancement. In *Proc. ITC*, pages 1–10, Sept 2013.

[XSRM17] C. Xanthopoulos, P. Sarson, H. Reiter, and Y. Makris. Automated die inking: A pattern recognition-based approach. In *Proc. ITC*, pages 1–6, Oct 2017.

[XST+01] Xinli Gu, Sung Soo Chung, F. Tsang, J. A. Tofte, and H. Rahmanian. An effort-minimized logic bist implementation method. In *Proc. ITC*, pages 1002–1010, Nov 2001.

[YAS02] Yong Chang Kim, V. D. Agrawal, and K. K. Saluja. Multiple faults: modeling, simulation and test. In *Proc. ASP-/DAC/VLSI Design*, pages 592–597, Jan 2002.

[YB08] X. Yu and R. D. Blanton. An effective and flexible multiple defect diagnosis methodology using error propagation analysis. In *Proc. ITC*, pages 1–9, Oct 2008.

[YCZG15] F. Ye, K. Chakrabarty, Z. Zhang, and X. Gu. Self-learning and adaptive board-level functional fault diagnosis. In *Proc. ASP-DAC*, pages 294–301, Jan 2015.

[YFY+16] F. Ye, F. Firouzi, Y. Yang, K. Chakrabarty, and M. B. Tahoori. On-chip droop-induced circuit delay prediction based on support-vector machines. *IEEE TCAD*, 35(4):665–678, April 2016.

[YT12] Joon-Sung Yang and Nur A. Touba. X-canceling MISR architectures for output response compaction with unknown values. *IEEE Trans. CAD*, 31(9):1417–1427, 2012.

[YV15] Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proc. ACM SIGKDD*, pages 1365–1374, New York, NY, USA, 2015. ACM.

[YZCG13] 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 TCAD*, 32(5):723–736, May 2013.

[YZCG14] F. Ye, Z. Zhang, K. Chakrabarty, and X. Gu. Board-level functional fault diagnosis using multikernel support vector machines and incremental learning. *IEEE TCAD*, 33(2):279–290, Feb 2014.

[YZHL10] Jing Ye, Xiaolin Zhang, Yu Hu, and Xiaowei Li. Substantial fault pair at-a-time (SFPAT): An automatic diagnostic pattern generation method. In *Proc. ATS*, pages 192–197, Dec 2010.

[ZA10] Yu Zhang and Vishwani D. Agrawal. A diagnostic test generation system. In *Proc. ITC*, pages 1–9, Nov 2010.

[ZA11] Yu Zhang and V.D. Agrawal. Reduced complexity test generation algorithms for transition fault diagnosis. In *Proc. ICCD*, pages 96–101, Oct 2011.

[ZCW+11] Z. Zhang, K. Chakrabarty, Z. Wang, Z. Wang, and X. Gu. Smart diagnosis: Efficient board-level diagnosis and repair using artificial neural networks. In *Proc. ITC*, pages 1–9, Sept 2011.

[ZGC+18] L. Zhao, S. Goh, Y. Chan, B. Yeoh, H. Hu, M. Thor, A. Tan, and J. Lam. Prediction of electrical and physical failure analysis success using artificial neural networks. In *Proc. IPFA*, pages 1–5, July 2018.

[ZGX+12] Z. Zhang, X. Gu, Y. Xie, Z. Wang, Z. Wang, and K. Chakrabarty. Diagnostic system based on support-vector machines for board-level functional diagnosis. In *Proc. ETS*, pages 1–6, May 2012.

AUTHOR'S STATEMENT

The thesis is based on the following publications of the author.

- **Journal publications**

(J3) M. Pradhan and B. B. Bhattacharya, A Survey of Digital Circuit Testing in the Light of Machine Learning, WIREs Data Mining and Knowledge Discovery, pp. 1-18, 2020 (DOI: 10.1002/widm.1360).

(J2) 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. CAD*, vol. 38, no. 12, pp. 2343-2356, Dec. 2019.

(J1) M. Pradhan and B. B. Bhattacharya, COMEDI: Combinatorial Election of Diagnostic Vectors From Detection Test Sets for Logic Circuits, *IEEE Tran. VLSI*, vol. 25, no. 4, pp. 1467-1476, April 2017.

- **Conference publications**

(C1) M. Pradhan and B. B. Bhattacharya, A Prufer-Sequence Based Representation of Large Graphs for Structural Encoding of Logic Networks, in Proc. *ACM CoDS-COMAD*, 293-296, 2019.