Unsupervised Learning Provides Intelligence for Testing Hard to Detect Faults

Soham Roy
Intel Corp., Santa Clara, CA 95054
soham.roy@intel.com

Vishwani D. Agrawal
Auburn Univ., Auburn, AL 36849
agrawvd@auburn.edu

***Abstract*—Detecting hard-to-detect faults has always been a matter of concern and research in large and complex designs. This is due to the exponential complexity of automatic test pattern generation (ATPG) and the challenge of detecting faults in near-linear complexity CPU time. Achieving this goal would lead to near-zero backtracks. Many researchers have developed new algorithms to achieve this goal, but detecting faults through forward and backward circuit tracing has always involved human intuition or hunches, i.e., heuristics. Testability measures are employed to improve fault detection in reduced CPU time and with higher fault coverage. However, this improvement has been a never-ending process. A few decades ago, researchers claimed that no single testability measure could be effective as a heuristic, which led to the idea of amalgamating multiple testability measures to achieve the maximum efficacy of heuristics in ATPG. This opened up new venues, such as considering reconvergent fanouts (with odd/even inversions), transitive fan-in, and transitive fan-out, which have always been suspected of causing test complexities but never incorporated in the ATPG. The present work uses unsupervised learning, i.e., principal component analysis (PCA) for amalgamating existing testability measures and circuit topologies along with the aforementioned new features to detect hard-to-detect faults and third-party tool-reported aborted faults for large complex designs in reduced CPU time. We report a backtrack reduction to zero for some circuits or a substantial reduction for most circuits, with zero or fewer aborted faults, increased fault coverage, and lower ATPG CPU time. In a more focused study of 100 hard-to-detect or ATPG-aborted faults obtained from a third-party ATPG tool, we observed reduced backtracks, lower ATPG CPU time, fewer aborted faults, and increased fault coverage.**

***Index Terms*—ATPG, Aborted faults, Backtrace, Digital testing, Hard-to-detect faults, Machine intelligence (MI), Machine learning (ML), PODEM, Principal component analysis (PCA), Reconvergence fanout, Testability measures, Unsupervised learning.**

I. Introduction

Digital automatic test pattern generation (ATPG) has been around for over half a century. Many algorithms and CAD systems exists. So, what is new? Well, the ATPG is proven to be NP-complete [1], implying that the worst-case complexity of finding a test for a fault could be exponential in time in terms of the circuit size. The circuit size has seen tremendous growth in these years and shows no sign of slowing down. In a practical scenario, that will disturb the cost versus quality trade-off.

For a complex problem like ATPG, first we implement a complete algorithm like D-algorithm, PODEM, or fan [2]–[4]. Then, for practical reasons do two things, (1) set a per-fault time limit, which would finish the job in reasonable time though faults may remain undetected, and (2) use intuitive heuristics to prioritize choices in the test algorithm to detect more faults within the specified time limit.

Although a simple measure like logic distance to primary inputs and outputs can provide simple heuristics, many testability measures [5]–[12] have been used in ATPG programs. To be useful, they must be computationally efficient, ideally requiring only linear-complexity. Research related to various aspects of testability measures, such as their interpretations [13], applications [14], limitations, and improvements [15], has been reported. Notably, due to specific approximations in each measure, it is found that the ATPG benefits from different testability measures for different faults [16], [17].

The above approach does get the test but the repeated ATPG runs with different testability measures are time consuming. The situation improves if multiple measures can be combined. That is where the unsupervised learning technique of principal component analysis (PCA) [18], [19] enters into the picture. When three measures, distance (from original PODEM [3] heuristic), COP [8], and SCOAP [20], were combined by PCA and used for guiding the backtracks and D-drives in a PODEM program, results were amazing [21]. We quote a typical result: test for a hard-to-test fault in b07 circuit required 122, 720, 105 and 56 backtracks respectively with distance, COP, SCOAP, and the PCA-combination. For several other faults, the PCA-combination reduced backtracks to zero. Similarly, identification of redundant faults used fewer backtracks.

The above observations point to the possibility of further improvements in ATPG efficiency. If we continue to add new linear-complexity measures, each carrying new pieces of information about the circuit, then continuing reduction of backtracks should be possible. Reconverging fanout paths, especially those reconverging with odd and even number of inversions, are being considered. The novelty of this paper lies in adding more features (provides the algorithm for even/odd parity of the number of inversions encountered in the reconvergence path, transitive-fan-in toward number of primary inputs (PI) and transitive-fan-out toward number of primary outputs (PO) that may account in better ATPG performance, once they are included in the PCA measure for ATPG guidance.

II. Prior Work

Testability analysis typically refers to procedures that identify circuit test bottlenecks [13]. These procedures have a linear or polynomial, but not exponential, complexity. They determine numerical measures that represent the controllability and observability of signals. The distance or logic depth through the circuit is the most straightforward measure used in an ATPG algorithm [3]. In this case, the distance of a signal site in terms of logic gates between PI and the site is considered the controllability measure, and PO is the observability measure. Other testability measures include TMEAS [6], Sandia Controllability/Observability Analysis Program (SCOAP) [7], CAMELOT [12], and Controllability and Observability Program (COP) [8]. The first four examine the circuit topology, while the latter examines signal probabilities. These measures have been used to improve digital circuit design or select one out of multiple choices within complex test generation programs.

Researchers have conducted both theoretical and experimental studies on testability measures. Two commonly used algorithms for measuring testability are SCOAP [7] and COP. SCOAP measures the effort required to set a line to logic 0 or 1, while COP provides a single-pass probabilistic measure. However, both algorithms have significant inaccuracies due to their assumption that signals at reconvergent fanout stems are independent, which makes them unreliable in predicting fault detectability.

There are different methods for computing the accuracy of signal probability. One of them is using an algebraic procedure for line controllability as proposed by a higher accuracy signal probability computing algorithm [22]. Another procedure called PREDICT [10], [11] suggests graph-partitioning of the circuit into ”supergates” that include reconvergent fanouts. However, this method comes with a higher computational cost.

The cutting algorithm [23] is another approach that involves cutting selected fanout lines to make the circuit a tree structure. It then initializes the cut lines to a controllability range of [0,1]. The resulting modified network has no reconvergent fanout, making it easier to compute controllability bounds for all lines.

Fault detection probability can be computed by manifesting it as 1-controllability of a signal. This means taking the XOR of the good and faulty circuit outputs. The methods for computing probabilistic controllability can also be used for computing fault detection probabilities. However, this method has the disadvantage of doubling the circuit size for which controllability must be computed.

Some signals may need to have narrow enough bounds to be helpful. To address this issue, COP [8] introduced a probability-based testability measure that maintains computational efficiency by neglecting signal correlations. However, an analysis of the error in detection probability calculation [24] concluded that probabilities of control and observation of a line cannot be multiplied since those are not independent events. Another statistical fault analysis procedure, STAFAN [9], defined 0-observability and 1-observability of a line l as probabilities of the line being observed with value 0 or 1, respectively. The observabilities are conditional and can be multiplied by appropriate controllability to obtain fault detection probabilities without error. Other authors [25], [26] used conditional observabilities to obtain exact detection probabilities. It is also worth mentioning a fast testability analysis program [15] and a high-level testability measure [27]. Additionally, machine intelligence has begun to be applied in this area [28]–[30].signals these bounds may not be narrow enough to be helpful. To overcome this disadvantage, COP [8] provided a probability-based testability measure that maintains computational efficiency by neglecting signal correlations. A later analysis of the error in detection probability calculation [24] concluded that probabilities of control and observation of a line cannot be multiplied since those are not independent events. The statistical fault analysis procedure, STAFAN [9], defined 0-observability and 1-observability of a line l as probabilities of the line being observed with value 0 or 1, respectively. The observabilities are conditional and, therefore, can be multiplied by appropriate controllability without error to obtain fault detection probabilities. Other authors [25], [26] used conditional observabilities to obtain exact detection probabilities. Also worth mentioning are a fast testability analysis program [15] and a high-level testability measure [27]. Machine intelligence applications in this area have also begun [28], [31].

Any testability measure can provide heuristics for ATPG and test point insertion (TPI) algorithms. Various noteworthy theories [1] show that the ATPG and TPI for combinational circuits belong to the class of NP-complete problems, which means having greater than polynomial computation time complexity. *Heuristic* search techniques are used in ATPG for efficiency and in TPI for superior fault coverage in decent ATPG CPU time. Single testability measures can assist in providing heuristics for these algorithms. In ATPG, heuristic search techniques are used for efficiency, while in TPI, they are used to achieve superior test points (TPs) that result in higher fault coverage and reduced TPI time. Recent research has presented an ANN-based signal probability predictor for VLSI circuits considering reconvergent fanouts. Test points (TPs) indicated by higher fault coverage and reduced TPI time [32]. Recent work [31] presented an ANN-based signal probability predictor for VLSI circuits considering reconvergent fanouts. Some testability measures have been combined into a composite measure using unsupervised learning [21]. Four testability measures have been defined for every signal node: 0-controllability, 1-controllability, 0-observability, and 1-observability [33]. The last two measures are often replaced by a single measure, observability, leading to three measures per node.

The PCA combined testability measure has successfully improved the performance of ATPG. However, its potential has not yet been fully explored. In light of this, a new study has been initiated to delve deeper into the amalgamation process and explore how it can be further enhanced by adding more circuit topological features. Additionally, the study investigates how the amalgamated testability measure can detect hard-to-detect faults and aborted faults that are not typically reported due to excessive ATPG CPU time.

It is conjectured that the PCA-based ATPG method can reduce the number of aborted faults for larger circuits and increase confidence in fault coverage because it takes less time than conventional ATPG methods. The study will attempt to demonstrate this using the PCA-based method to detect as many aborted faults as possible.

Furthermore, the study will also incorporate the reconvergent fanout feature into the amalgamation process. This feature has not been considered by conventional testability measures or the PCA-based ATPG method developed by the authors of [21]. The aim is to prove the superior efficacy of the new approach compared to the conventional testability measure-based ATPG and the PCA-based ATPG method developed by the authors of [21].

III. Present Work

Machine Intelligence (MI) is a process that involves two key phases: learning from problem-specific data and utilizing that knowledge to solve problems. These phases may be called Artificial Neural Network (ANN) training and ANN guidance in supervised learning. However, in unsupervised learning, statistical tools such as Principal Component Analysis (PCA) [18], [34] is used instead.

The MI tool analyzes the problem-specific data during the first phase to extract relevant characteristics. These characteristics are then leveraged in the second phase to solve problems directly. In other words, the knowledge gained from the data is used to devise solutions to the problems.

Using PCA as the statistical tool, we have applied unsupervised learning to the Automatic Test Pattern Generation (ATPG) problem. This technique extracts the most relevant features from the data and uses them to generate test patterns automatically. By utilizing this approach, we can efficiently generate test patterns and achieve better results than traditional methods.

A. Dimensionality Reduction

Data is a valuable resource in the digital era, but its storage and computation pose challenges. PCA is a statistical technique that can reduce data dimensionality by generating new principal components. PCs hold the same information as the original data and can be used to identify patterns and trends. SVD is a technique used to obtain PCs based on ”explained variance.”

The explained variance of a subset of principal components (PCs) depends on the number of PCs and the individual variance of each PC. Adding up the individual variances of the PCs in a subset gives the total variance, which can be expressed as a percentage of the total explained variance. Often, only the first few PCs are necessary, but in some situations, the last few PCs may be of interest, such as in outlier detection or image analysis.

1) Putting Together Testability Measures: In this subsection, we will provide a comprehensive guide on how to create a composite measure using PCA [21] by combining the testability measures listed in Section II. We strongly advise that readers follow the amalgamation process illustrated in [21] before adding additional features such as PI and PO width, node reconvergence, and even/odd parity of reconvergent fanouts.

For each signal node in a circuit, ten measures and circuit topologies are defined: distance [3], COP-controllability-1 (CC1) [8], COP-observability (CO) [8], SCOAP-controllability-0 (SC0) [7], SCOAP-controllability-1(SC1) [7], SCOAP-observability(SCO) [7], transitive-fan-in affecting PIs (nPIs), transitive-fan-out affecting POs (nPOs), node reconvergence (ReConvNode), and even/odd parity of reconvergent fanouts (parity).

The following steps must be followed precisely to compute relevant values for each signal node (N) in the circuit and combine the features of a signal in the circuit. All calculations should be a single-pass, linear time complexity process:

- Calculate the minimum distance of the node ”N,” i.e. *distance*.
- Calculate CC1 for the ”N” that determines the probability of setting ”N” as 1. Since it is a probability, the complement of this value can determine the probability of setting ”N” as 0. Also, calculate CO to determine the observability of ”N” in the circuit.
- Calculate SC0 and SC1, the effort of setting the node ”N” as 0 and 1, respectively. Also, calculate SCO to determine the effort required to observe the node ”N” on the PO of the circuit.
- Evaluate the cone’s fan-in and fanout cones and bases to determine the nPIs and nPOs, respectively. They are further normalized by dividing them by their respective number of PIs and number of POs.
- Determine if the node ”N” is a convergent node by evaluating all its fanouts to explore whether they reconvene (ReConvNode). If they reconvene, the value is set to 1; otherwise, it is kept to 0.
- Calculate the inversions of two fanout paths in combination, whether they are even or odd, and select the minimum offset bound reconvergence point. The minimum offset is taken since the impact of backtracking in ATPG is heavily on the minimum offset reconvergence path, and this conflict further ripples toward PIs and POs, making the backtracing and forward tracing more time-expensive due to sky-rocketing undoing of decisions while choosing backtrace path and D-Frontier in the circuit. If the parity of inversion is odd, (parity) is set to 1; otherwise, it is kept to 0. The details of this algorithm are discussed in Section III-B.

Once the relevant values have been computed, they must be normalized from 0 to 1, and a phase correction must be applied as illustrated in [21]. This normalization enables the values to be compared and combined meaningfully, avoiding any bias or disproportionate influence of any particular value.

Finally, all measures must be combined using PCA, a statistical technique that identifies patterns and correlations in the data. If n measures are combined, then PCA computes n values for each node in the circuit.

In this study, we have two amalgamation tables: a backtracing table (distance, CC1, SC0, SC1, nPIs, ReconvNode, parity) and a forward tracing table (distance, CO, SCO, nPOs, ReconvNode, parity). The largest of these values is the principal component, used as the combined measure for setting the signal values to 0 and 1, respectively.

B. Algorithm for calculating even/odd parity of reconverging fanout

Firstly, we will delve into reconverging fanouts in a circuit node. These are the points where multiple signals converge after branching out from a single source. Next, we will focus on a node in the circuit with several reconverging fanouts.

For each combination of two signals, A and B, we will:

- Identify multiple reconverging points (represented by N = n1, n2, n3,...N) where the two signals converge.
- Select a reconverging point closest to the node (represented by d = minimum distance between the node and N).
- Calculate the parities (even/odd) for both reconverging fanouts while traversing from the node to N.
- Evaluate the XOR of the two parity for the reconverging fanouts (represented by X).
- Store the results in a look-up table that comprises d, X.

Finally, we will sort the look-up table in ascending order and choose the parity for the minimum value of ”d.”

In the study, the author deliberated using the minimum distance to calculate the even/odd parity of the reconvergence fanout. This choice was made to account for the impact of odd inversion that a signal may encounter within the reconverging fanouts.

Odd inversion occurs when a signal is inverted an odd number of times as it passes through a circuit. This can cause the signal to change polarity, significantly impacting the system's performance. If the reconverging point is closest to the reconverging fanout stem, this odd inversion has a more catastrophic effect on the signal than an even inversion.

By using the minimum distance, the number of backtracks is expected to reduce, improving the performance of test generation about CPU time. Backtracks occur when a signal has to retrace its steps to avoid conflicts within the circuit. This can slow down the overall system performance; thus, reducing the number of backtracks is highly desirable. Overall, the choice of minimum distance helps optimize the system's performance by minimizing the impact of odd inversion and reducing the number of backtracks.

IV. Experiments on Test Generation: Results and Discussion

The present study endeavored to demonstrate the potential of machine intelligence (MI) in enhancing the performance of electronic design automation (EDA) tools. However, the absence of access to the internal details of commercial software necessitated the use of in-house EDA tools to compare algorithmic improvements. Experiments were performed on an Intel-i7-10610U processor and 16 GB RAM. The EDA software was implemented in C++ using MSVC++14.15 compiler with due emphasis on optimizing performance. The PCA algorithm was executed using the Python programming language, whereas PODEM ATPG [3] was implemented with an event-driven simulator [37] using the C++ programming language. In this regard, the PODEM algorithm was programmed to enable the application of any testability measure, including but not limited to distance [3], COP [8], SCOAP [7], or PCA, to various benchmark circuits [35], [36]. As the ATPG process is computationally expensive, some faults may be aborted. Nonetheless, it was observed that a comparable fault coverage could be accomplished with each testability measure by suitably configuring the per-fault time limit. The proposed ATPG system employed in the present study can identify all checkpoint single stuck-at faults. Furthermore, the system initiates a random pattern detection (RPD) phase to eliminate faults that can be detected using random patterns. The remaining faults are then tested using PODEM ATPG, guided successively by a single testability measure, such as distance, COP, SCOAP, or by a combined measure of PCA. Also, a suitable per-fault time limit produces fault coverage similar to all heuristics. The effectiveness of the PCA algorithm was tested in three sets of experiments, which have been elaborated in subsequent subsections.

A. Test Generation with zero aborted faults

The experiment aimed to evaluate the performance of the PODEM ATPG method using various testability measures such as “Distance”, “COP”, “SCOAP”, “PCA [21]”, and “new PCA”. The ATPG was performed on nine circuits from the ISCAS'85 [36] and ITC'99 [35] benchmarks (b06, b03, b09, b08, b10, b13, c880, b11, b12 based on the ascending order of their number of nodes as illustrated in the figures) to test checkpoint stuck-at faults, keeping the similar fault coverage since the per-fault time limit was kept the same across all the testability measures under consideration in this study..

The study's results regarding ATPG CPU time and the number of backtracks were analyzed, as shown in Figures 1 and 2. It was observed that circuit b06 had no backtracks for all testability measures. Moreover, the new PCA-based ATPG outperformed other testability-measure-based methods regarding quality. This was evidenced by the new PCA-based ATPG method reduced the backtracks to zero compared to other testability measure-based methods and substantially reduced the ATPG CPU time.

The study also found that the new PCA-based ATPG of this study reduced the ATPG CPU time across circuits, namely b09, b08, b10, b13, c880, b11, and b12, with a substantial reduction in backtracks except for b09, b10, b11, b12, and b13 where the reduction in backtracks was not substantial but still lower compared to other testability measure-based ATPG methods.

The study demonstrated that the PCA-based ATPG method is an effective and efficient way to perform ATPG and improve the quality of backtracing. The results are precise: this method significantly reduces the time taken to perform ATPG for zero backtracks and improves the quality of backtracing. This is a

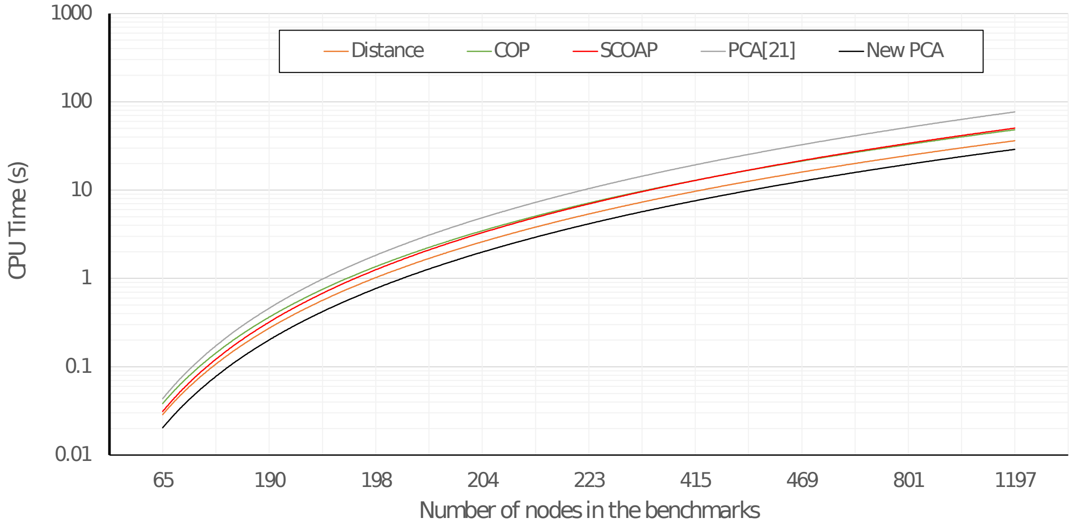

Fig. 1. Total ATPG CPU time for finding a test or proving redundancy for all checkpoint faults left undetected by the random ATPG phase applied to ITC'99 [35] and ISCAS'85 [36] benchmark circuits.

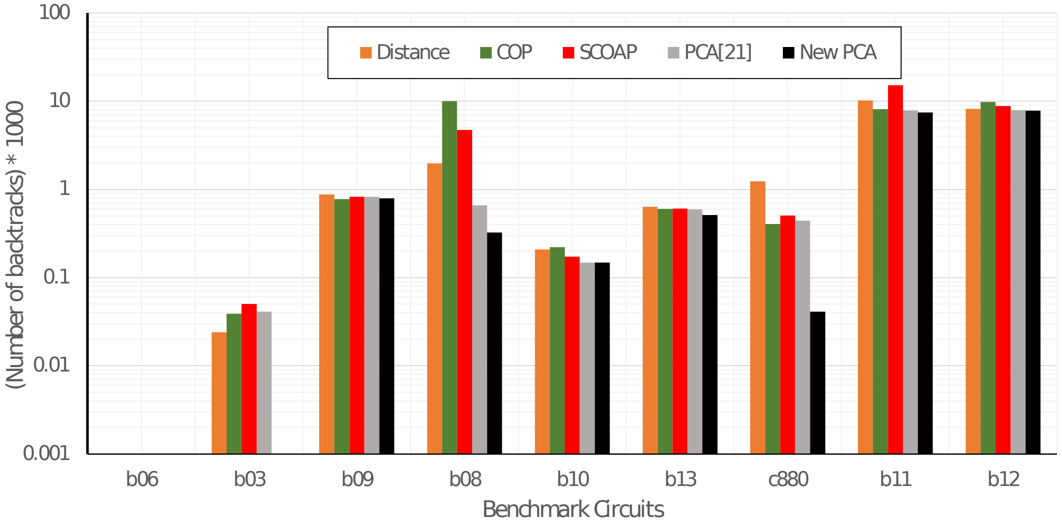

Fig. 2. Total backtracks used while finding a test or proving redundancy for all checkpoint faults left after the random ATPG phase applied to ITC'99 [35] and ISCAS'85 [36] benchmark circuits.

significant breakthrough in testing methodology, as it ensures that all faults can be detected without any aborted faults or backtracks for specific circuits.

It is important to note that the improvement in backtracing quality is a significant achievement in itself. Other testability-measure-based ATPG methods can lead to backtracks for faults, even in circuits with no redundant faults. However, this study's PCA-based ATPG method ensures that all faults can be detected with no aborted and redundant faults and zero backtracks for specific circuits, making it an essential method for any testing methodology.

B. *Test Generation with aborted faults*

The third-party tool usually does not target aborted faults by default. However, it retargets them and uses dynamic learning during retargeting. Note that using this setting can sometimes result in significantly higher run times. This experiment aimed to evaluate the performance of the PODEM ATPG method using various testability measures such as "Distance", "COP", "SCOAP", "PCA [21]", and "new PCA" to observe the impact

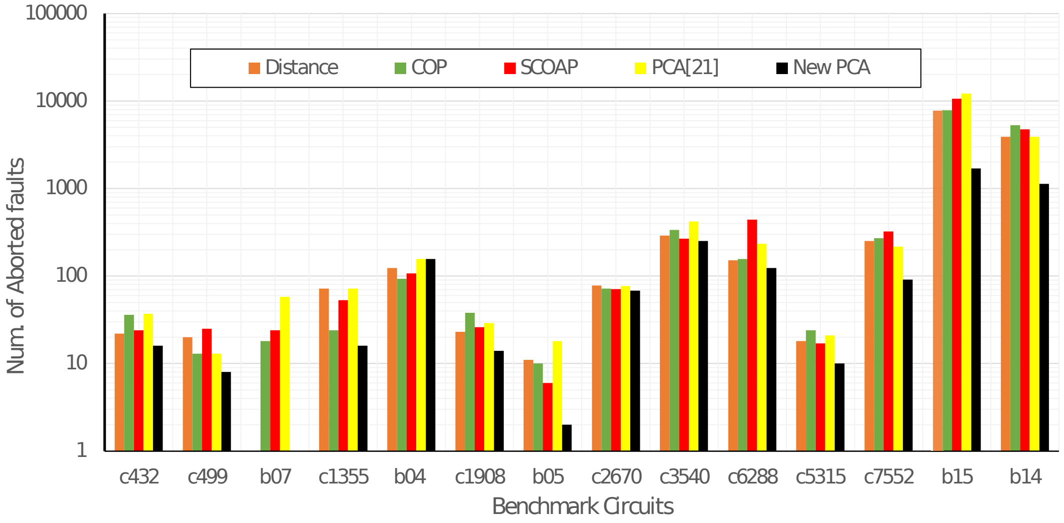

Fig. 3. Total aborted faults after the random and deterministic ATPG phase applied to ISCAS'85 [36] and ITC'99 [35] benchmark circuits.

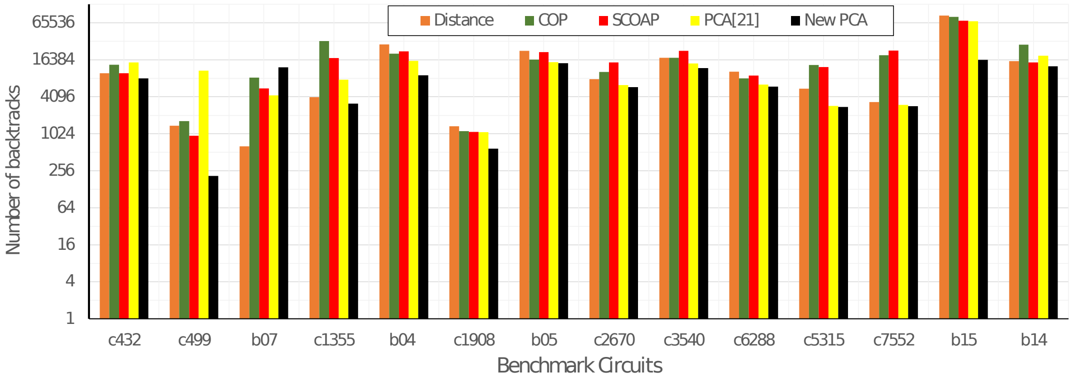

Fig. 4. Total backtracks used while finding a test or proving redundancy for all checkpoint faults left after the random ATPG phase applied to ISCAS'85 [36] and ITC'99 [35] benchmark circuits.

on the detection of aborted faults, ATPG CPU time, and number of backtracks, and most importantly impact on fault coverages as illustrated in Figures 3, 4, 5 and 6, respectively. The ATPG was performed on fourteen circuits (arranged from left to right based on the number of nodes) from the ISCAS'85 [36] and ITC'99 [35] benchmarks (c432, c499, b07, c1355, b04, c1908, b05, c2670, c3540, c6288, c5315, c7552, b15, b14 based on the ascending order of their number of nodes as illustrated in the figures) to test checkpoint stuck-at faults.

We observed that there had been a constant reduction in ATPG CPU time and a substantial reduction in the number of backtracks and aborted faults, as well as a substantial increase in faut coverages, with few exceptions. More prominence can be found in circuits comprising more nodes illustrated to the extreme right. In a circuit like b07, the new PCA can reduce the number of aborted faults to zero but at the cost of many backtracks and ATPG CPU with a lesser impact on the tool efficiency and within the decent ballpark. Also, in a circuit like b04, fault coverage can be seen as unimproved; however, it has reduced backtracks and CPU time (not visually promising reduction).

C. Test Generation with 100 hard-to-detect faults on few larger benchmarks

This experiment aimed to test the effectiveness of a new PCA method in detecting difficult-to-detect faults in more complex and extensive circuits. We ran a third-party ATPG tool on the circuits namely, b20, b21, and b22 and reported the list of aborted faults and extracted 100 hard-to-detect faults based on COP values using our internal EDA tool and

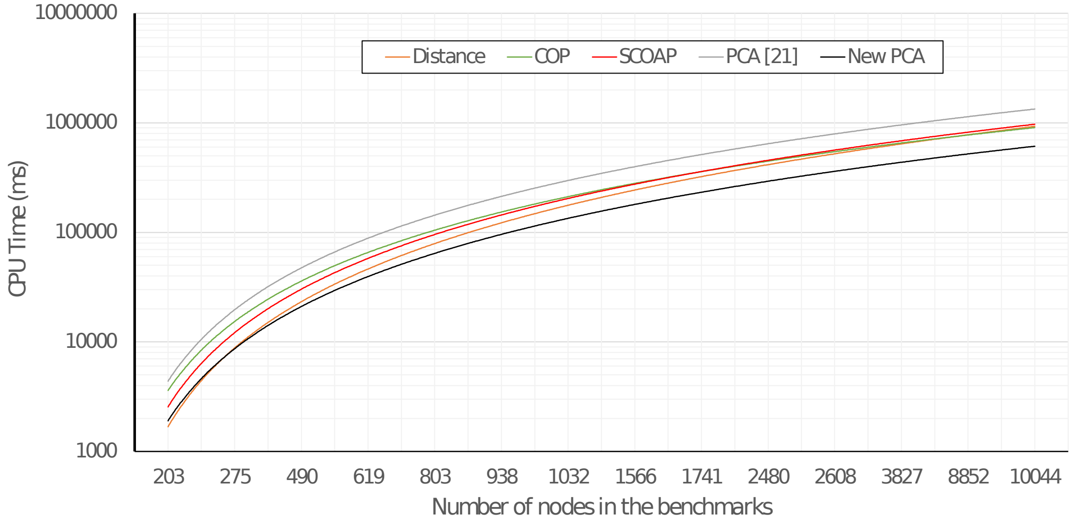

Fig. 5. Total ATPG CPU time for finding a test or proving redundancy for all checkpoint faults left after the random ATPG phase applied to ISCAS'85 [36] and ITC'99 [35] benchmark circuits.

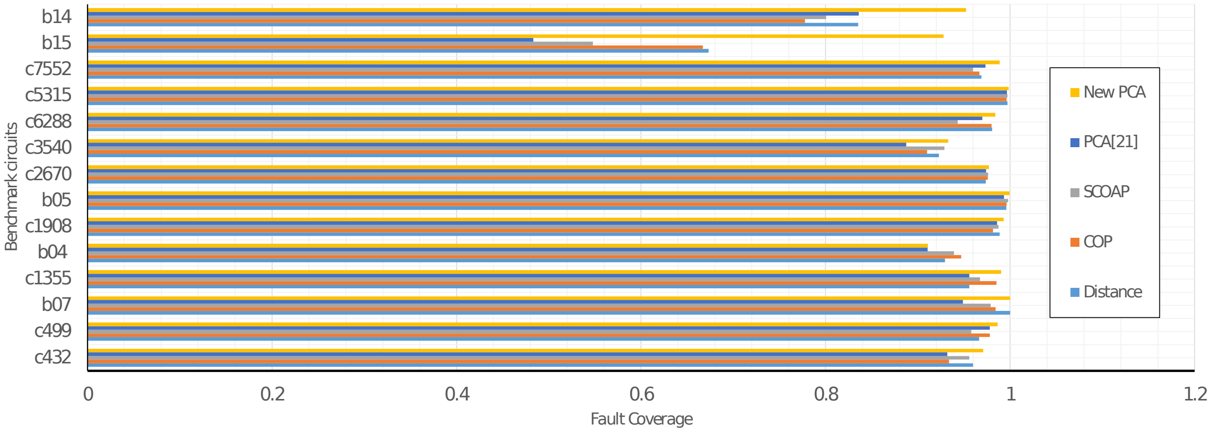

Fig. 6. Total fault coverages while finding a test or proving redundancy for all checkpoint faults of ISCAS'85 [36] and ITC'99 [35] benchmark circuits.

performed PODEM ATPG method using various testability measures such as "Distance", "COP", "SCOAP", "PCA [21]", and "new PCA" to observe the impact on the detection of aborted faults, ATPG CPU time, and number of backtracks, and most importantly impact on fault coverages as illustrated in Figures 7, 8, 9, and 10, respectively. The study found that reduced backtracks, ATPG CPU time, and aborted faults increased circuit fault coverage. This indicates that the method discussed in this study can detect much harder faults of larger circuits that third-party tools cannot detect within the decent bound constrained abort limit set by default.

V. Conclusion and Future Directions

This study aimed to investigate whether a PCA-based combination of testability measures and circuit topological features can enhance the efficiency of ATPG methods. The results of this study are clear - this approach can significantly improve the number of backtracks, reduce the ATPG CPU time, and enhance the detection of hard-to-detect faults. This method increases the fault coverages required for the ISCAS'85 and ITC'99 benchmarks and provides more robust detection of aborted faults than traditional heuristic-based ATPG. The researchers are confident that their new method can be applied to other fault models, detect user-defined fault models and transition/stuck-at/cell-aware fault models more efficiently, and provide greater fault coverage with less ATPG CPU time.

This research can be the driving force behind developing silent data error detection and can be an incremental upgrade to the current third-party ATPG tool. Although EDA vendors have restrictions on divulging information, the authors of this study are confident that their method can be an excellent addition to the current third-party ATPG tool while improving its quality.

The study has demonstrated that the PCA-based ATPG method is a reliable and efficient way to improve the quality of backtracing followed by a reduction in ATPG CPU time, detecting more aborted faults, and providing a good number of fault coverage. These results are a significant step forward in testing various NP-hard methodologies that can help create more robust and efficient testing methods for electronic systems. The researchers are confident that this research can be the foundation for future testing methodologies and provide a more reliable and efficient way to ensure the quality of various electronic systems.

References

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

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

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

[4] H. Fujiwara and T. Shimono, “On the Acceleration of Test Generation Algorithms,” *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1137–1144, Dec. 1983.

[5] J. Stephenson, “A Testability Measure for Register-Transfer Level Digital Circuits,” Ph.D. dissertation, Carneigie-Mellon Univ., 1974.

[6] J. Grason, “TMEAS, A Testability Measurement Program,” in *16th Design Automation Conference*, 1979, pp. 156–161.

[7] L. Goldstein, “Controllability/Observability Analysis of Digital Circuits,” *IEEE Transactions on Circuits and Systems*, vol. CAS-26, no. 9, pp. 685–693, Sep. 1979.

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

[9] S. K. Jain and V. D. Agrawal, “Statistical Fault Analysis,” *IEEE Design & Test of Computers*, vol. 2, no. 1, pp. 38–44, 1985.

[10] S. C. Seth, B. B. Bhattacharya, and V. D. Agrawal, “An Exact Analysis for Efficient Computation of Random-Pattern Testability in Combinational Circuits,” in *Proc. Fault Tolerant Computing Symposium (FTCS)*, Vienna, Austria, Jul. 1986, pp. 318–323.

[11] S. C. Seth and V. D. Agrawal, “A New Model for Computation of Probabilistic Testability in Combinational Circuits,” *Integration*, vol. 7, no. 1, pp. 49–75, 1989.

[12] R. G. Bennetts, C. M. Maunder, and G. D. Robinson, “CAMELOT: A Computer-Aided Measure for Logic Testability,” *IEE Proceedings E - Computers and Digital Techniques*, vol. 128, no. 5, pp. 177–189, 1981.

[13] V. D. Agrawal and M. R. Mercer, “Testability Measures – What Do They Tell Us?” in *Proc. International Test Conf.*, Philadelphia, PA, Nov. 1982, pp. 391–396.

[14] D. M. Singer, “Testability Analysis of MOS VLSI Circuits,” in *Proc. Int. Test Conf.*, 1984, pp. 690–696.

[15] I. Ratiu, “VICTOR: A Fast VLSI Testability Analysis Program,” in *Proc. of the International Test Conf.*, 1982, pp. 397–401.

[16] J. Patel and S. Patel, “What Heuristics are Best for PODEM?” in *Proc. First International Workshop on VLSI Design*, 1985, pp. 1–20.

[17] S. Patel and J. Patel, “Effectiveness of Heuristics Measures for Automatic Test Pattern Generation,” in *Proc. 23rd ACM/IEEE Design Automation Conference (DAC)*, 1986, pp. 547–552.

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

[19] I. Jolliffe, *Principal Component Analysis*. Springer-Verlag, 2002.

[20] L. H. Goldstein, “Controllability/Observability of Digital Circuits,” *IEEE Trans. on Circuits and Systems*, vol. CAS-26, no. 9, pp. 685–693, Sep. 1979.

[21] S. Roy and V. D. Agrawal, “An Amalgamated Testability Measure Derived from Machine Intelligence,” in *Proceedings of 37th International Conference on VLSI Design & 23rd International Conference on Embedded Systems*, Jan. 2024, pp. 696–701.

[22] K. Parker and E. McCluskey, “Probabilistic Treatment of General Combinational Networks,” *IEEE Transactions on Computers*, vol. C-24, no. 6, pp. 668–670, 1975.

[23] J. Savir, G. S. Ditlow, and P. H. Bardell, “Random Pattern Testability,” *IEEE Transactions on Computers*, vol. C-33, no. 1, pp. 79–90, Jan. 1984.

[24] J. Savir, “Good Controllability and Observability Do Not Guarantee Good Testability,” *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1198–1200, 1983.

[25] V. D. Agrawal and S. C. Seth, *Tutorial: Test Generation for VLSI Chips*. Computer Soc. Press, 1988.

[26] S. Seth, L. Pan, and V. D. Agrawal, “PREDICT - Probabilistic Estimation of Digital Circuits Testability,” in *Proc. of the International Fault-Tolerant Computing Symp.*, 1985, pp. 220–225.

[27] T. Lee, W. Wolf, N. Jha, and J. Acken, “Behavioral Synthesis for Easy Testability in Data Path Allocation,” in *Proceedings 1992 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, 1992, pp. 29–32.

[28] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2343–2356, Dec. 2019.

[29] S. Roy, “Toward Zero Backtracks in Test Pattern Search Algorithms with Machine Learning,” Ph.D. dissertation, Auburn University, USA, 2021.

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

[31] J. Immanuel and S. K. Millican, “Calculating Signal Controllability Using Neural Networks: Improvements to Testability Analysis and Test Point Insertion,” in *Proc. IEEE 29th North Atlantic Test Workshop (NATW)*, 2020, pp. 1–6.

[32] Y. Sun, “Novel Test Point Insertion Applications in LBIST,” Ph.D. dissertation, Auburn University, USA, 2021.

[33] S. K. Jain and V. D. Agrawal, “Statistical Fault Analysis,” *IEEE Design & Test of Computers*, vol. 2, pp. 38–44, Feb. 1985.

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

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

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

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

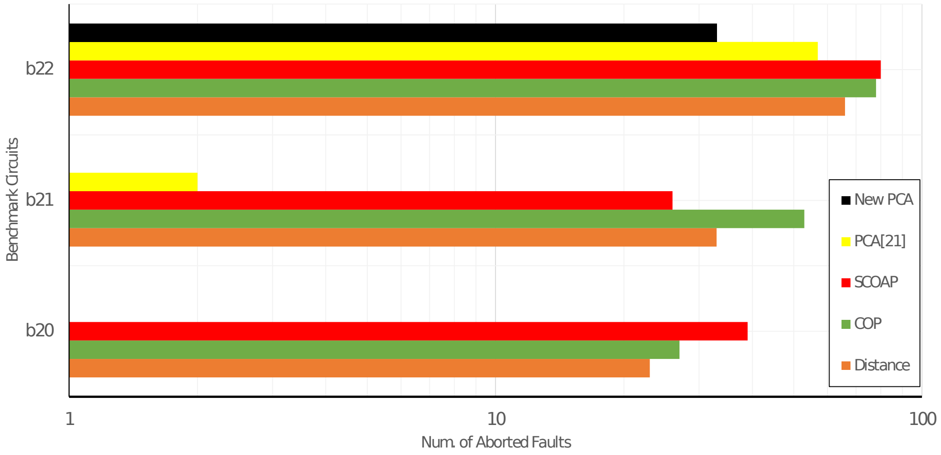

Fig. 7. Total aborted faults after the deterministic ATPG phase applied to 100 hard-to-detect faults of ITC'99 [35] benchmark circuits.

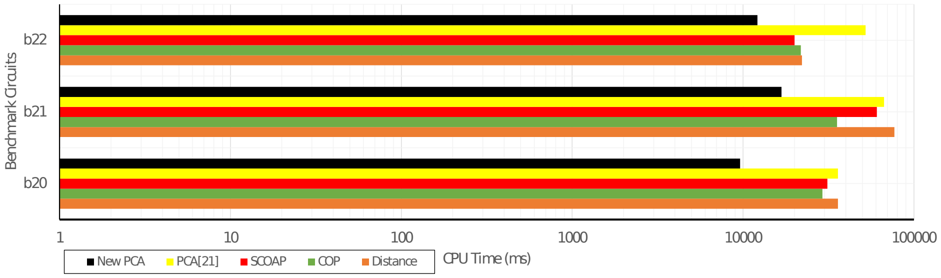

Fig. 8. Total ATPG CPU time consumed while finding a test or proving redundancy for 100 hard-to-detect faults of ITC'99 [35] benchmark circuits.

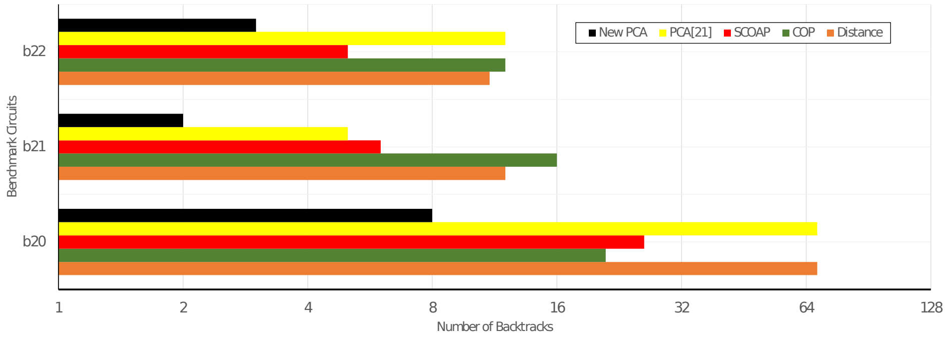

Fig. 9. Total backtracks used while finding a test or proving redundancy for 100 hard-to-detect faults of ITC'99 [35] benchmark circuits.

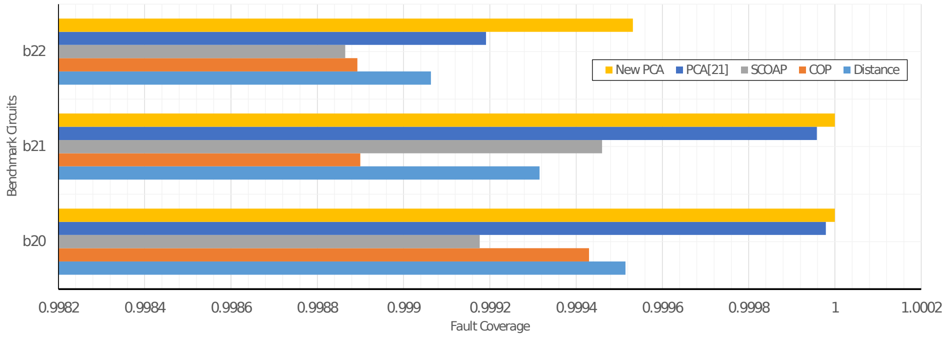

Fig. 10. Total fault coverages while finding a test or proving redundancy for 100 hard-to-detect faults of ITC'99 [35] benchmark circuits.