INFORMATION THEORETIC AND SPECTRAL METHODS OF TEST POINT, PARTIAL-SCAN AND FULL-SCAN FLIP-FLOP INSERTION TO IMPROVE INTEGRATED CIRCUIT TESTABILITY

BY RAGHUVEER AUSOORI

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Prof. Michael L. Bushnell

and approved by

New Brunswick, New Jersey

October, 2009

ABSTRACT OF THE THESIS

Information Theoretic and Spectral Methods of Test Point, Partial-Scan and Full-Scan Flip-Flop Insertion to Improve Integrated Circuit Testability

by Raghuveer Ausoori

Thesis Director: Prof. Michael L. Bushnell

We present a radically new *design for testability* (DFT) algorithm, which inserts *test points* (TPs) and *scanned flip-flops* (SFFs) into large circuits to make them testable. The algorithm measures testability using Shannon's entropy measure (from information theory), which will be shown to be a vastly superior way to measure testability, and spectral co-efficients. The spectral measures are superior in measuring *fault coverage* (FC) improvement. The algorithm can determine the DFT candidates using a gradient descent method or using an *integer linear program* (ILP). The optimal insertion of the TPs and SFFs reduces the amount of DFT hardware, since the algorithm trades off inserting a TP versus inserting a SFF. Various other derived measures are used and found to be effective in making the circuit testability better at various stages. The integer linear program finds the optimal solution to the optimization, and the testability measures are used to maximize information flow through the *circuit-under-test* (CUT). The result, on full-scan designs with test points, is a 40.05% reduction in *test volume* (TV) and a 54.24% reduction in *test application time* (TAT), compared to a full-scan design without test points. The method, used in conjunction with the Synopsys TetraMAX^{TM} *automatic test pattern generator* (ATPG), achieves 1.55% higher *fault coverage* (FC) and 100% *fault efficiency* (FE) on ITC '99 benchmark circuits compared to conventional methods and reduces the ATPG time by 90.24%. The method works better than all prior methods on partial-scan circuits, as well. We achieve TV reductions of 19.56% and 33.42% and TAT reductions of 21.63% and 31.23%, over the previous best SPARTAN PS+TP1 and PS+TP2 partial scan ideas, respectively, on ISCAS '89 benchmark circuits. We also get 32.62% TV reduction and 25.39% TAT reduction over the *mpscan* algorithm.

Acknowledgements

I would like to express my gratitude to Prof. Bushnell for helping me and spending his valuable time with me to bring my thesis to fruition. I am thankful for all the attention he diverted toward my work especially during the latter part of the thesis. I would also like to convey my special thanks to Dr. Tapan Chakraborty and Dr. Xinghao Chen for their feedback and ideas to make this work as good as it is now.

I would like to acknowledge my parents and family for their faith in me, which is all I needed to keep pushing myself through the hardest of times. I also want to convey my gratitude to my roommates, Hari, Shyam, Shiva and Venkat, who were my family away from home, without whom I could not have survived a day after landing in the US.

My acknowledgments would not be complete without the mention of my mentors Dr. Rajamani Sethuram and Dr. Omar Khan from whom I have learned much about DFT and I would like to thank them for all their insight and timely advice. I also thank my other seniors Dr. Hari Vijay, Varadan, Roystein, Sharanya and Aditya for making my lab visits something to look forward to. I thank all of my friends for making my graduate and personal life a lot more fun.

Dedication

To my parents Banu and Krishna, my grandmother Janakibhai Kannan, my uncle and aunt Hanumantha Rao and Sasikala, my aunt Nalini Sathyakumar, my uncle Rajendran, my uncle Narendran, my roommates Hari, Shyam, Shiva and Venkat, and all my friends.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

The Equation below represents Moore's Law [45], which stated that the number of transistors on a chip will double roughly every year and a half. This is mathematically shown by Bell [5] as follows:

$$\begin{aligned} \text{Transistors per chip} &= 2^{t-1959} \text{ for } 1959 \leq t \leq 1975 \\ & \quad 2^{16} \times 2^{(t-1975)/1.5} \text{ for } t \geq 1975 \end{aligned}$$

The first half of the Equation shows that every year after 1959, the number of transistors per chip doubles from the previous year. The growth in the *integrated circuit* (IC) industry was so rapid that Moore's Law had to be modified as shown in the second half of the Equation, which shows that the number of transistors in a chip doubles every one and a half years. The transistor count on each chip has already broken the one billion barrier and continues to increase rapidly. It is now commonplace to have netlists with more than 50 million logic gates. By 2010, scalable CMOS microprocessors could combine into powerful, multiple processor clusters of up to one million independent computing streams [5]. Designing and fabricating these devices pose a tough challenge to the industry and this challenge has to be met with less cost than for the previous generation to survive in the market. Companies strive to be the first to release new chip sets and capture a greater share of the market. In order to achieve this, a good fabrication line and a good testing scheme are required.

1.1 Need for DFT Insertion

Hardware testing is the process of testing the chip to determine whether the chip has been fabricated properly. The cause of failure can be that the fabrication process was faulty or there could be a design or specification mistake or the test itself was wrong to begin with. The role of testing is to detect whether something went wrong and if anything did, then the defect can be diagnosed and rectified, and the process is repeated. A good test helps remove the bad chips in a short amount of time. While testing, the metric that is observed is *fault coverage* (FC), which is the percentage of faults detected by the tests. The higher the FC, the better the quality of the chip. Getting high FC on chips with one billion transistors is hard and to make the process easy, the chips are designed so that they are easily testable. This concept is called as *design-for-testability* (DFT). In DFT, the chip logic is altered so that the chip is easier to test with higher FC, while maintaining the original functionality. The following are some ideas used in DFT:

- **Full Scan:** Scanning is the process of converting the hard-to-test flip-flops present in the circuit into shift registers, and thus, making them easier to control from the pins. When all of the flip-flops present in the circuit are scanned, then the circuit acts as a combinational circuit and it is said to be *full-scanned*. Combinational circuits are easier to test with high FC, but have high hardware and delay overheads.

- **Partial Scan:** When only a subset of flip-flops present in the circuit are scanned, then the circuit is said to be *partially-scanned* [2]. The circuit still remains sequential but a good partial-scan algorithm should be able to achieve high FC [35, 36]. The hardware overhead is less than for a full-scanned circuit.

- **Test Point Insertion:** Test points are special designs that help introduce a certain logic value into a particular combinational part of the circuit or help observe the output of a gate. They help improve the testability of combinational parts of the circuits [62]. Another advantage of test points is that when placed properly, they can help reduce the amount of time taken to test the circuit [57].

Testing a chip costs money and it is estimated that testing alone comprises about 33% to 40% of the chip cost [10]. *Automatic test equipment* (ATE) is very costly and its operating cost increases with every second. ATE tests the chips by applying certain inputs to the pins of the chip and observing the outputs to detect faults. The inputs that are applied to the chips are called test vectors, which are determined before the fabrication of the chip. The test vector sets are processed to achieve high FC with fewer vectors. The total number of vector bits stored in the ATE is called the *test volume* (TV) and the time taken by the ATE to apply the test vectors to one chip and read its outputs is called the *test application time* (TAT). Higher TV and TAT add to the test cost of the chip. Testing cost with the latest equipement is estimated to be 7 cents per second. The more time each chip spends on the ATE, the higher the testing cost incurred. For example, let us assume that the test time for a digital *application specific IC* (ASIC) is around 10 seconds and that 100,000 chips are fabricated every day.

$$\text{Testing Cost} = 7 \times 10 \times 100,000 = 7,000,000 \text{ cents} = \$70,000 \text{ per day}$$

By using the DFT ideas discussed above intelligently, the TV and TAT can be reduced, which in turn, helps reduce the test cost. The ideas proposed in this thesis help reduce the TV by 40.05% and TAT by 54.24%, while acheiving a near 100% stuck-at FC. Now the testing cost calculation becomes:

$$\text{Testing Cost} = 7 \times 10 \times 100,000 \times (1 - 0.5424) = \$32,032 \text{ per day}$$

Using the ideas proposed here, the testing cost is cut in half. Hence, DFT insertion not only contributes to higher FC, but it can also help reduce the cost of the product.

1.2 Original Contributions

The previous section explained why DFT insertion is important from the quality and economic viewpoint of the IC industry and how the ideas proposed in this thesis help with better DFT. This section explains what makes the ideas unique and better than existing ideas.

1.2.1 Fully Automated Procedure

Existing automated methods insert partial-scan or full-scan DFT hardware. However, for high-performance designs, the DFT insertion process is largely manual, because various performance and low power design constraints make it difficult to fully automate this. This leads to a very labor intensive process of an expert test engineer inserting the DFT hardware by hand into a high-performance design, such as a microprocessor or cell phone chip. This thesis provides better design automation, because the algorithm for DFT hardware insertion allows performance critical nets in the circuit to be weighted, so that the algorithm will never put DFT hardware on those nets, but instead the hardware is driven off of the nets onto other sites. The algorithm, thus, reduces the manual process involved in DFT insertion and reduces the design time and time-to-market of the product.

1.2.2 Combined Scan Flip-Flop and Test Point Insertion

Existing partial scan with test point insertion algorithms first scan the flip-flops and then insert test points [35, 36, 37]. Following this two-step procedure could increase the hardware overhead and TV of the circuit. One original idea here is to simultaneously insert test points while scanning the flip-flops, so that the advantages of either can be weighed and a wiser choice can be made. In the former method, if the testability of a combinational block is found to be low, a nearby flip-flop will be scanned to improve the testability but if this does not help the cause, then the second stage of the algorithm will insert a test point, thus increasing overhead. The latter method would automatically insert a test point at the appropriate point in the combinational block. This concept can be observed through the results obtained where the idea performs better than the previous best idea with less overhead.

1.2.3 Entropy Analysis

Entropy is the amount of information flowing through a particular point of interest in the circuit, first proposed by Shannon [59]. If there is more information flowing through a gate, then there is more activity in the gate and, hence, it can be more easily controlled. Increasing the information flow throughout the circuit translates into better circuit testability. Entropy is a very simple testability measure that can be easily calculated by logic simulating the circuit with a certain number of random input vectors. The entropy analysis is explained in detail in the Chapter 4. Using entropy as a testability measure has been proposed before by Dussault [23]. He proposed a way to calculate the entropy and conditional entropy of the gates in the circuit and defined observability, controllability and testability measures from the entropy values. Agrawal [3] proposed a method to use entropy for digital fault testing. He calculates the detection probability of a fault and explains how to suitably design a pattern generator to improve the detection probability. Thearling and Abraham [64] proposed information theory based testability measures at the functional level. Probabilities were estimated either via logic simulation, called *sampling*, or by using functional level information of the circuit components, known as *composition*. They also proposed partitioning of the circuit, to improve testability, using entropy-related measures. We use the following ideas to obtain better performance.

1.2.3.1 Unbiasing

Entropy is calculated using logic simulation. This simulation does not exhaust the input space for the circuit as that can be very time consuming for larger circuits and, hence, a sample of the input space is used for logic simulation. This input vector sampling can introduce a statistical sample bias into the entropy values [28, 31]. We propose a novel way of calculating this bias and a method to remove the bias. Our bias removal improves the algorithm performance greatly when compared to using biased entropy as the testability measure.

1.2.3.2 Integer Linear Programming

Integer linear programming is an optimization algorithm that chooses the optimal solution for the problem. The proposed algorithm uses an integer linear program to choose candidates for scanning and test point insertion based on the testability measures. Another advantage with using an integer linear program is that in the case of many candidates having the same testability measure, the selection is made randomly, which works better than selecting based on a candidate characteristic such as the gate name.

1.2.3.3 Derivative Measures

Entropy as such can be an effective testability measure for candidate selection but the entropy measure only accounts for difficult-to-control gates. To account for difficult-to-observe gates, derivative measures were invented:

- *Entropy gain* selects candidates by the increase in entropy at the fan-out gates if the candidate is scanned or receives a combinational logic test point.
- *Entropy differential* selects candidates by the drop in entropy from the candidate to its fan-out gates.

After inserting a test point or scanning a flip-flop, the variables used for the entropy calculation are cleared for recalculation of the entropy in the next simulation. When these variables are not cleared, the information from previous simulations accumulates and the entropy calculated using this information is called *accumulated entropy.* The calculations of all of the measures are explained in detail in the Chapter 4.

1.2.4 Spectral Analysis

Spectral analysis of a circuit gives information about the types of frequencies that best excite the circuit. The output from each gate for 32 clock periods is multiplied with a 32×32 Hadamard matrix, to calculate the spectral co-efficients of the gate. The spectral co-efficients calculated from the result help identify the natural response frequencies for the gate. Using spectral co-efficients as testability measures has been done before [35, 36, 37] and in this thesis, we use the existing ideas of spectral analysis in combination with our entropy ideas.

1.2.4.1 High Speed Bit-Wise Computation

The original contribution to spectral analysis from the approach here is a high speed bit-wise computation to multiply the 32×32 Hadamard matrix with the output of each gate to calculate its spectral co-efficients. The matrix multiplication is replaced with a bit-wise XNORing operation. Each row of the Hadamard matrix is transformed into a 32-bit word and the 32-bit logic response from the gate is bit-wise XNORed with each row to obtain the spectral co-efficients. This method speeds up the spectral analysis many fold.

1.3 Results

The complexity of the proposed gradient descent algorithm is $O(N^2)$ with logic simulation taking the most computation time. The complexity of the algorithm while using integer linear program is $O(2^N)$, but the AMPL solver used manages to find a solution in a very short time and hence, this method can be used practically. The algorithm can be made to insert test points in a full-scan circuit or to insert test points and scan flip-flops in a partial-scan circuit. By inserting test points into a full-scan circuit, the algorithm was able to achieve a 40.47% TV and a 54.24% TAT reduction over a full-scan circuit with no test points. The fault coverage went up by 1.64% while keeping the hadware overhead below 10%, on average. By inserting both test points and scan flip-flops into a circuit with partial scan, the algorithm acheives 19.56% TV reduction and 21.63% TAT reduction over the previous best algorithm.

1.4 Outline of the Thesis

This thesis is organized as follows. Chapter 2 discusses the prior work in the DFT field. Section 2.1 introduces various interesting ideas used for partial scan, Section 2.2 explains the algorithms that insert test points into sequential circuits with no scan flip-flops, Section 2.3 discusses test point insertion algorithm in full-scan and partial scan designs and Section 2.4 introduces entropy as a testability measure and discusses its advantages. Chapter 3 discusses the integer linear programming approach and the formulation used by the algorithm to select the test point candidates. Chapter 4 explains the algorithm in detail including the various testability measures used and provides a flow chart of the entire procedure. Chapter 5 discusses in detail the various results obtained for various runs of the algorithm. Chapter 6 explains the implementation of the algorithm. Chapter 7 presents the conclusion and future work.

Chapter 2
Prior Work

The *design-for-testability* (DFT) techniques are required to improve the testability and reduce the test cost of the circuit. DFT techniques include ad-hoc and structured techniques. *Test point insertion* (TPI) is a common ad-hoc DFT technique to improve the circuit testability and scan design is the most widely used structured DFT methodology. Full-scan, partial-scan and *random-access scan* (RAS) design are the three most common scan architectures. Full-scan techniques scan all the flip-flops in the circuit and make the circuit combinational so that it is easier to test. Partial scan techniques scan a subset of the flip-flops in the circuit and, hence, incur smaller hardware overhead and test volume than full-scan techniques. RAS techniques group the flip-flops present in the circuit into *random access memory* (RAM) and write the test vectors into them by using a special address scan register and address decoding hardware. TPI techniques concentrate on increasing the testability of the circuit by inserting controllability and observability points.

2.1 Entropy as a Testability Measure

Entropy as proposed in Shannon's Information Theory [59] is an effective measure of randomness in data. It is widely used as a measure of information in signals to combat noise-related errors of communication.

2.1.1 Why Entropy is a Better Testability Measure

In this work, we use entropy as a testability measure to estimate the testability of circuits. The Entropy E of a signal that has n distinct values [44, 73, 74] is:

$$\begin{aligned} E &= -\sum_{i=1}^{n} p_i \log_2(p_i), \\ &\quad \text{where } p_i = \text{ probability of outcome } i \end{aligned} \tag{2.1}$$

Equation 2.1 is plotted in Figure 2.1 with Entropy E on the ordinate and the probability of an outcome being 1, $p(1)$, on the abscissa. The maximum entropy in Figure 2.1 occurs when 0 and 1 are equally likely. Consider an 8-input AND

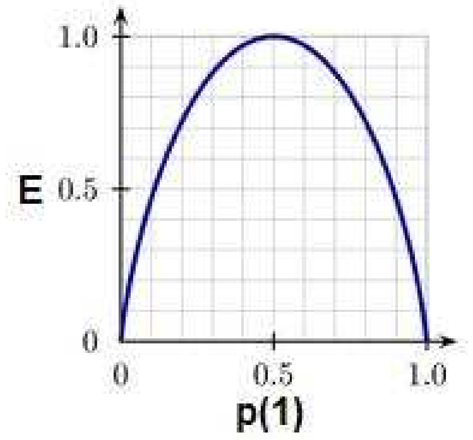

Figure 2.1: Entropy, E, vs. $p(1)$ for a signal

gate, with $p(0) = p(1) = 0.5$ on each input. From SCOAP measures [26], the difficulty of setting the output to a 1 is $CC(1) = 9$, and the difficulty of setting it to a 0 is $CC(0) = 2$. From COP measures [49], the probability of the output being 1 is $p(1) = 1/256$ and $p(0) = 255/256$. However, the entropy measure of the output, which incorporates both 0 and 1 values, is 0.03687.

Dussault proposed the first information theoretic testability measure [23]. He proposed a way to calculate the entropy and conditional entropy of the gates in the circuit. Conditional entropy is the average information required to specify an event $x \in X$ after $y \in Y$ is known as shown in Equation 2.2. X and Y are events of logic 0 or logic 1 occurring at a gate or a primary input or a primary output.

$$E(X|Y) = -\sum_{x,y} P(x,y)\log_2 P(x|y) \tag{2.2}$$

He also defines the following measures for a circuit with inputs being X and outputs being Y:

$$\begin{aligned}
\text{Observability measure} &= \frac{1}{E(X|Y)} && (2.3)\\
\text{Controllability measure} &= \frac{1}{E(Y|X)} && (2.4)\\
\text{Testability measure} &= I(X;Y) = E(X) - E(X|Y) && (2.5)\\
&= E(Y) - E(Y|X)
\end{aligned}$$

Based on these measures he proposes to select test points for the circuit. Lines with low observability measures are observability test point candidates, lines with low controllability measures are controllability test point candidates, lines with low testability measures are complete test point candidates. He also proposes to calculate these measures using logic simulation or using Parker-McCluskey Equations [52]. He does not report any experimental results for his ideas, but the concepts he proposed were investigated further by Agrawal [3] and Thearling and Abraham [64].

Agrawal proposed an information theoretic approach to testing digital circuits [3] and derived the probability $P(T)$ of detecting a stuck-at fault by a vector sequence T as:

$$P(T) \;=\; 1 - 2^{-E_o T/k} \tag{2.6}$$

where k is the number of lines through a circuit partition where the detectable fault exists and E_o is the entropy at the output of the circuit. Consider a 2-input AND gate with inputs $i1$, $i2$ and output Z. If the probability of logic 0 (logic 1) occurring at the inputs is 0.5 (0.5), the entropies at the inputs, $i1$ and $i2$, are:

$$E_{i1} = E_{i2} = -0.5\ \log_2(0.5) - 0.5\ \log_2(0.5) = 1.0 \tag{2.7}$$

Therefore, the total information present at the inputs is 1+1 = 2. The probability of logic 0 (logic 1) at output Z is 0.75 (0.25) and the entropy of Z is:

$$E_Z = -0.25\ \log_2(0.25) - 0.75\ \log_2(0.75) = 0.811 \tag{2.8}$$

So, the AND gate has information loss of $2.0 - 0.811 = 1.189$. Agrawal proposed an ATPG method that reduces the loss (by increasing entropy) of information and maximizes $P(T)$ in the circuit by adjusting the probabilities of 0 and 1 at the inputs. He discusses a way to generate test patterns that increases the information flow in the circuit based on the functionality of the circuit. Since he uses the functionality of the circuit to generate patterns, different implementations of the same functionality can produce different results. By increasing the information flow in the circuit, he is able to detect permanent faults and intermittent faults, faults that become active intermittently.
For sequential circuits, he calculates the information flow on a per pattern basis. Consider a circuit that is designed to perform n operations. Let us assume that the j^{th} operation requires an input sequence of v_j patterns and can produce m_j distinct output sequences. The information output of this operation will be maximum when each of the m_j output sequences are made equiprobable. Then the average information output of j^{th} operation is given as:

$$h_j = \frac{log_2 m_j}{v_j} \text{ bits/pattern} \tag{2.9}$$

If the probability of executing j^{th} operation is p_j, the average information output of the circuit is:

$$H_o = \sum_{j=1}^{n} p_j \times h_j \text{ bits/pattern} \tag{2.10}$$

In order to generate patterns, first, p_j's are assigned to the various operations such that the output information as given by Equations 2.10 and 2.9 is maximized. The pattern generator would then proceed by selecting operations with the assigned probabilities and generating an input pattern sequence for the selected operation. An input pattern sequence is generated from the functional description of the circuit and data patterns are selected to make all possible outputs equiprobable. These patterns are then stored as the test patterns for the circuit.

Thearling and Abraham proposed information theory based testability measures at the functional level [64]. They use relative information and mutual information measures instead of absolute values. They use the *information transmission co-efficient* (InTC) as their testability measure. InTC is defined as a measure of the fraction of information that can be transmitted through a line. It is computed by taking the ratio of the entropy on the inputs of a line and the entropy of the outputs of the line. They use InTC as their controllability measure. For an observability measure, they use the ratio of mutual information of an output with an internal circuit node and the entropy of the output. Probabilities of logic 0 (logic 1), $p(0)$ ($p(1)$), were estimated either via logic simulation, called *sampling*, or by using functional level information of the circuit components, known as *composition.*

They also proposed partitioning of the circuit, to improve testability, using entropy-related measures. By partitioning a circuit, increased controllability and observability can be achieved through DFT insertion. When data flows through a circuit, it may be difficult to produce desired values on the inputs of some gates. Similarly, data from gates may not always flow to the outputs. To alleviate this problem, the circuit is partitioned and additional controllability and observability are added. Their basic idea is to use their testability measures to determine the amount of information compression occurring in a circuit and then to use that knowledge to perform partitioning of the circuit. They provide an algorithm to effectively partition the circuit. The basic premise of the partitioning algorithm is to first push the limits of the partition until signal paths have a controllability less than the desired threshold. The partition limits are then pulled back until each path in the partition has an observability greater than than the desired threshold. This is repeated until the entire circuit has been partitioned. Their results show that partitions with higher values of controllability and observability threshold achieve higher fault coverages.

From the above mentioned work, it is clear that the advantage of using entropy, rather than conventional testability measures, is that it can be reliably calculated using a limited number of simulation vectors and it is a single metric that accurately reflects the information flow in the circuit.

2.2 Partial Scan

2.2.1 Designing Circuits with Partial Scan

Agrawal *et al.* [2] discuss two methods of selecting flip-flops for scanning. In this scan design methodology, only selected faults are targeted for detection. The targeted faults are those not detected by the functional vectors and the test generator decides exactly which flip-flop to scan using one of the two ways. By using partial scan, they achieve 40% savings over the full scan overhead while using comparatively fewer test vectors to obtain similar fault coverages.

In the first method known as the Frequency Approach, all of the possible tests are generated for each targeted fault and then the test vectors that require the fewest flip-flops to be scanned are selected. This approach uses a modified *path oriented decision making* (PODEM) test generation program [25] to generate all possible tests for the targeted fault by only setting inputs that are needed to be set while leaving all other inputs as “don’t care’s.” The fault and corresponding vectors are analyzed to obtain a minimal set of vectors that covers the maximum faults but requires fewer flip-flops to be scanned. This is done by selecting vectors that detect hard-to-detect faults and removing all the faults that get detected by these vectors from the fault list. The flip-flops required by these vectors are scanned. Then the flip-flops and vectors that test the remaining faults, which can also be considered as flip-flops with higher frequency, are selected for scanning.

The second approach is the Distance Approach, which generates only one test per fault. The test vectors are generated by setting the *primary inputs* (PI) nearest to the fault site and propagating the fault to the nearest *primary outputs*

(PO). In the distance approach, all the PIs and POs are assigned 0 distance, while combinational logic whose inputs and outputs are derived from flip-flops is set to a distance value of 100. Once a flip-flop is scanned, these inputs and outputs get a distance value of 0.

They also provide an option table to the designer for each approach suggesting the fault coverage that can be obtained by selecting a certain percentage of flip-flops. So, the designer will have a choice of selecting the number of flip-flops that he needs to scan to obtain the desired fault coverage.

They achieve fault coverages comparable to full scan, but by only scanning a smaller percentage of flip-flops for the same circuit. The total vectors required for partial scan are less than half compared to full scan for some circuits. The frequency approach fares better than the distance approach as they obtain an optimized set of test vectors for the targeted faults. But, the test generation time for the frequency approach can be very high for big circuits as multiple vectors are generated for each fault. They also do not provide the test generation time for either of the approaches. They use PODEM [25] for test generation, which is an outdated algorithm and better test generation algorithms have been developed since.

2.2.2 An Exact Algorithm for Selecting Partial Scan Flip-Flops

Chakradhar *et al.* [12] develop an exact algorithm for selecting flip-flops in partial scan design to break all feedback cycles. They make use of graph transformations, partitioning schemes and *integer linear programming* (ILP) to develop their algorithm, which performs well on the ISCAS '89 benchmark circuits and several production VLSI circuits within reasonable computation time.

Their main idea in this algorithm is to cut all of the feedback paths except self-loops present in a circuit as proposed previously by Cheng and Agrawal [14] and Gupta and Breuer [27]. For this they model the circuit as a directed graph called the *S-graph* with the flip-flops in the circuit as vertices and the arcs (or edges) connecting the vertices modeling the combinational logic present in the circuit. The problem of selecting flip-flops to break all feedback cycles is equivalent to the problem of finding a set of vertices whose removal makes the S-graph acyclic. The vertex set is referred as the *minimum feedback vertex set* (MFVS) and this problem is an NP-hard problem. But, by using an MFVS-preserving graph transformation that defines a new class of graphs, they solve the problem in polynomial time complexity.

They propose three graph transformations that are MFVS-preserving. If there exists an arc from $(v_i \rightarrow v_j)$, then v_i is the predecessor of v_j and v_j is the successor of v_i. Let $remove(v_i)$ denote the process of removing all incoming and outgoing arcs of vertex v_i. The three transformations are as follows:

- **T1:** If v_i has a self-loop then $remove(v_i)$ from the s-graph and return v_i. A MFVS for the S-graph is obtained by adding v_i to any MFVS of the modified graph.

- **T2:** If v_i has either indegree or outdegree equal to 0 then $remove(v_i)$ from the s-graph. The MFVS's of the S-graph and the modified graph are identical.

- **T3:** If v_i has either indegree or outdegree equal to 1 but no self-loop then $ignore(v_i)$, which connects each predecessor of v_i to all its successors, $remove(v_i)$ and collapses multiple arcs (if any) into a single arc. Any MFVS of the modified graph is an MFVS for the S-graph.

Each of these transformations will remove one vertex and they are applied to the S-graph until no more transformations are applicable. If these transformations reduce the S-graph to an empty graph, then this S-graph is called a *two-way reducible* graph and the MFVS can be determined in polynomial time complexity. If an S-graph is not two-way reducible then its MFVS can be computed in polynomial time, and it is classified as *scc-compressible.* If the S-graph is not scc-compressible then these transformations will reduce it to a final graph containing one or more *strongly connected components* (SCC). An SCC that cannot be further reduced is called a *compressed scc*.

The branch-and-bound partitioning scheme is used when the above transformations cannot reduce the graph completely. A Boolean variable x_i is assigned to each vertex v_i in the graph. For any assignment of 0-1 values to the Boolean variables, they construct a vertex set that includes only those vertices for which the corresponding Boolean variables assume the value 1. The 0-1 assignments that correspond to feedback vertex sets are called feasible solutions and the rest are infeasible solutions. The branch-and-bound procedure systematically searches the space of all 0-1 vectors for an optimum solution corresponding to the MFVS. This way the graphs that cannot be reduced by graph transformation are converted into vertex clusters using branch-and-bound partitioning.

The MFVS for the compressed SCCs is found by formulating the problem as an *integer linear program* (ILP). Weights w_i are associated with each v_i and start with the requirement that for every arc $(v_i \rightarrow v_j)$, $w_i - w_j \geq 1$. But for feedback loops, this will not be true. Hence they add n to the weight of v_i where n is the number of arcs in the longest cycle. Now the requirement can be expressed as: for every arc $(v_i \rightarrow v_j)$, $w_i - w_j + nx_i \geq 1$. Here x_i is a Boolean variable associated with v_i and is set to 1 whenever v_i is on a cycle. Now to find an MFVS, it suffices to minimize $\sum x_i$.

They compare their results with Lee and Reddy [41], PASCANT [6] and OPUS [17]. They compute optimal solutions for large circuits in ISCAS '89 benchmarks in less than a minute while the other methods only produce sub-optimal solutions. They also provide results for three production VLSI circuits. But, they do not show the results for the fault coverage obtained, which is the basic metric to compare the quality of the algorithms. Moreover, breaking the feedback loops present in a circuit alone is not enough to ensure that the overall testability of the circuit is increased.

2.2.3 Partial-Scan Design Based on Circuit State Information and Functional Analysis

Xiang and Patel [70] present a multi-phase flip-flop selection approach that breaks critical cycles and selects flip-flops based on conflict resolution. Critical cycles are broken using a combination of valid circuit state information and conflict analysis. Circuit state information is obtained using logic simulation and is updated after selection of a few flip-flops for scanning. Valid-state-based testability measures and conflict-based testability measures are presented in this work.

The definitions and notations provided by them to understand their work better are as follows:

Definition 1:

A state for a fault-free circuit is an assignment of Boolean values $\{0, 1\}$ to the output values of the flip-flops. The reset state is a state that can be reached from any state of the circuit.

Definition 2:

A state is called a *valid state* if it is reachable from the reset state; a state is called an *invalid state* if it is not reachable from the reset state. Let $v_1, v_2, ..., v_n \in \{0, 1\}$, $i_1, i_2, ..., i_k \in \{1, 2, ..., n\}$ and $k \leq n$. Assume that a state is an n-tuple, where n is the number of flip-flops in the circuit, and a partial state $(v_{i_1}, v_{i_2}, ..., v_{i_k})$ is called a partial invalid state if the corresponding state is invalid. State mapping maps a state to all of the cycles in the circuit, where each cycle contains a subset of circuit flip-flops.

Definition 3:

The vertices of the directed s-graph of a sequential circuit are the flip-flops of the circuit. There is an edge (v_m, v_n) in the s-graph if there is a combinational path from flip-flop v_m to flip-flop v_n.

Definition 4:

The *density of encoding* of a circuit is defined as $\frac{V}{2^n}$, where n is the number of flip-flops in the circuit, and V is the number of valid states in the circuit. The number of valid states V for a circuit with n flip-flops is usually much less than 2^n. When $\frac{V}{2^n}$ is much less than 1, the test generator may frequently justify invalid states. An invalid state indicates that backtracks are needed.

Definition 5:

A $conflict$ is defined as follows: A line l is assigned value v. In the previous process of test generation, l needs to be assigned the value v'. If the intersection of v and v' produces a value in the logic system, line l is assigned value $v \cap v'$; otherwise, a conflict occurs on l.

The conflict measures are calculated for the circuit using:

- $i - Controllability, C(i)$, of a node l, which is the potential number of conflicts and the number of clock cycles required in order to justify a signal requirement (l, i), where $i \in \{X, 0, 1\}$.
- $v - Observability, O_A(v)$ $(v \in \{D, \overline{D}\})$, reflects the number of conflicts or the number of clock cycles required to propagate fault effect v along the *easiest fault effect propagation* (EFEP) path.

These were adapted from their previous work [72], which gives a clear explanation on how the conflict analysis is done.

The valid state analysis is performed by logic simulating the circuit from either a reset state (if known) or by having all of the flip-flops in the unspecified state. Then, the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions for two continuous valid states is observed for each flip-flop. With the results from this analysis, they conclude that if a circuit has many flip-flops that do not transition much and if a circuit with cycles has few valid states, then the testability of the circuit is poor. Based on the valid state and conflict analysis, they define a new testability measure

$T(f,c)$ and *testability improvement potential* (TIP), which gives the improvement obtained by scanning a flip-flop. The summation of TIP with the controllability and observability values for each flip-flop is adopted for scan flip-flop selection.

They achieve high fault coverages on the ISCAS '89 benchmarks. They compare their results with OPUS [17] and COP [49], which use partial scan designs, ZSCAN [54], OPSCAN [69] and SDSCAN [20], which are implemented using valid state information, and SAMSON [34], which uses the implicit state enumeration method. They achieve better results than all of these algorithms on most of the ISCAS '89 benchmarks.

2.2.4 SPARTAN – A Spectral and Information Theoretic Approach to Partial-Scan

Khan *et al.* [35, 36] propose a greedy algorithm that analyzes the *circuit-under-test* (CUT) and selects *scan flip-flops* (SFFs) using three measures: spectral analysis of the flip-flop oscillations, entropy from information theory, and logic gate circuit level information to measure observability and controllability. They use the spectral analysis and entropy combination because spectral analysis incorporates only functional information and does not use any structural information of the CUT. The experimental results show that they achieve very high fault coverages comparable to full-scan designs.

To select flip-flops for scanning, they construct an s-graph from the circuit with each node as a flip-flop. They condense this s-graph to contain nodes that are only SCCs. They propose a two-step approach to select flip-flops for scanning:

- **Step 1:** Calculate correlation coefficients using spectral analysis and select flip-flops based on this information.

- **Step 2:** Calculate entropy information passing through each flip-flop and select flip-flops based on this entropy analysis

The final set of scan flip-flops is selected by combining the sets of candidates obtained by spectral analysis and entropy analysis.

The spectral analysis first logic simulates the circuit with random vectors. *Spectral coefficients* (SCs) are calculated based on the logic simulation values for each flip-flop and PO using the 16×16 *Rademacher-Walsh transform* (RWT). These coefficients describe the controllability of the flip-flop. The SC_{i_avg} is calculated for each flip-flop and PO using the following equations:

$$\begin{aligned} SC_{i\_avg} &= \frac{\sum_{i=1}^{k} w_i \times |SC_i|}{k} && (2.11) \\ w_i &= \frac{\varphi_i + 1}{\sum_{j=1}^{k} (\varphi_i + 1)} && (2.12) \end{aligned}$$

where φ_i is the sequency of a transform matrix row, which represents the number of $1 \rightarrow -1$ and $-1 \rightarrow 1$ transitions in a matrix row; $|SC_i|$ represents the SC of each row; and k represents the number of rows in the matrix. If SC_{i_avg} of a flip-flop and any of the POs in its fanout cone are equal, then the flip-flop is considered to have good observability. The flip-flops that have poor observability and have lower SC_{i_avg} than an *SC Threshold* are marked for scanning. Then, the largest SCCs in the condensed graph are selected and flip-flops in these SCCs that are marked for scanning are scanned. This ensures that the cycles in the circuit are also broken. The number of flip-flops that are scanned is limited to 25% of the total number of flip-flops present in the circuit.

The entropy analysis logic simulates the circuit with random vectors to compute $p(0)$ and $p(1)$, which are the probabilities of a flip-flop being 0/1, respectively. The entropy $H(Q)$ and average conditional entropies $H(D|PI)$ and $H(Q|PO)$ are:

$$\begin{aligned} H(Q) &= -(p(0)\log_2 p(0) + p(1)\log_2 p(1)) && (2.13) \\ H_{avg}(D|PI) &= \frac{\sum_{x=1}^{p} H(D|PI_x)}{p} && (2.14) \\ H_{avg}(D|PO) &= \frac{\sum_{x=1}^{q} H(D|PO_x)}{q} && (2.15) \end{aligned}$$

where p is the number of PIs in the fanin cone of the flip-flop and q is the number of POs in the fanout cone of the flip-flop. Now for each SCC, a flip-flop is scanned and the entropies are updated. Flip-flops that give the maximum improvement to the entropy of the SCC are scanned.

Experimental results show that SPARTAN performs better than the previous best result of MPSCAN [70] on all ISCAS '89 benchmark circuits. They achieve higher fault coverage with lower test volume than MPSCAN. But, they achieve this at the cost of scanning more flip-flops than MPSCAN.

2.2.5 Other Partial Scan Algorithms

A variety of partial-scan design methods have been developed to improve the fault coverage obtained by *automatic test pattern generators* (ATPGs). They can be classified into the following three categories: structure-based [4, 12, 13, 14, 16, 27, 40, 41, 51, 61], testability-measure based [1, 7, 16, 33, 34, 38, 49, 51, 54, 66, 71], and test-generation-based methods [29, 42, 43, 50, 60].

Cheng and Agrawal [14] proposed a structure-based method in order to break cycles and reduce sequential depth for the first time. They pointed out that test generation complexity may increase exponentially with the number of cycles present in the circuit and linearly with the sequential depth. Algorithms were presented to break cycles and reduce the sequential depth. Gupta and Breuer [27] presented a structure-based method that requires only combinational test generation and attains complete coverage of all detectable faults. Scan flip-flops are selected in such a way that the resulting kernel belongs to a so-called *balanced sequential structure* (B-structure), test generation of which can be treated as that of combinational logic. Jiang *et al.* [32] proposed a novel method to reduce the length of the synchronizing sequences by scanning several flip-flops of the sequential machines. Trischler [66] introduced a simple testability measure to select scan flip-flops, which is the first testability-analysis based partial-scan design method. Parikh and Abramovici [49] presented a partial-scan design based on a simple testability measure. The testability measure represents the number of clock cycles required to activate, propagate or detect a fault. Abramovici *et al.* [1] selected partial-scan flip-flops by untestability analysis. Many methods have been introduced to reduce test application time. Narayanan *et al.* [47] proposed an optimal k-scan chain configuration using dynamic programming, which can get an optimal solution in O($k \cdot N^2$) (N is the number of scan flip-flops).

2.3 Test Point Insertion in Non-Scan Designs

2.3.1 Self-Driven Test Structure for Pseudo-random Testing of Non-Scan Sequential Circuits

Muradali and Rajski [46] introduce a self-driven test point structure that permits at-speed, on-chip, non-scan, sequential testing using parallel pseudo-random test patterns applied only to the primary inputs of the circuit-under-test. The test network is unique in that aside from a test mode flag, all I/O signals needed for test system operation are tapped from within the circuit itself. They achieve high single stuck-at fault coverage for a number of ISCAS '89 benchmarks. This is the only paper to come out on non-scan *built-in self-test* (BIST) with test point insertion.

The testability measure they use is based on probability analysis of gate I/Os obtained after logic simulation. They analyze the fault-free information local to a circuit line, which can be used to quickly recognize regions that might hinder fault detection. The observability is estimated using the sampled OL_i, which is the probability that the bit value, i, at that gate input is observable at the gate output, obtained from a circuit tracing method similar to STAFAN [31]. Observability point candidates are selected by calculating observabilities of each gate as a weighted sum of the products of local observabilities evaluated along each combinational path backtraced from the output of a gate toward its inputs. That is, such a path concludes when a flip-flop is reached. Observabilities are calculated for every time frame by backtracing paths that start from each flip-flop encountered in the previous time frame. Therefore, the upper bound on the number of time frames is equal to the sequential depth of the longest path. Since sequential loops can exist, the number of time frames per calculation is limited.

Controllability point candidates are accumulated using threshold-driven backtracing techniques in which flip-flops are transparent, given an initial reference switching profile determined using logic simulation. This begins by identifying all lines with a OL_i value below a user defined block threshold (that is, blocking lines). Next, since the OL_i at a gate input is influenced by the bias at the other inputs to that gate, for each blocking line, the circuit is backtraced along the most controlling path commencing at the neighbor of the blocking line and terminating when an OL_i value above a user-defined stop threshold is reached. The backtrace may also begin at lines that switch below a threshold rate. For each backtrace, the appropriate controllability point can be determined by the start gate and the number of inversions encountered.

The most interesting part of their work is their self-driving test-point structure, which does not need any external input to drive it. This is done by extracting candidate test-point source lines from circuit regions suspected to be uncorrelated to the activity of the respective test points and connecting these source lines to the corresponding control points. They achieve high stuck-at fault coverage for all the ISCAS '89 benchmarks and the results show that this BIST scheme outperforms their *automatic test pattern generator* (ATPG) method. But, they assume that all flip-flops are initialized and do not include the area overhead due to the initializing hardware into their calculations.

2.3.2 Non-Scan Design-for-Testability Techniques for Sequential Circuits

Chickermane *et al.* [18] introduce a new technique of parallel loading of flip-flops in test mode for enhanced controllability combined with probe point insertion for enhanced observability. Selection of candidate flip-flops and probe points is done by their tool, OPUS-NS, which is their previous work. They achieve fault coverages higher than 96% and ATPG effectiveness improvement greater than 99.7%. ATPG effectiveness is the percentage of faults either detected or proven to be untestable within the gate-level logic simulation constraints. Their work is not fully focused on test point insertion but has good results.

Flip-flop selection is based on cycle cutting and SCOAP [26] measures. After fault simulation, observation points are inserted based on information from hard-to-detect faults. Another alternative to their method is to do fault simulation first and then target flip-flops based on hard-to-detect faults. But they have dealt only with their former idea in this paper.

Parallel loading of flip-flops means that the inputs of those flip-flops that are scanned will be directly connected to the PIs in test mode and hence they have an upper bound of scanning only as many flip-flops in a circuit as the number of PIs. For enhancing the controllability of the circuit, some flip-flops are selected to be scanned depending upon the criteria that they help in cutting cycles in the circuit and based on SCOAP measures. They calculate a variable called *profit* depending upon the improvement of SCOAP values of a circuit by scanning a flip-flop and select those with high profit. They do not provide a clear idea on an *observability point* (OP) selection procedure but propose two schemes to reduce pin overhead due to the OP insertion. One is to use XOR gates to compress their outputs into one pin and the other is to connect the outputs of these points to POs making only one of the POs or OPs observable at any given time.

They use HITEC [48] for test generation and obtain high fault coverages for selected ISCAS '89 benchmarks and achieve high improvements in HITEC performance. But, their parallel load scheme requires as much hardware as normally scanning the flip-flop and does not provide any better results. Their results are beaten in the paper discussed next.

2.3.3 Non-Scan Design for Testability for Synchronous Sequential Circuits Based on Conflict Resolution

Xiang *et al.* [72] propose a non-scan design-for-testability method for synchronous sequential circuits. A testability measure called *conflict* based on conflict analysis in the process of synchronous sequential circuit test generation is introduced. Reconvergent fanouts with non-uniform inversion parity are still one of the main causes of redundancy and backtracking in the process of sequential circuit test generation. A new concept called *sequential depth for testability* is introduced to calculate the conflict-analysis-based testability measure. Potential conflicts between fault effect activation and fault effect propagation are also checked because they are closely related. The testability measure estimates the number of potential conflicts to occur or the number of clock cycles required to detect a fault. The non-scan design for testability method based on the conflict measure can reduce many potential backtracks, make many hard-to-detect faults easy-to-detect and make many redundant faults testable; therefore, it can greatly enhance the fault coverage of the circuit. It is believed that non-scan design for testability using the conflict measure can improve the actual testability of a circuit. They present extensive experimental results to demonstrate the effectiveness of the method.

Some definitions are provided in their paper that will help one to understand the calculation of *conflict* measures,

Definition 1:

A *conflict* is defined as follows: A line l is assigned value v. In the previous process of test generation, l needs to be assigned value v'. If the intersection of v and v' produces a value in the logic system, line l is assigned value $v \cap v'$; otherwise, a conflict occurs on l.

Definition 2:

Inversion parity of a path is defined as the number of inversions in the path modulo 2. Inversion parity inv_v(B, A) ($v \in \{0, 1\}$) from node A to B is defined as a two-bit binary number with these meanings:

- **00** if there is no path from A to B or no signal requirement on node A in order to meet signal requirement (B,v).
- **01** if the easiest way to justify (B,v) passes only a path of odd inversion parity from A to B.
- **10** if the easiest way to justify (B,v) passes only a path of even inversion parity from A to B.
- **11** if the easiest way to justify (B,v) passes at least one path of even inversion parity and one path of odd inversion parity from A to B, respectively.

Definition 3:

Sequential depth for testability $seq_v(l, s)$ ($v \in \{0, 1\}$) from a fanout stem s to a line l is defined as the number of clock cycles required to justify a signal requirement (l, v) at the line l to the fanout stems in the easiest way.

The paper provides formulas for calculating inv_v(B, A) and $seq_v(l, s)$ for each type of node present in a circuit. The controllability and observability of each line is calculated similar to SCOAP [26] measures but they incorporate the inv_v(B, A) and $seq_v(l, s)$ values calculated for each gate into these measures. Then, they choose the lines with hard faults and their immediate successors and predecessors as *test point candidates* (TPCs) on the basis of conflict. The *testability gain* (TG) function is calculated for each candidate from the conflict measures. Then, depending upon the requirement, they insert a 1-controllability point, a 0-controllability point, or an observability point and update the conflict measures. The inputs to controllability points are derived from existing PIs and easy-to-control nodes while taking into consideration the conflicts and reconverging fanouts introduced into the circuit due to this. Exclusive-OR chains are used to drive the observability point outputs to the POs.

They also provide a new test point structure that uses internal lines to drive the test points but, unlike Muradali and Rajski [46], they use extra pins to drive test points in some cases. They implement a system called *nscan* to test the non-scan circuit. They achieve higher fault coverages than OPUS-NS by Chickermane *et al.* [18] and HITEC [48]. Their test volume is higher than HITEC but they claim that this is because they manage to detect hard-to-detect faults, which HITEC does not. They have the best results for test point insertion in a non-scan design.

2.4 Test Point Insertion in Full-Scan and Partial Scan Designs

2.4.1 Constructive Multi-Phase TPI for Scan-Based BIST

Tamarapalli and Rajski [62] present a novel test point insertion algorithm based on a constructive methodology. They partition the circuit and conduct tests in multiple phases while inserting test points in each phase targeting a specific set of faults. Control points within a particular phase are enabled by fixed values, resulting in a simple and natural sharing of the logic driving them. By inserting a few test points and with a minimal number of phases, they achieve high fault coverages. In each phase, control point candidates are selected using a probabilistic fault simulation technique, which accurately computes the impact of a new control point in the presence of the control points selected so far. Observation points maximally enhancing the fault coverage are selected by a covering technique that utilizes the probabilistic fault simulation information.

Their BIST scheme contains a phase decoder block. The inputs of the phase decoder are driven by a pattern counter, which is part of the BIST controller. The outputs of the phase decoder, which indicate the current phase in progress, drive control points inserted in the combinational logic. Given the number of phases N, the phase decoder block can be synthesized with the number of outputs ranging from $\lceil \log_2 N \rceil$ to $(N-1)$ based on the constraints on routing and area overhead. In each phase, a set of control points is enabled by the phase decoder outputs and a specific number of patterns is applied. Each phase thus facilitates detection of a specific set of faults and contributes to the fault coverage obtained so far.

For every phase from 0 to (N-1), probabilistic fault simulation is performed to determine, for each node in the circuit, the list of faults that propagate to it, along with the associated detection probabilities. The conditional probability of a D or $\overline{D}$ occurring at an output when the inputs are $\{0, 1, D, \overline{D}\}$ is calculated for each node in the circuit. The node that offers a higher improvement to the testability of the circuit is selected as the control point candidate. The formulas for the probability calculation are given in the paper.

The observability point candidates are selected such that detection probability of a maximum number of faults meets a user specified threshold DTh. A three step process explained below is followed to achieve this objective.

1. Probabilistic fault simulation is performed to determine the propagation profile. This information is represented internally as a sparse two-dimensional matrix $T_{M \times N}$, with the collected nodes as rows, undetected faults as columns, and the probability of detecting a fault j, at a node i, as entry $T[i.j]$. The problem of selecting a pre-specified number of observation points now becomes equivalent to that of selecting the set of rows that maximizes the number of columns satisfying the detection threshold DTh.

2. The *partial covering* (PC_j) of a column j represents the approximate cumulative detection probability of the corresponding fault at nodes corresponding to the selected rows. A row i is said to cover a column j if $PC_j < DTh$ and $PC_j + T[i, j] \geq DTh$. Let $W_i = \sum_{j=1}^{N} max(0, min((DTh - PC_j), T[i, j]))$ denote the weight of a row i, where $i = 1$ to M. The selection

of rows is performed iteratively and in each step, the row that covers the maximum number of columns is selected. When two or more rows cover the same number of faults, their weight function is used to select them. This selection process continues until a pre-specified number of observation points are selected or no observation point meets the minimum-benefit-per-cost criterion.

3. Following the completion of the selection process, an improvement of the set of selected rows is performed. Since the selection of a row at iteration i does not consider the effect of rows selected in subsequent iterations, the final number of columns covered by such a row could be less than the number of columns covered by it at the point of selection. Hence, the selected row that covers the least number of columns at the end of the selection process is returned and the partial cover of affected columns is changed accordingly. The best unselected row is then determined. The returned selected row is replaced by the best unselected row, if it covers fewer columns than the latter.

The advantages of their method are its ability to converge, due to partitioning, and increased accuracy in predicting the impact of multiple control points, due to the usage of fixed values. In addition, the number of ineffective control points is reduced, since conflicting control points are enabled in different phases. They achieve optimal or near optimal fault coverage for large benchmark circuits with only few phases. But, the disadvantage with their method is that they need to fault simulate for every phase and this would increase the design time for large circuits if the fault simulation takes longer.

2.4.2 Timing-Driven TPI for Full-Scan and Partial-Scan BIST

Cheng and Lin [15] introduce a gradient-based approach to estimate the random pattern testability improvement factors for the test point candidates of either full-scan based or partial-scan based BIST. They also propose a symbolic computation technique to compute testability for circuits under the partial-scan based BIST scheme. They use a greedy approach to select one test point at each iteration. The selection is based on a cost function that comprises the profit of global random pattern testability and the penalty of timing. Timing analysis is done to determine the slack available in each node as the candidate test points can only be at the cell boundaries but not inside the cells as they could disrupt the timing of the chip.

An observation point is inserted at a node to improve the observabilities of the node and all the other nodes that feed the node directly or indirectly. The effect of inserting an observation point increases the observability of all of the nodes present in the fanin cone of the observation point. Actual implementation of an observation point is done by simply connecting the node to the output data compactor. A control point is inserted at a node to improve controllabilities as well as observabilities of nodes in a circuit. Changing the controllability of a node changes also the controllabilities of nodes in the fanout cone of the node. The 1-controllability C_s and observability O_s of a node s are calculated using the COP [49] measures. They define a cost function U based on these measures as:

$$U = \frac{1}{|F|}\left(\sum_{i \in F} \frac{1}{pd_i}\right) \tag{2.16}$$

$$pd_i = C_s \cdot O_s \tag{2.17}$$

$$pd_i = (1 - C_s) \cdot O_s \tag{2.18}$$

where pd_i is the reciprocal of the expected test length of the fault at i, F represents the fault set and $|F|$ is the cardinality of F. Equation 2.17 represents the pd_i for i being a stuck-at 0 fault at s and Equation 2.18 represents the pd_i for i being a stuck-at 1 fault at s.

For better use of the cost function U, they define the controllability gradient as in Equation 2.19 and the observability gradient as in Equation 2.20:

$$G_{C_s} = \frac{dU}{dC_s} \tag{2.19}$$

$$G_{O_s} = \frac{dU}{dO_s} \tag{2.20}$$

To improve the testability measure still further they make use of a *cost reduction factor* (CRF) to approximate the Actual Cost Reduction, which is the reduction of U due to insertion of a test point. The CRFs of OR and AND gates and those of inserting an observability point are given in the paper. Based on the slack and CRFs at each node, they select test point candidates to increase the fault coverage. The difference between the implementations of this algorithm in full-scan and partial-scan BIST is only during the calculation of the COP measures. Since partial-scan circuits will still have some flip-flops left in them, there will exist feedback loops for which C_s and O_s cannot be calculated easily. Due to the feedback loops, the equations of C_s and O_s no longer remain linear. To overcome this, they suppress any high order exponents contributed by the feedback loops to eliminate the inaccuracy caused by the statistical dependence among the fanout branches [52].

The experimental results indicate that test point insertion without taking timing into account usually yields a better fault coverage but with some performance degradation while zero performance degradation and a comparatively lower fault coverage of random patterns can be achieved by their timing-driven test point insertion algorithm. For some circuits, they claim they insert more test points in order to achieve a high fault coverage without inserting test points at critical nets. But, they do not explain how they will test faults present in these critical nets.

2.4.3 Zero Cost TPI Technique for Structured ASICs

Sethuram *et al.* [57] show different ways in which unused *multiplexers* (MUXes) and scan flip-flops in a *structured application specific integrated circuit* (SA) design can be re-configured to insert test points to drastically reduce test volume and test generation time. SAs are an alternative technology to full-custom chip designs and are different from cell-based ASICs by virtue of there being pre-defined logic, clock, power and test metal layers (thus reducing manufacturing time and cost) and pre-characterization of what is on the silicon (thus reducing design cycle time). Since only unused hardware is used, the proposed *test point insertion* (TPI) technique does not entail any extra hardware overhead. Test points are inserted using timing information, so they do not degrade performance. They also present novel gain functions that quantify the reduction in test volume and ATPG time due to TPI and are used as heuristics to guide the selection of signal lines for inserting test points. Experimental results show that they reduced ATPG time by up to 63.1% and test data volume by up to 64.5% while also achieving a near 100% fault efficiency for very large industrial designs.

Their work is an extension of Sethuram *et al.*'s [58] work where they insert only observation points using a novel gain function to increase the number of don't cares present in the test vector. In this paper they present four new test point designs, which use the unused flip-flops present in the SA, and describe their effectiveness and give the gain functions of each test point. Their main objective in this paper is to increase the number of don't cares in the test vectors by inserting control points, which enable the ATPG to not set some PIs that may be required to sensitize a fault site or propagate a fault effect. This will be accomplished using the control point. The four test points are:

1. A *conventional control point* (CP)

2. A *complete test point* (CTP)

3. A *pseudo-control point* (PCP)

4. An *inversion test point* (ITP)

The structure of the above test points and their gain functions are discussed elaborately in the paper. The gain function of inserting a CTP at line l is calculated as given in Equation 2.21:

$$G_{CTP}(l) = G^1_{CP}(l) + G^0_{CP}(l) + G_{OP}(l) \tag{2.21}$$

where $G^1_{CP}(l)$, $G^0_{CP}(l)$, and $G_{OP}(l)$ are the gain functions of inserting a conventional 1-control point, a conventional 0-control point, and an observability point at line l, respectively. These are calculated as follows:

$$\begin{aligned} G^1_{CP}(l) &= (F^1_C(l) + F^1_O(l)) \times N^1_C(l) && (2.22) \\ G^0_{CP}(l) &= (F^0_C(l) + F^0_O(l)) \times N^0_C(l) && (2.23) \\ G_{OP}(l) &= N_f(l) \times N_i(l) && (2.24) \end{aligned}$$

where $F_C^{1/0}(l)$ is the number of faults excited by a CP at l, $F_O^{1/0}(l)$ is the number of faults observed by a CP at l, $N_C^{1/0}(l)$ is the number of PIs or *pseudo-primary inputs* (PPI) that must be specified to control l, $N_f(l)$ is the total number of faults in the fanin cone of l, and $N_i(l)$ is the total number of PIs/PPIs that must be specified to observe the fault effect at l.

The gain function formulas for other test points are given in the paper and also an $O(n)$ algorithm is described to calculate the values of $F_C^{1/0}(l)$, $F_O^{1/0}(l)$, $N_C^{1/0}(l)$, $N_f(l)$, and $N_i(l)$ in the paper. Their overall algorithm is as follows:

- Collapse faults into a condensed fault set.
- Calculate the gain functions for all the nodes.
- Select a test point candidate.
- Update the circuit testability measures.
- Insert the test point into the circuit.
- Repeat the above three steps until the required test points are inserted.

Experimental results show that they reduce test generation time and test data volume on all of the industrial and ITC benchmarks. The ATPG CPU time is shorter in all cases except one for which they claim that the compaction of test vectors took significantly longer than for other methods. They also show results of inserting different test points into the circuit and a CTP insertion gives a better result than all others as it increases both the controllability and observability of the circuit. They reduce test generation time by 63.1% and test data volume by 64.5% overall. Since they use unused hardware in the SAs, no extra overhead is incurred but if their algorithm is to be used in a full-custom design, then the hardware overhead needs to be investigated.

2.4.4 SPARTAN – A Spectral and Entropy-based Partial-scan and Test Point Insertion Algorithm

Khan *et al.* [37] extend their greedy algorithm, SPARTAN [35, 36], that analyzes the *circuit-under-test* (CUT) and selects *scan flip-flops* (SFFs) to insert test points. The experimental results show that they achieve very high fault coverages comparable to full-scan designs with lower test volume and test application time than full-scan on all ISCAS '89 benchmark circuits except s38417.

They first scan the flip-flops in the circuit and then insert test points. They propose two test point insertion algorithms for selecting candidates:

- **TPI1:** In this algorithm, they randomly select candidate lines from the circuit and perform spectral and entropy analysis [35, 36]. Then they perform *observability primary input* (OPI) analysis on the remaining lines. OPIs are PIs that are needed to be set to propagate a fault effect to the PO and by inserting a CTP at a line l, all of the OPIs corresponding to the line l need not be set. Lines with large numbers of OPIs are selected for test point insertion.

- **TPI2:** In this algorithm, they get the value of *level_range* from the user and perform OPI, spectral, and entropy analysis on the lines that fall within the *level_range*.

Experimental results show that the partial scan algorithm with TPI1 achieves lower test volume and the partial scan algorithm with TPI2 achieves higher fault coverage than full-scan implemented with TRAN [11]. They have implemented the algorithm only for CTPs. They have not tried other types of test points such as conventional control points, observability points, and inversion test points. Their TPI1 algorithm randomly selects candidates for analysis, which can be done more cleverly.

2.4.5 Other Test Point Insertion Algorithms

Optimal placement of test points in circuits with reconvergent fanouts has been shown to be NP-complete [39]. Several approximate techniques for test point placement have been developed, which can be categorized depending on whether the technique uses fault simulation or testability measures to place the test points. For fault simulation techniques, the set of patterns generated by the test generator is used to fault simulate the circuit to identify the undetected faults. Test points are then inserted so that the undetected faults are detected. Iyenger *et al.* [30] use fault simulation to locate gates that block faults and insert test points at these gates to allow fault propagation. Touba and McCluskey [65] use path tracing to identify a set of test point solutions for each undetected fault and a covering algorithm that gives the minimum number of test points to detect these undetected faults. The limitation of fault simulation-based techniques is that they require test patterns ahead of time. Furthermore, for very big designs, fault simulation itself can take a long time, which can prolong the design process. Also, a late change in the design can change the test patterns and annul the test point analysis.

Testability measure-based techniques avoid these problems because they do not require any ATPG knowledge. The testability measures approximate the detection probability of random-pattern-resistant faults. Seiss *et al.* [55] form a cost function based on the COP [49] testability measures and then compute, in linear time, the gradient of the function with respect to each possible test point. The gradients are used to approximate the global testability impact for inserting each test point. The test point with the maximum benefit is inserted and COP measures are recomputed. The process continues iteratively until the testability of the circuit is satisfactory. Boubezari *et al.* [8] compute the testability measures at the *register-transfer level* (RTL) allowing RTL synthesis to take the test points into consideration when optimizing the design. Timing-driven test point insertion techniques have been developed to address the problem of adding test points on a critical timing path, causing the circuit to fail the timing requirements. Tsai *et al.* [67] compute the timing slack of each node and eliminate any node whose slack is not sufficiently long as a test point candidate. As the test points are inserted, the slack information is updated. The number of test points needed to achieve sufficient fault coverage may increase because test points cannot be inserted in some locations due to the timing constraints. Cheng and Lin [15] improve the work by Tsai *et al.* with COP measures and cost reduction functions.

2.5 Summary

After analyzing the quality of the results attained by the algorithms discussed, the best results for partial scan algorithm were reported by Khan *et al.* for their partial scan algorithm SPARTAN [35, 36], they improved on the previous best results obtained by Xiang and Patel [70]. The best results for test point insertion algorithm in a full scan design were reported by Sethuram *et al.* [57]. Xiang *et al.* [72] report the best results for their conflict-based test point insertion algorithm in a non-scan design and Khan *et al.* report the best results for their test point insertion algorithm SPARTAN [37] in a partial-scan design.

Simultaneous scan-flop insertion and test point insertion has never been tried before, which could increase the testability of the circuit better than a two-step approach of first scanning, and then inserting, test points. Inserting test points other than the CTP must also be investigated. Reducing the test volume and increasing the test application time while achieving high fault coverage is the direction partial scan and test point insertion algorithms need to take.

Chapter 3
Integer Linear Programming Approach and Formulation

3.1 Why We Chose an Integer Linear Program to Select Scan-Flops and Test Point Candidates

A *linear program* (LP), or in general, a Mathematical Program, is an optimization problem subject to constraints. Given an objective function and a set of constraints, the program will try to find a solution that will either maximize or minimize the objective function depending on the requirement. *Integer Linear Programs* (ILPs) are LP problems that require whole number, or integer, values of the variables in order to be properly solved. Since the solver needs to find integer solutions, Integer LP is harder to solve than normal LP problems. In a digital circuit, the number of scan-flops and test points that are being inserted will be a whole number, and hence, an Integer Linear Program is most suitable for DFT hardware insertion. This was demonstrated by Chakradhar *et al.* [12] who used ILPs to select partial scan flip-flops. From here on in this work, the acronym ILP refers to an Integer Linear Program. ILP algorithms are in many practical situations NP-hard with complexity of $O(2^N)$. The challenge lies in formulating an ILP algorithm that converges and is solvable in a reasonable amount of time.

3.1.1 Advantages of Integer Linear Programs over Greedy Algorithms

A greedy algorithm solves a problem by choosing the best candidate at every step and solves the subproblems that arise later. The choice made by a greedy algorithm may depend on choices made so far but not on future choices or on all of the solutions to the subproblem. It iteratively makes one greedy choice after another, reducing each given problem into a smaller one. In other words, a greedy algorithm never reconsiders its choices. So it may happen that the greedy algorithm ends up choosing the local maximum or minimum while there exists a global maximum or minimum for the subproblem, as shown in Figure 3.1. For example, let us consider the change-making problem. The change-making

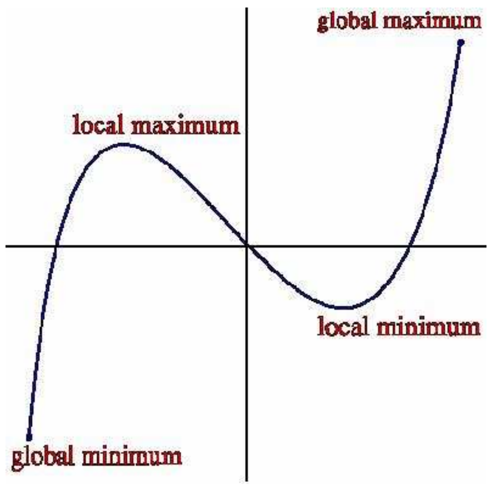

Figure 3.1: Range of values for a sub-problem

problem is a knapsack type problem, which addresses the following question: “How can a given amount of money be made with the least number of coins of given denominations?” A greedy algorithm of picking the largest denomination of coin that is not greater than the remaining amount to be made will always produce the optimal result. This is not automatically the case, though: if the coin denominations were 1, 3 and 4, then to make 6, the greedy algorithm would choose three coins $(4, 1, 1)$ whereas the optimal solution is two coins $(3, 3)$.
Hence, a greedy algorithm does not always find the optimum solution to the given

problem. The optimization algorithms such as LP or genetic algorithms are better than greedy algorithms in the sense that they always find the global maximum or minimum for a problem and its subproblem. The LP manages to find an optimal solution to all problems unlike a greedy algorithm and, hence, is better suited for DFT insertion where the hardware overhead is a big issue.

3.2 Integer Linear Program Formulation

The ILP is formulated in terms of sets, parameters, variables, a cost function, and constraints.

3.2.1 Sets

A *Set* in an AMPL [24] model defines a collection. The various sets we use in our AMPL model are the following:

- **gate:** Set of all the logic gates present in the circuit
- **flop:** Set of all the flip-flops present in the circuit
- **cycles:** Each SCC in the circuit is represented by a cycle number. Set *cycles* represents the set of all cycle numbers of SCCs present in the circuit, with size greater than 1, that is, there are more than 1 flip-flop in that SCC.

3.2.2 Parameters

Parameters can store a single scalar value or a collection of values indexed by a set. The following are the parameters used in the model:

- **L:** The total number of logic gates present in the circuit. L should always be greater than 0 as a circuit will have some logic gates present.
- **F:** The total number of flip-flops present in the circuit. F is 0 for a purely combinational circuit and greater than 0 for a sequential circuit.

- **nCycles:** The total number of SCCs present in the circuit. Parameter *nCycles* can be greater than or equal to 0.
- **$\mathbf{E_i}$:** The testability measure of i, where i can be a gate or a flip-flop. The entropy of any gate or flip-flop will always lie between 0 and 1.
- **SCC:** The collection of flip-flops present in each SCC.
- **wPS:** The Partial Scan weight. Parameter wPS governs the number of scan-flops being inserted into the circuit and its value is between 0 and 1. The higher the value, the fewer scan flip-flops inserted.
- **wTP:** The Test point weight. Parameter wTP governs the number of test points being inserted in the circuit and its value is between 0 and 1. The higher the value, the fewer test points inserted.

3.2.3 Variables

Variables are those whose values are determined by the AMPL solver. The following are the variables used in the formulation:

- **$\mathbf{X_i}$:** The state of logic gate i. If 0, the gate is not a test point candidate, else if 1, the gate is a test point candidate.
- **$\mathbf{Y_i}$:** The state of flip-flop i. If 0, the flip-flop is not a candidate for scanning, else if 1, the flip-flop is a scan flip-flop candidate.
- **T:** The total testability is the sum of entropies of all gates and flip-flops. It is calculated using the Equation 3.1.
- **nPS:** The number of scan flip-flop candidates. It is equal to the sum of the Y_i.
- **nTP:** The number of test point candidates. It is equal to the sum of the X_i.

- **DFT:** The total number of DFT candidates. It is equal to the sum of nPS and nTP.

Equation 3.1 gives the formula for calculating the total testability T of the circuit.

$$T = \sum_{i \in gate} (X[i] + E[i] \times (1 - X[i])) + \sum_{j \in flop} (Y[j] + E[j] \times (1 - Y[j])) \tag{3.1}$$

Equation 3.1 will be easier to understand when split into smaller equations.

$$\begin{aligned}
T_1 &= \sum_{i \in gate} X[i] &(3.2)\\
T_2 &= \sum_{i \in gate} E[i] \times (1 - X[i]) &(3.3)\\
T_3 &= \sum_{j \in flop} Y[j] &(3.4)\\
T_4 &= \sum_{j \in flop} E[j] \times (1 - Y[j]) &(3.5)\\
T &= T_1 + T_2 + T_3 + T_4 &(3.6)
\end{aligned}$$

T_1 in Equation 3.2 is the sum of entropies of all test point candidates. Since inserting a test point or scanning a flip-flop will make the Entropy of that particular candidate go to 1, it is enough to add the state of test point candidates, $X[i]$, as they are already 1. T_2 in Equation 3.3 is the sum of entropies of all gates that are not test point candidates. A gate is not a test point candidate when $X[i]$ is equal to 0. The entropy coefficients of these gates can be identified by calculating $(1 - X[i])$ and then multiplying it by its corresponding entropy $E[i]$. T_3 in Equation 3.4 and $T4$ in Equation 3.5 are similar to T_1 and T_2 respectively, but they are calculated using the entropies of flip-flops. It is clear from all the above equations that $X[i]$ and $Y[i]$ are the variables whose values the ILP needs to determine inorder to calculate other variables and maximize the objective function.

3.2.4 Objective Function

The objective function is the quantity the ILP will try to maximize or minimize by assigning values to variables, in our model $X[i]$ and $Y[i]$. The objective function for our ILP model is given in Equation 3.7.

$$Maximize\ (T - wPS \times nPS - wTP \times nTP) \tag{3.7}$$

From Equation 3.7 we can observe that our ILP model tries to maximize the total testability, T, of the circuit while reducing the number of scan-flops and test points being inserted. From the testability T we subtract the number of scan flip-flop candidates and test point candidates chosen, multiplied by their corresponding weight factors. With this objective function, we are guiding the ILP to choose values for $X[i]$ and $Y[i]$ that will maximize the testability of the circuit with as little DFT hardware as possible.

3.2.5 Constraints

Constraints are conditions given to the ILP model that are to be satisfied while maximizing or minimizing the objective function. Constraints have to be chosen properly as they could be too restrictive and make the ILP unable to find any solution satisfying them. The constraints for the proposed ILP model are given by Equations 3.8, 3.9 and 3.10.

$$\begin{aligned} 0 &\leq \sum_{i \in flop} Y[i] \leq F & (3.8)\\ SCC[nCycles, F] \times Y[F] &\geq 1 & (3.9)\\ DFT &= \text{nCycles if nCycles} \geq 5 \text{ else } 5 & (3.10) \end{aligned}$$

The constraint in Equation 3.8 is the scan limit. This constraint is set to select a minimum of 0 scan flip-flop candidates to a maximum of all flip-flops in the circuit. This constraint can be varied according to the requirement, to limit the amount of scan flip-flops inserted in the circuit.

The constraint in Equation 3.9 is the cycle breaking constraint. When there are SCCs present in the circuit, this constraint asks the ILP to select at least one flip-flop as a scan flip-flop candidate in each SCC cycle. This way, the model ensures that any SCC present in the circuit will be broken. The number of scan flip-flop candidates to be selected in each SCC can be varied depending upon requirements.

The constraint in Equation 3.10 specifies the number of candidates to be selected for one iteration. Since the cycle-breaking constraint requires the ILP to select one flip-flop in each SCC, when the number of SCCs, *nCycles*, is greater than 5, the candidate limit is set to *nCycles*. When *nCycles* becomes less than 5, as SCCs are broken in each iteration, the ILP is asked to return 5 candidates for each iteration. The value 5 is arbitrarily chosen and can be varied depending upon the user, but this value should always be greater than 1. Setting this value to 1 constrains the model too much and the AMPL solver fails to converge to a solution. The candidate limit constraint is used as the *stopping criterion* during partial-scan with test points experimentation. The algorithm is set to terminate when the ILP model returns less than 5 candidates. This is done to ensure that the ILP has full control in deciding the optimal amount of DFT hardware to insert and once the ILP cannot find even 5 candidates, it translates to the testability of the circuit being high and, hence, the algorithm is terminated. For full-scan with test points experiment, the user was allowed to decide the number of test points to insert into the circuit.

3.3 Convergence of the ILP

The complexity of an ILP is $O(2^N)$ and for some cases an ILP is categorized as an NP-hard problem. The discussed formulation converges to a solution when the number of DFT candidates it is asked to return is greater than 1. Even though the ILP is of exponential complexity, the AMPL solver manages to find a solution in a short time and hecne, the ILP can be used for practical purposes. This is evident from the results, which are discussed in Chapter 5. As the size of the circuit increases, the number of logic gates and flip-flops increase and hence, the number of variables for the ILP also increase. The AMPL solver is found to take longer time to solve the model for larger circuits but it has always converged for all benchmark circuits.

3.4 Summary

Chapter 3 introduces the concepts of an integer linear program and why we chose an integer linear program to select the scan-flop and test point candidates. Integer linear programs are ILP problems that require the solutions to be integers and are harder to solve than normal ILP problems. Integer ILPs are advantageous over greedy algorithms as greedy algorithms choose the best option that is available at every step but may end up choosing a local maxima or minima while the ILP always works toward the global maxima or minima. Section 3.2 discusses the ILP formulation in terms of sets, parameters, variables, a cost function and constraints. A set defines a collection, parameters can store values, variables are those whose values are determined by solver, the cost function is the function that the ILP will try to maximize or minimize and constraints are conditions given to the ILP model that are to be satisfied while maximizing or minimizing the cost function.

Chapter 4
Algorithm

This algorithm analyzes the circuit and selects candidates for test point insertion and scanning using six measures:

- Probability
- Entropy
- Unbiased Entropy
- Accumulated Entropy
- Entropy Gain
- Entropy Differential
- Spectral Coefficients

A good candidate will enhance the testability of the circuit, which allows the ATPG to attain higher fault coverage with fewer test patterns. The entropy analysis uses the structural information of the circuit while the spectral analysis uses the functional information of the circuit. The chapter explains each step of analysis being carried out on the circuit immediately after it is read from the input file.

4.1 Strongly Connected Components

In a sequential circuit, an *s-graph* [68] is the logic circuit graph where each node represents a flip-flop and an edge between two nodes is a path through combinational logic between the flip-flops. When the s-graph is condensed, each node

of a condensed s-graph is called an SCC. The SCC is a graph where all the vertices are reachable from each other. The algorithm to form SCCs for a circuit graph G is obtained from Introduction to Algorithms [19], which is as follows,
STRONGLY-CONNECTED-COMPONENTS(G)

1. Call DEPTH-FIRST-SEARCH(G) (DFS) to compute finishing times $f[u]$ for each flip-flop u. The finishing time $f[u]$ is the time stamp that records when the DFS finishes examining u's adjacency list.
2. Compute the transpose, G^T.
3. Call DEPTH-FIRST-SEARCH(G^T), but consider the flip-flops in order of decreasing $f[u]$.
4. Output the vertices of each tree in the depth-first forest formed in above step as a separate SCC.

SCCs are formed every time after a flip-flop is scanned and not after a test point is inserted in the circuit. Scanning a flip-flop means replacing the flip-flop with a pseudo-primary input and pseudo-primary output. Hence, the cycle that the flip-flop has been removed from may not exist anymore or it could have broken down into smaller cycles. So the SCCs have to be formed after scan insertion.

4.2 Logic Simulation

The logic simulator used is a pseudo-random logic simulator that simulates 32 vectors in parallel and stores the logic value of each gate in a 32 bit integer. First, the state machine is initialized to a randomly-chosen state. Warm-up simulation is performed with a user-specified number of warm-up random vectors. Flip-flop states and output logic values of each gate are obtained via logic simulation. The number of ones occurring at the output of a gate or flip-flop is counted and stored. At the end of the simulation, the probability $p(1)$ is calculated using Equation

4.1 and probability $p(0)$ is calculated as $1 - p(1)$.

$$p(1) \quad = \quad \frac{\text{Number of ones occurring}}{\text{Total number of random vectors}} \tag{4.1}$$

Entropy recalculation after inserting a test point or scan-flop can be done the same way as described above, but fewer random vectors are used, which reduces the run time. Iterative simulation is performed using a user-specified number of iterative random vectors for entropy recalculation. As explained already, this is one advantage of entropy as a testability measure, that it can be reliably calculated using a limited number of simulation vectors. While selecting a scan-flop or test point candidate, one of the important assumptions made is that the entropy of the node becomes 1.0 after inserting a test point at the gate or scanning the flip-flop. To make this assumption a valid one, the scan-flops and test points are given an alternating 0 and 1 sequence as input during entropy recalculation. Doing this ensures that equal number of logic 0's and 1's appear at the scan-flops and test points, thus making their entropy 1.0 and validating the assumption. This step is required so that we update the entropies of the gates lying in the fan-out cone of the candidates, even if we already know the entropy of the candidate.

4.2.1 Advantages over Parker-McCluskey Equations

Parker and McCluskey proposed equations to calculate the $p(1)$ of any gate, called *Parker-McCluskey equations* (PME) [52]. These equations use probability principles to calculate $p(1)$ and are faster than logic simulation but to calculate the probabilities accurately, it requires exponential time. Seth and Agrawal developed PREDICT [56] that calculates the probability accurately using PME but their computational complexity is exponential in the size of the circuit. Probabilities can be calculated quickly using PME but they will not be accurate. This can be understood better if we analyze the circuit in Figure 4.1. Let us assume that $p(1)$ for the inputs a, b and c are 0.5 and using PME we get the value of $p(1)$ of output z to be 0.15625. Table 4.1 gives the truth table for the circuit. From the table we can calculate the value of $p(1)$ of z to be 0.25. We can clearly see the discrepancy in

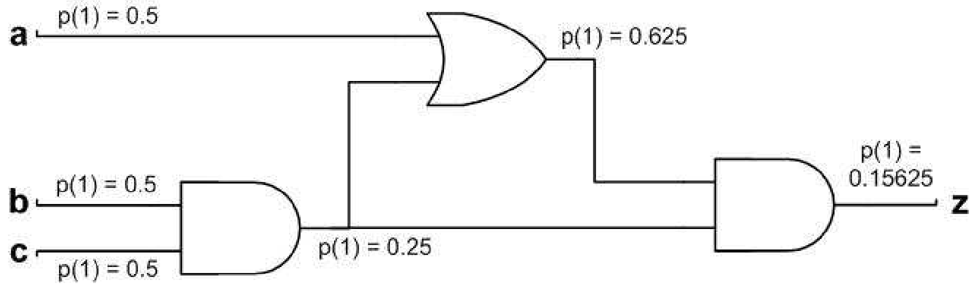

Figure 4.1: Example circuit

the values of actual probability and PME probability calculated. Logic simulation on this circuit would yield the actual probability even if it is slower than PME calculations. Since signal probabilities and entropies calculated from them are the core of the algorithm, we need accurate values and logic simulation is faster for computing than either PME or PREDICT calculations.

Table 4.1: Truth table for example circuit.

a	b	c	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

4.3 Entropy Calculation

After logic simulation, we have the probability information for all of the gates and flip-flops. Using this information, we can calculate the entropy of each gate using Equation 2.1. Pseudo-random logic simulation is not an exhaustive logic simulation. We simulate only a sample of vectors from the total available vector set for each circuit. Vector sampling could lead to biasing in the result as discussed by Heragu *et al.* [28] and Jain and Agrawal [31] and hence, the final result needs to be unbiased to capture the real property of the circuit. Let x_0 be the mean probability of all the gates in the circuit simulated for the sample of vectors and E_0 be the entropy corresponding to the mean x_0. x_0 and E_0 differ from the actual mean and entropy that the circuit would have when it is exhaustively simulated. This difference is the bias that needs to be calculated for every circuit. The expected value of Entropy of a gate with $p(1) = x$ is:

$$E(E) = \frac{1}{2\sigma \ln 2}\left[-\int_{x_0-\sigma}^{x_0+\sigma} x \ln x \, dx - \int_{x_0-\sigma}^{x_0+\sigma} (1-x)\ln(1-x)\, dx\right]$$

where σ is the standard deviation of the probabilities of the circuit.
$E(E)$ can be written as follows for easier understanding:

$$\begin{aligned} E(E) &= \frac{-1}{2\sigma \ln 2}\left[I_1 + I_2\right] \text{ , where} \\ I_1 &= \int_{x_0-\sigma}^{x_0+\sigma} x \; \ln x \, dx \\ I_2 &= \int_{x_0-\sigma}^{x_0+\sigma} (1-x)\ln(1-x)\, dx \end{aligned} \tag{4.2}$$

To solve I_1 we need to use integration by parts as shown below:

$$\begin{aligned} \text{Let} \qquad & u = \ln x \;\Rightarrow\; du = \frac{1}{x} dx \\ & v = \frac{x^2}{2} \;\Rightarrow\; dv = x\, dx \\ \text{Using} \qquad & \int u \, dv = u\, v - \int v \, du \end{aligned}$$

$$\begin{aligned} I_1 &= \left[\frac{x^2}{2}\ln x\right]_{x_0-\sigma}^{x_0+\sigma} - \int_{x_0-\sigma}^{x_0+\sigma} \frac{x}{2}\, dx \\ I_1 &= \left[\frac{x^2}{2}\ln x\right]_{x_0-\sigma}^{x_0+\sigma} - \left[\frac{x^2}{4}\right]_{x_0-\sigma}^{x_0+\sigma} \\ I_1 &= \left[\frac{(x_0+\sigma)^2}{2}\ln(x_0+\sigma) - \frac{(x_0-\sigma)^2}{2}\ln(x_0-\sigma)\right] \\ &\quad - \left[\frac{(x_0+\sigma)^2}{4} - \frac{(x_0-\sigma)^2}{4}\right] \\ I_1 &= \left[\frac{x_0^2+\sigma^2}{2}\ln\left(\frac{x_0+\sigma}{x_0-\sigma}\right) + x_0\sigma\ln(x_0^2-\sigma^2)\right] - \left[\frac{4x_0\sigma}{4}\right] \end{aligned} \tag{4.3}$$

I_2 can also be solved in a similar procedure as I_1 to get:

$$\begin{aligned} I_2 &= \left[\frac{(1-x_0)^2+\sigma^2}{2}\ln\left(\frac{1-x_0+\sigma}{1-x_0-\sigma}\right) + (1-x_0)\sigma\ln\left((1-x_0)^2-\sigma^2\right)\right] \\ &\quad - \left[\frac{4\sigma}{4} - \frac{4x_0\sigma}{4}\right] \end{aligned} \tag{4.4}$$

Substituting Equations 4.3 and 4.4 in Equation 4.2, we get:

$$\begin{aligned} E(E) &= \frac{-1}{2\sigma\ln 2}\left[\frac{x_0^2+\sigma^2}{2}\ln\left(\frac{x_0+\sigma}{x_0-\sigma}\right)+x_0\sigma\ln(x_0^2-\sigma^2)\right] \\ &+\frac{-1}{2\sigma\ln 2}\left[\frac{(1-x_0)^2+\sigma^2}{2}\ln\left(\frac{1-x_0+\sigma}{1-x_0-\sigma}\right)\right] \\ &+\frac{-1}{2\sigma\ln 2}\left[(1-x_0)\sigma\ln\left((1-x_0)^2-\sigma^2\right)-\frac{4\sigma}{4}\right] \end{aligned}$$

Assuming that σ^2 is very small, we get:

$$\begin{aligned} E(E) &= -x_0\ln x_0-(1-x_0)\ln(1-x_0)+\frac{1}{2\ln 2} \\ &-\frac{x_0^2}{4\sigma\ln 2}\ln\left(\frac{x_0+\sigma}{x_0-\sigma}\right)-\frac{(1-x_0)^2}{4\sigma\ln 2}\ln\left(\frac{(1-x_0)+\sigma}{(1-x_0)-\sigma}\right) \\ E(E) &= E_0+\gamma \qquad (4.5) \end{aligned}$$

where γ is the bias:

$$\gamma = \frac{1}{2\ln 2}-\frac{x_0^2}{4\sigma\ln 2}\ln\left(\frac{x_0+\sigma}{x_0-\sigma}\right)-\frac{(1-x_0)^2}{4\sigma\ln 2}\ln\left(\frac{(1-x_0)+\sigma}{(1-x_0)-\sigma}\right) \qquad (4.6)$$

The γ in the above equations is the bias arising due to logic simulation with only a sample of vectors. This bias has to removed from the probabilities of the gates as it affects the quality of DFT hardware inserted in some circuits. To calculate the actual mean probability of the circuit from the expected value of the entropy calculated in Equation 4.5 is very complicated. Hence, the unbiasing factor for probabilities is calculated using Equation 4.7 and the unbiased probability $p_u(1)$ of each gate is calculated using Equation 4.8:

$$\begin{aligned} \theta &= \gamma/\beta \qquad (4.7) \\ &\text{where } \beta \text{ is found empirically} \\ p_u(1) &= \theta\times p(1) \qquad (4.8) \end{aligned}$$

Depending upon the mode the algorithm operates in, the corresponding probability values are chosen to calculate the entropy. Various modes of operation of the algorithm are explained later in this chapter. The steps involved in finding the value of β will be discussed in the results chapter.

4.4 Spectra Calculation

Spectral calculation is done after each logic simulation. Spectra of each gate can be calculated by two methods. The first method of calculating the spectra is by multiplying the responses of each gate with a 32×32 Hadamard matrix, but this multiplication is done by overlapping the responses for every gate for all the random vectors as proposed by Khan [37]. This method is found to be time consuming to calculate spectra for all the logic gates and flip-flops in the circuit. Hence a quicker and less accurate method is adopted, which is the non-overlapping method, that calculates the spectra without overlapping the responses for each gate.

$$H(1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{4.9}$$

$$H(2) = \begin{bmatrix} H(1) & H(1) \\ H(1) & -H(1) \end{bmatrix} \tag{4.10}$$

$$H(3) = \begin{bmatrix} H(2) & H(2) \\ H(2) & -H(2) \end{bmatrix} \tag{4.11}$$

$$H(4) = \begin{bmatrix} H(3) & H(3) \\ H(3) & -H(3) \end{bmatrix} \tag{4.12}$$

A 32×32 Hadamard matrix is chosen for spectra calculation because each step of logic simulation has 32 responses. The fourth order Hadamard matrix is represented in Equation 4.12. After one step of logic simulation, the 32-bit response of each gate is multiplied with each row of the Hadamard matrix. This multiplication yields 32 *spectral coefficients* (SC) corresponding to each row and gives the input vector's correlation with that row. The average of these SCs are calculated using Equation 4.13.

$$SC_{avg} = \frac{\sum_{i=1}^{32} w_i \times |SC_i|}{32} \tag{4.13}$$

$$w_i = \frac{\varphi_i + 1}{\sum_{j=1}^{32} (\varphi_j + 1)} \tag{4.14}$$

The w_i in Equation 4.14 is the weight attached to the i^{th} row of the Hadamard matrix and φ_i is the *sequency* of the i^{th} row, φ, which is analogous to frequency, is the number of $1 \rightarrow -1$ and $-1 \rightarrow 1$ transitions in that row. Matrix multiplication is a time consuming process and to repeat it for all the steps of logic simulation is very tedious. To facilitate the process, the -1 in the Hadamard matrix is replaced with 0 and each row is converted into a 32 bit binary integer containing only 1 and 0. Then the 32-bit logic response from a gate is bit-wise XNORed with each row of the transformed Hadamard matrix. This is equivalent to a multiplication of the logic response with the Hadamard matrix row. The SCs of each row are calculated using the following algorithm:

```
1. Repeat for $i = 1$ to 32
    (a) $result \leftarrow \overline{H(4)[i] \oplus logic\ response}$
    (b) Repeat for all 32 bits of $result$
        i. Peel one $bit$ off $result$ and if this $bit$ is 1, increment $SC[i]$, else
           decrement $SC[i]$
```

Replacing the matrix multiplication with bit-wise XNOR operation speeds up the spectra calculation considerably. The SCs calculated this way are normalized by dividing them by the number of times the above mentioned steps are repeated, which is equal to the number of random logic vectors divided by 32. SC and SC$_{avg}$ for each gate are calculated and stored.

4.5 Candidate Selection

Candidates for test point insertion and scan-flops are selected using the entropy and spectral measures that have been already calculated. Various ideas have been tried to achieve better results and these ideas have been incorporated in the algorithm as different modes of operation that can be run individually or simultaneously. In all of the modes, candidates connected to a PI or a PO are discarded. During each iteration, five test point and/or flip-flop candidates are selected. The reason for selecting five candidates was already explained in the ILP formulation chapter.

4.5.1 ILP Mode

The integer linear program formulation explained in the previous chapter is used to select five candidates based on the testability measures. In this mode, an AMPL compatible data file is written out, sent to the AMPL solver and the output from the solver is obtained. The candidates from the output file are read in and the best one among them is chosen.

4.5.2 Gradient Descent Mode

In this mode, candidates are chosen without using the integer linear program. The candidates are sorted into an array as soon as the testability measure is calculated. The array size is set to five in order to be consistent with the ILP mode. Gradient descent mode is slightly faster than ILP mode as the candidates are available immediately after all of the calculations are finished. Since the candidates are sorted into an array as soon as the testability measures are calculated, if there are more than one candidate with equal values, then the gate with the lowest gate number is put ahead in the array. This is the main disadvantage of gradient descent mode, as the ILP mode gives a random selection of candidates in the same scenario. Hence, the results will vary between gradient descent mode and ILP mode.

4.5.3 Probability Mode

Using the probability information from logic simulation, five candidates with the least $p(1)$ or $p_u(1)$ are selected for test point insertion and scanning. Out of these five candidates, the one with the lowest probability is selected.

4.5.4 Entropy Mode

Entropies of the gates are calculated using Equation 2.1 and five candidates with the least entropy are selected for test point insertion and scanning. Out of these five candidates, the one with the lowest entropy is selected.

4.5.5 Unbiased Entropy Mode

As explained earlier, calculating entropy without accounting for the bias that may arise due to vector sampling could reduce the performance of the algorithm. In this mode, the probabilities are calculated using Equation 4.8 and the entropy values are calculated from them. Five candidates are chosen with the least entropy and the best candidate is chosen as the one with the lowest entropy.

4.5.6 Accumulated Entropy Mode

After every iteration, the number of ones that occurred at each gate are cleared and the circuit is simulated to calculate the new ones count for each gate. In accumulated entropy mode, the ones count is not cleared but instead gets accumulated after every iteration. By doing this, the complete history of transitions at each gate is maintained and the probability and entropy values calculated from these accumulated count could be a better testability measure for the algorithm to select the DFT candidates. Five candidates are chosen with the least entropy and the best candidate is chosen as the one with the lowest entropy.

4.5.7 Entropy Gain Mode

For each logic gate and flip-flop in the circuit, the gain in entropy achieved by inserting a test point or scanning the flip-flop is calculated. For each candidate, the gain is calculated as the sum of gain in entropy of the candidate and the gain in entropy of the fan-out gates of the candidate. The gain in entropy of the candidate is calculated using Equation 4.15 as it is assumed that when a test point or flip-flop is scanned the entropy of the candidate becomes 1.0. The gains of fan-out gates of the candidate are calculated by assuming that the $p(1)$ and $p(0)$ of the candidate are 0.5 and using PMEs to calculate what the probabilities of the fan-out gates would be if the candidate is scanned or a test point is inserted. Using PME to estimate the probability is easier than logic simulation in this case. As we are using PME to calculate the probabilities for only the fan-out gates of the candidate, there is not much signal correlation involved and hence, the calculated probabilities will be correct. From these probabilities, the new entropies of the fan-out gates are calculated using Equation 4.16. The total entropy gain of the candidate is calculated using Equation 4.17.

$$\begin{aligned} G_0 &= 1.0 - \text{entropy (candidate)} & (4.15) \\ G_i &= \text{PME\_entropy (fanout}_i) & (4.16) \\ &\quad - \text{entropy (fanout}_i) \\ \text{entropy\_gain (candidate)} &= G_0 + \sum_i G_i & (4.17) \\ &\quad \text{where } i = 1 \text{ to \# of fanout gates of candidate} \end{aligned}$$

After calculating the entropy gains of all the gates, the five candidates with the highest entropy gain are chosen and the candidate with the highest gain among the five is chosen as the best candidate. Entropy gain mode can be used only in gradient descent mode as the ILP formulation only accounts for choosing the nodes with lowest values of any testability measure passed to it.

4.5.8 Entropy Differential Mode

For each logic gate and flip-flop in the circuit, the entropy difference between the gate and its fan-out gate is calculated and stored. If more than one fan-out gate exists, then the difference with the highest value is stored as shown in Equation 4.18.

$$\begin{aligned} D_i &= entropy\ (candidate) - entropy\ (fanout_i) \\ \text{entropy\_differential } (candidate) &= MAX(D_i) & (4.18) \end{aligned}$$

where i = Number of fanout gates of the candidate

After calculating the entropy differentials of all the gates, the five candidates with the highest entropy differential are chosen and the candidate with the highest entropy differential among the five is chosen as the best candidate. Entropy differential mode, like entropy gain mode, can be used only in gradient descent mode as the ILP formulation only accounts for choosing the nodes with lowest values of any testability measure passed to it.

4.5.9 Spectral Mode

The spectral measures calculated for each gate can be used in various ways for candidate selection. Each variation is implemented as a separate mode as follows:

Mode 1 – Entropy + Controllability Mode:

From the average SC calculated for each logic gate and flip-flop, the maximum SC$_{avg}$ and minimum SC$_{avg}$ are identified and the *spectral threshold* (SC$_T$) is calculated as the average of the maximum and minimum SC$_{avg}$ as shown in Equation 4.19. The gates with SC$_{avg}$ lesser than SC$_T$ are considered to have poor controllability. Mode 1 should be used in conjunction with any other mode that identifies the best candidate, as identifying the best candidate from many candidates with poor controllability is not possible.

$$SC_T = \frac{SC_{avg\_max} + SC_{avg\_min}}{2} \tag{4.19}$$

Mode 2 – Entropy + Observability Mode:

For each gate, the POs that lie in the fan-out cone of the gate are identified. Each of the 32 SCs of the gate is compared with the corresponding SC of each PO lying in the fan-out cone. If there is a PO such that all of its SCs match with all of the SCs of the gate, then the gate is considered to have good observability. If there is no such PO, then the gate is considered to have poor observability.

Mode 2 should be used in conjunction with any other mode that identifies the best candidate, as identifying the best candidate from many candidates with poor observability is not possible.

Mode 3 – Entropy + Controllability and Observability Mode:

This is the combination of Modes 1 and 2. Gates with poor controllability and observability are identified using the methods discussed in Modes 1 and 2, which are summarized below:

- if $(SC_{avg}(gate) < SC_T)$ $poor\_controllability(gate)$ = TRUE
- if $(SC_i\ (PO) == SC_i\ (gate))\ \forall i = 1$ to 32,
  then $poor\_observability\ (gate)$ = FALSE

Mode 3, like Modes 1 and 2, should be used in conjunction with any other mode that identifies the best candidate.

Mode 4 – Controllability and Observability Mode:

In this mode, five candidates with the least SC$_{avg}$ and poor observability are identified and the best candidate is chosen as the one with the lowest SC$_{avg}$. Entropy measures are not used in this mode.

4.6 DFT Insertion

In each mode, one best candidate is selected. If the candidate is a logic gate, then a test point is inserted at the output of the gate, otherwise if the candidate is a flip-flop, then it is scanned. If a selected candidate is a logic gate connected to a flip-flop, then the flip-flop is scanned instead of inserting a test point, to save on hardware overhead. The corresponding test point or scan flip-flop counter is incremented. The above process is repeated until the scan limit and/or test point limit are satisfied and, in the case of ILP mode, until the AMPL solver returns less than five candidates.

A way to speed up the algorithm would be to insert the selected five candidates in every iteration. After inserting the first candidate, the gates in the fan-in and fan-out cone of the candidate are marked to prevent further DFT insertion. So, if any of the remaining candidates are in the marked cones, they will not be scanned nor will any test point be inserted. When the next candidate does not fall in the marked cones, DFT is inserted. This way, a maximum of five times speed up can be obtained. The number of candidates selected in each iteration can be made into a user specified parameter.

4.7 Flow Chart

The flow chart in Figure 4.2 gives the top level view of the algorithm. It gives a clear idea of how a circuit is taken through each step of the algorithm until the end.

4.8 Complexity Analysis of the Algorithm

Assume P_i PIs, P_o POs, N logic gates, F flip-flops and L lines are present in the circuit. The circuit is logic simulated with V vectors.

- The SCC formation depends on the DFS on the s-graph. The complexity of DFS is $O(N + L + F)$ [19].
- The complexity of logic simulation is $O((V/32) \times (N + F))$.
- The complexity of entropy calculation is $O(P_i + P_o + N + F)$.
- The overlapping spectra of the circuit are calculated by bitwise XNORing a 32×32 RWT matrix with the 32 bit logic response of the gate and POs. As explained earlier, the SCs are found by sliding the RWT matrix over the response of the gates bit by bit. The complexity of this analysis is explained in detail by Khan [37]. The complexity of calculating overlapping spectra is $O((F + N + P_o) \times (V - 31) \times 32)$.

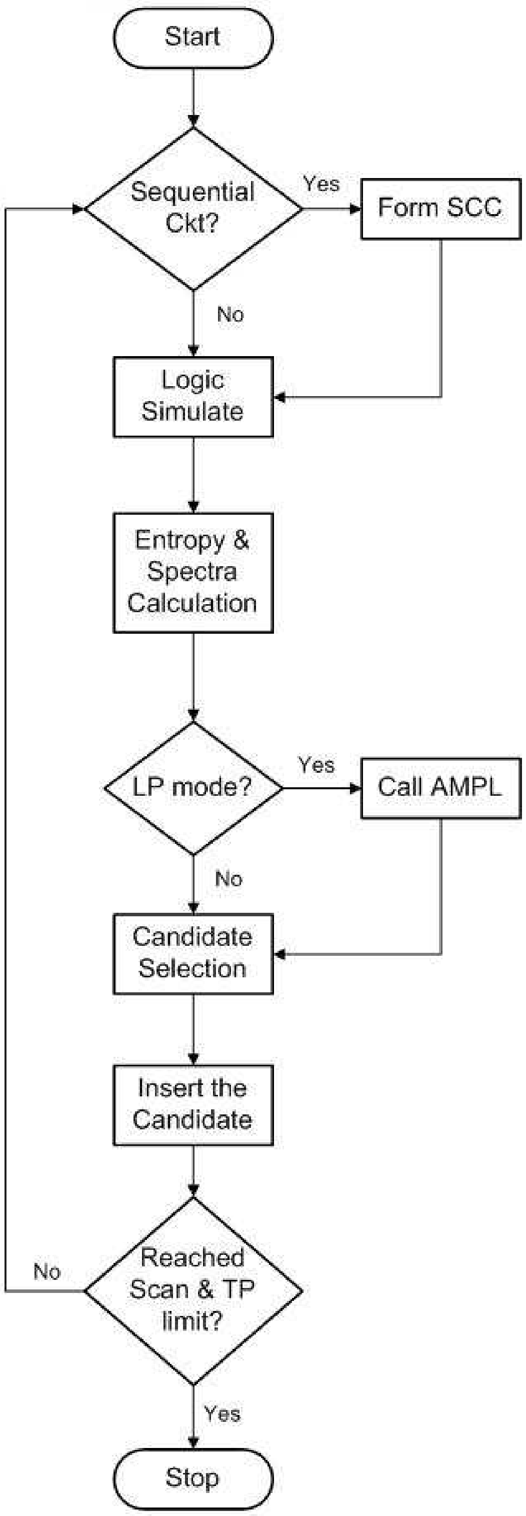

Figure 4.2: Flow chart of the algorithm

- The non-overlapping spectra of the circuit are calculated similarly to overlapping spectra but without analyzing the logic response bit by bit. The complexity of this analysis is $O((F + N + P_o) \times (V/32))$.

- The complexity of entropy gain and entropy differential analysis is $O(N+F)$.

- Gradient mode has the complexity of $O(P_i + P_o + N + F)$ in entropy mode, $O((F + N + P_o) \times (V - 31) \times 32)$ in overlapping spectra mode, $O((F + N + P_o) \times (V/32))$ in non-overlapping spectra mode and $O(N + F)$ in entropy gain or entropy diffrential mode. The ILP mode requires the data and output files to be transferred. This depends on the speed of the network and the size of the files. The time AMPL takes to solve the model is also dependent on the size of the circuit. All these overheads remain constant for a particular circuit. The complexity of the ILP is $O(2^N)$. Hence, the complexity of ILP mode is $O(L + N + F + 2^N)$ plus the complexity of the different modes listed above.

Depending on what mode the algorithm is using the complexity varies but logic simulation and SCC formation are required steps. The worst case complexity of the algorithm is when running in overlapping spectra mode, which would have the complexity of $O((V/32)\times(N+F)+(F+N+P_o)\times(V-31)\times 32)$. If $V \approx N, F << N$ and $P_o << N$, then the approximate complexity of the algorithm is $O(N^2)$. The complexity of non-overlapping spectra mode is $O((V/32)\times(N+F)+(F+N+P_o)\times(V/32))$. Applying the same assumptions as above, we get the complexity of non-overlapping spectra to be $O(N^2)$. The complexity of non-spectral ILP mode is $O((V/32) \times (N + F) + (L + N + F + 2^N) + (P_i + P_o + N + F))$, which becomes $O(2^N)$ with the assumptions. The complexity of non-spectral gradient descent mode is $O((V/32)\times(N+F)+(P_i+P_o+N+F))$, which becomes $O(N^2)$ with the assumptions. The overall complexity of the algorithm is $O(N^2)$ for the gradient descent mode and $O(2^N)$ for ILP mode.

4.9 Summary

This chapter introduces the algorithm used to select the scan-flop and test point candidates. Various ideas have been tried and implemented in the algorithm. Once the circuit netlist is read in, if it is a sequential circuit, then the SCCs are formed. Then logic simulation is performed using user-specified number of pseudo-random vectors. After logic simulating with each vector, the ones counts of all of the PIs, gates, flip-flops and POs are updated. If the algorithm is in spectra mode, then the SCs are calculated by bitwise XNORing the 32×32 RWT matrix with the logic response of gates, flip-flops and POs. After all of the vectors are simulated, the ones probabilities, $p(1)$, of PIs, gates, flip-flops and POs are calculated. Calculating probabilities by logic simulation is slower than using PMEs but is more accurate. After calculating the probabilities and SCs, entropy and SC_{avg} for each gate is calculated.

If the algorithm iruns in ILP mode, then a data file is written out based on the testability measure being used by the algorithm. Then the data file is sent to AMPL and the output file from AMPL with the selected candidates is read in. If the algorithm is in gradient descent mode, five candidates based on the testability measure used by the algorithm are chosen and the best among them is selected. In probability mode, the candidates with lowest $p(1)$ or $p_u(1)$ are chosen depending on the user's preference. In entropy mode, the candidate with lowest entropy is chosen. In unbiased entropy mode, the candidate with the lowest unbiased entropy is chosen. In entropy gain mode, the gain in entropy obtained by inserting a test point or scan-flop at each gate is calculated and the candidate with the highest entropy gain is chosen. In entropy differential mode, the difference in entropy between a gate and its fanout is calculated and the candidate with the highest difference is chosen. In spectra mode 1, the spectral threshold SC_T is calculated for the circuit and the candidate with lowest entropy and SC_{avg} less than SC_T is chosen. In spectra mode 2, observabilities of the gates are analyzed by comparing their SCs with the SCs of the POs in their fanout cones. If any of SCs do not match with POs, then the gate is considered to have poor observability. The candidate with lowest entropy and poor observability is chosen. In spectra mode 3, the candidate with lowest entropy, SC_{avg} less than SC_T, and poor observability is chosen. In spectra mode 4, the candidate with the lowest SC_{avg} and poor observability is chosen and entropy measures are not used. After the candidate is selected, if it is a logic gate, a test point is inserted after it; otherwise, if the candidate is a flip-flop, it is scanned. After DFT insertion, the algorithm repeats until the user-specified number of test points and/or flip-flops is inserted. The complexity of the algorithm is found to be $O(N^2)$ for the gradient descent mode and $O(2^N)$ for ILP mode.

Chapter 5
Results

In this chapter we present the *fault efficiency* (FE), *fault coverage* (FC), *test vector length* (V), *test volume* (TV), *test application time* (TAT) and *hardware overhead* (O) results obtained for partial scan with test point insertion and full scan with test point insertion. The partial scan results are compared with the best partial scan results reported by Khan *et al.* [35, 36, 37] for the ISCAS '89 benchmark circuits [9] and the full scan results are compared for the ITC '99 benchmark circuits [21] before and after the entropy algorithm was run on them.

5.1 Preliminaries

Fault Coverage

Fault coverage is defined as the percentage of faults detected by the ATPG for all faults present in the circuit.

$$\text{FC} = \frac{\text{Faults detected by ATPG}}{\text{Total faults in circuit}} \times 100$$

Fault Efficiency

Fault efficiency is defined as the percentage of faults detected by the ATPG for all testable faults present in the circuit. FE gives a better picture about the testability of the circuit than FC, which does not account for the untestable faults.

$$\text{FE} = \frac{\text{Faults detected by ATPG}}{\text{Total faults in circuit} - \text{Untestable faults}} \times 100$$

Test Vector Length

Test vector length is defined as the number of vectors generated by ATPG while testing the circuit. These vectors are stored in the *automatic test equipment* (ATE) memory and are loaded into the fabricated chip while testing. The test vector length reduction is calculated using Equation 5.1. V_{old} represents the vector length of the circuit before DFT insertion and V_{new} represents the vector length after DFT insertion. Parameters TV_{old} and TV_{new} and TAT_{old} and TAT_{new}, have analogous meanings.

$$\text{V Reduction (\%)} = \frac{\text{V}_{old} - \text{V}_{new}}{\text{V}_{old}} \times 100 \tag{5.1}$$

Test Volume

Test volume is the total number of bits stored in the ATE that will be fed into the chip being tested. The total number of bits applied to a circuit per test vector is the sum of the bits applied to each primary input and the number of bits shifted into the scan chain. Parallel vectors are vector sequences generated by the sequential ATPG program with scan flops and test points as Pseudo-PIs and Pseudo-POs. These parallel test sequences are then serialized according to the scan structure and the final serialized test vectors are developed. Each serialized vector includes the test vector that will be applied to the PIs and the scan-in and scan-out sequences for each parallel test sequence generated by the ATPG. The total number of bits applied to the primary inputs will be n_{PI}, where n_{PI} is the number of PIs. Similarly, the total number of bits shifted into the scan chain is $n_{SFF} + n_{TP}$, where n_{SFF} is the number of scan flops and n_{TP} is the number of test points present in the circuit. Therefore, the total number of bits per vector will be $(n_{PI} + n_{SFF} + n_{TP})$ and the total number of bits for the complete test set with V vectors will be:

$$\text{TV} = V \times (n_{PI} + n_{SFF} + n_{TP} + 1) \tag{5.2}$$

$$\text{TV Reduction (\%)} = \frac{\text{TV}_{old} - \text{TV}_{new}}{\text{TV}_{old}} \times 100 \tag{5.3}$$

TV_{old} represents TV of the circuit before DFT insertion and TV_{new} represents TV after DFT insertion. The less the TV, the less ATE memory that is required to store the bits. By reducing TV, the amount spent on ATE memory can be reduced.

Test Application Time

Test application time is the time taken to apply all of the vectors to the chip and read out the responses. When all of the scan flops and test points are connected into one long scan chain, the TAT is given by Equation 5.4 [10]. The TAT can be reduced considerably by breaking the long scan chain into smaller scan chains of length N that can be loaded in parallel. In this case, the TAT is given by Equation 5.5. By reducing TAT, the total cost of testing a chip can be reduced, so the TAT reduction in Equation 5.6 is a very important parameter. TAT_{old} represents TAT of the circuit before DFT insertion and TAT_{new} represents TAT after DFT insertion.

$$\text{TAT} = (V+2)(n_{SFF}+n_{TP}) + V + 4 \tag{5.4}$$

$$\text{TAT} = (V+2)\times N + V + 4 \tag{5.5}$$

$$\text{TAT Reduction (\%)} = \frac{\text{TAT}_{old} - \text{TAT}_{new}}{\text{TAT}_{old}} \times 100 \tag{5.6}$$

Hardware Overhead

Inserting scan flops and test points comes at the cost of extra hardware. The percentage of extra hardware added to the circuit by means of DFT is called hardware overhead. Each logic gate has 4 transistors on average and each flip-flop has 18 transistors. Each complete test point has 18.25 transistors and scanning a flip-flop increases its transistor count to 24. The hardware overhead for partial scan with test points is given by Equation 5.7. The hardware overhead for full scan circuits with test points is given by Equation 5.8. The hardware for full scan circuits accounts for flip-flops having scan hardware from the onset, and hence, only test points are considered as overhead. That is why Equation 5.8 has only

n_{TP} in the numerator.

$$O_{PS+TP} = \frac{24 \times n_{SFF} + 18.25 \times n_{TP}}{4 \times n_{gates} + 18 \times n_{FF}} \times 100 \tag{5.7}$$

$$O_{FS+TP} = \frac{18.25 \times n_{TP}}{4 \times n_{gates} + 24 \times n_{FF}} \times 100 \tag{5.8}$$

5.2 Full Scan with Test Point Insertion Results

Many options explained in Appendix A were tried in the full scan mode. The full scan results were generated using the ITC '99 benchmark circuits [21]. The circuit *b*05 was modified to accommodate buses present to suit the tool as the rutmod netlist format, which the algorithm is implemented to parse, does not support buses. The rutmod netlist format does not allow the same name to refer to different fanout branches of the same fanout stem. To accommodate for buses in *b*05, the gate driving the buses was replicated and each line of the bus was given a different name, as shown in Figure 5.1. The modified *b*05 netlist is called as *b*05*s*. The number of test points inserted into the benchmark circuits has been

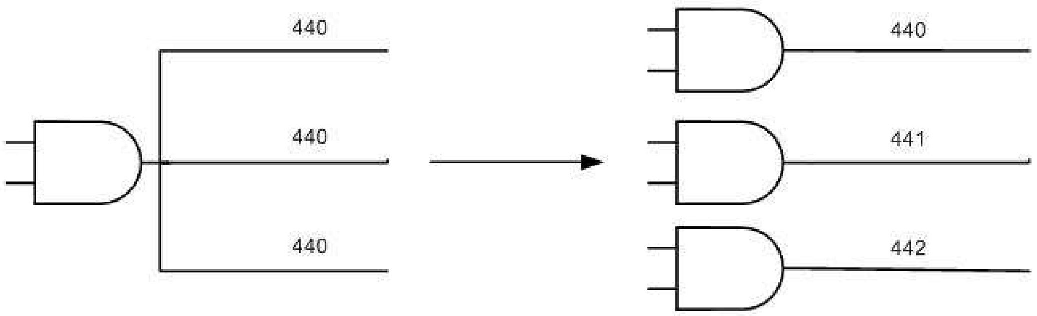

Figure 5.1: Example gate showing modification of b05

restricted such that they incur only 10% of hardware overhead. It is assumed that the scan and test point chains are broken into parallel chains of length 64 to reduce the TAT. Hence, for calculating TAT from Equation 5.5, $N = 64$ will be used. Each circuit was warmed up with a fixed number of warmup vectors and the testability measures were updated with a fixed number of update vectors. The characteristics and parameters of the ITC '99 benchmark circuits that remain the same for all of the experiments are shown in Table 5.1. The TP penalty column in the table indicates the penalty of inserting a TP in the ILP cost function.

Table 5.1: Characteristics and parameters for ITC '99 Benchmark Circuits

Ckt.	PIs	FFs	Gates	Test Points	Warmup Vectors	Update Vectors	TP Penalty
b01	2	5	32	1	51200	512	0.90
b02	1	4	19	1	102400	1024	0.90
b03	4	30	103	3	102400	1024	0.90
b04s	11	66	463	19	100000	1000	0.90
b05s	1	34	783	21	100000	1000	0.95
b06	2	9	36	2	100000	1000	0.85
b07s	1	49	326	13	100000	1000	0.90
b08	9	21	116	5	100000	1000	0.90
b09	1	28	115	6	100000	1000	0.90
b10	11	17	139	5	100000	1000	0.90
b11s	7	31	388	12	100000	1000	0.90
b12	5	121	817	34	51200	512	0.90
b13s	10	53	236	12	100000	1000	0.90
b14s	32	245	4124	123	200000	2000	0.90
b15s	36	449	7844	231	51200	512	0.99
b17s	37	1415	21556	659	200000	2000	0.99
b20s	32	490	8189	245	200000	2000	0.90
b21s	32	490	8544	252	200000	2000	0.90
b22s	32	735	13162	385	200000	2000	0.90

5.2.1 Experimental Conditions and Validation

Full scan results were generated by running the C program on the ITC '99 benchmarks on Sun Ultra 5 machines. Full scan results were generated for various modes of operation of the program, which are listed in Appendix A. The integer linear programming algorithm was run using AMPL on an IBM PC. Tests were generated for the ITC '99 benchmark circuits using the Synopsys TetraMAX ATPG before TP insertion and after TP insertion. The results were compared. The ATPG was run in auto mode to enable dynamic compaction during test generation and the generated vectors were compacted statically using built-in options in TetraMAX that are listed as follows:

- **read netlist file_name**
- **run build_model**
- **run drc**
- **add faults -all**
- **run atpg -auto_compression -ndetects 1**
- **run pattern_compress 20**

The options **run atpg** and **run pattern_compress** were repeated until satisfactory results were obtained, such as 100% fault efficiency or a lower vector length.

5.2.2 Fault Coverage Results

The FC results for various modes of operation of the algorithm are shown in Table 5.2. All of the algorithm options are run in gradient descent mode unless specified otherwise. The second column in Table 5.2 is the FC result for circuit with no TPs inserted. The third column is the result using $p_u(1)$ (unbiased probability) as the testability measure. The fourth column is the result using Entropy without unbiasing as testability measure. The fifth column is the result using Unbiased Entropy as the testability measure and the sixth column uses the same Unbiased Entropy but the algorithm was run in ILP mode. The seventh column is the result using accumulated ones counts from each iteration of the algorithm to calculate the Unbiased Entropy measures. The eighth column is the result using Unbiased Entropy measures but inserting more than one TP in each iteration. The ninth column is the result using Entropy Gain as the testability measure and the tenth column is the result using Entropy Differential as the testability measure. Unbiasing is not used with Entropy Gain mode or Entropy Differential mode because the gain and differential measures are calculated based on entropy increase or decrease, respectively. The sample bias will be removed automatically when these measures are calculated as a difference in entropies of each gate and its fanouts. The eleventh column is the result using Unbiased Entropy together with Overlapped Spectral controllability (Mode 1) to select TP candidates. The twelfth column is the result using Unbiased Entropy together with Non-Overlapped Spectral controllability (Mode 1) to select candidates. The thirteenth column is the result using Unbiased Entropy together with Non-Overlapped Spectral observability (Mode 2) to select candidates. The fourteenth column is the result using Unbiased Entropy together with Non-Overlapped Spectral controllability and observability (Mode 3) to select candidates. The fifteenth column is the result using Non-Overlapped Spectral controllability and observability (Mode 4) to select candidates but no entropy measures are used. The last column in Table 5.2 is the collection of best FCs obtained for each circuit using any of the modes from the algorithm.

It can be observed from these tables that the FC gets better with insertion of TPs and the best FC obtained by inserting TPs is on average 1.64% more than the FC obtained without them. Figures 5.2 and 5.3 compare the FC of Biased Entropy with Spectral-based results. The Biased Entropy measure is chosen to be compared with Spectral measures because it has the highest average FC among all of the Entropy modes. Figures 5.4 and 5.5 compare the FC of various Entropy-based results. From the figures and table presented, it can be observed that to get the highest FC, Biased Entropy should be chosen as the testablity measure.

Table 5.2: Fault Coverage results of ITC '99 Benchmark Circuits

Ckt.	No TP	Proba-bility	Biased Entropy	Unbiased Entropy	Unbiased Entropy ILP mode	Accum-ulated Entropy	Multi TP	Entropy Gain	Entropy Differ-ential	Overlap Spectra Mode 1	Non-overlap Spectra				***Best***
											Mode 1	Mode 2	Mode 3	Mode 4	
b01	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b02	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b03	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b04s	99.27	99.87	99.87	99.87	99.87	99.87	99.87	99.87	99.87	**100.00**	99.87	99.87	99.87	99.87	***100.00***
b05s	80.08	90.94	92.80	90.94	90.98	89.82	91.06	89.16	91.41	83.00	90.94	91.13	91.13	**92.88**	***92.88***
b06	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b07s	99.47	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b08	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b09	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b10	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b11s	97.66	99.10	99.11	99.11	99.11	98.89	98.89	99.18	99.41	99.48	**100.00**	99.11	98.89	99.11	***100.00***
b12	99.96	99.96	99.96	99.99	99.97	99.99	99.97	**100.00**	**100.00**	**100.00**	99.97	99.97	99.97	**100.00**	***100.00***
b13s	96.97	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	99.89	**100.00**	**100.00**	**100.00**	98.75	***100.00***
b14s	97.86	99.94	99.94	**99.95**	**99.95**	**99.95**	99.95	99.53	99.09	99.60	**99.95**	**99.95**	**99.95**	97.89	***99.95***
b15s	97.77	99.60	99.60	99.61	99.68	99.68	99.68	99.71	99.67	**99.80**	99.63	99.53	99.54	99.49	***99.80***
b17s	97.91	99.67	99.67	99.68	99.69	99.69	99.68	99.70	99.66	**99.87**	99.68	99.69	99.70	99.50	***99.87***
b20s	98.08	99.96	99.96	**99.97**	**99.97**	99.93	**99.97**	99.52	99.13	99.90	**99.97**	**99.97**	**99.97**	99.21	***99.97***
b21s	98.13	99.85	99.85	99.86	99.86	99.89	99.86	99.54	99.30	**99.94**	99.85	99.85	99.85	99.32	***99.94***
b22s	98.07	99.84	99.84	99.84	99.85	**99.89**	99.85	99.45	98.55	99.87	99.79	99.84	99.80	99.17	***99.89***
Avg.	97.96	99.41	**99.51**	99.41	99.42	99.35	99.41	99.25	99.27	98.02	99.46	99.42	99.40	99.22	***99.60***

Variability: 99.51% - 1.55% + 0.09%

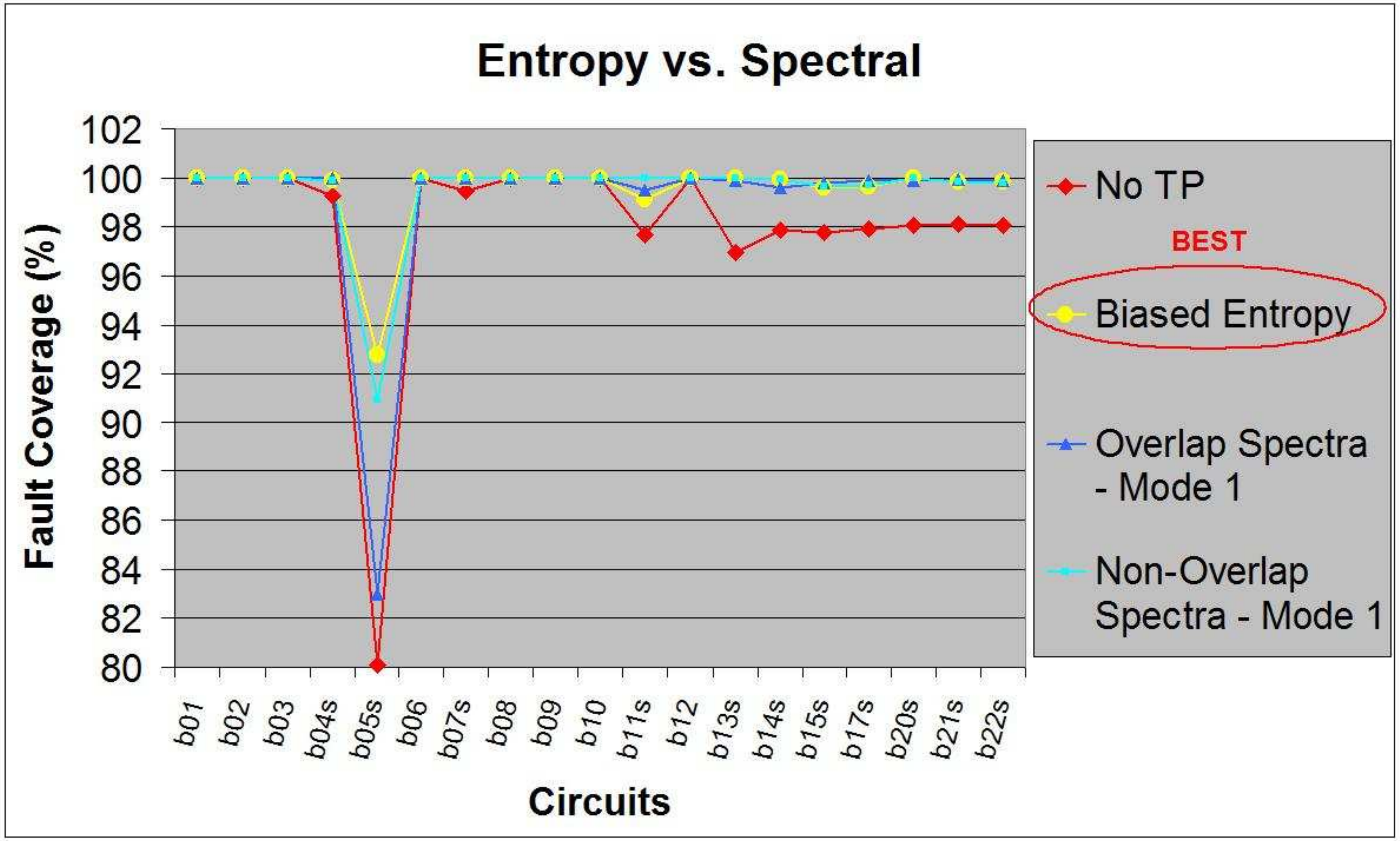

Figure 5.2: Fault Coverage comparison: Entropy vs. Spectral

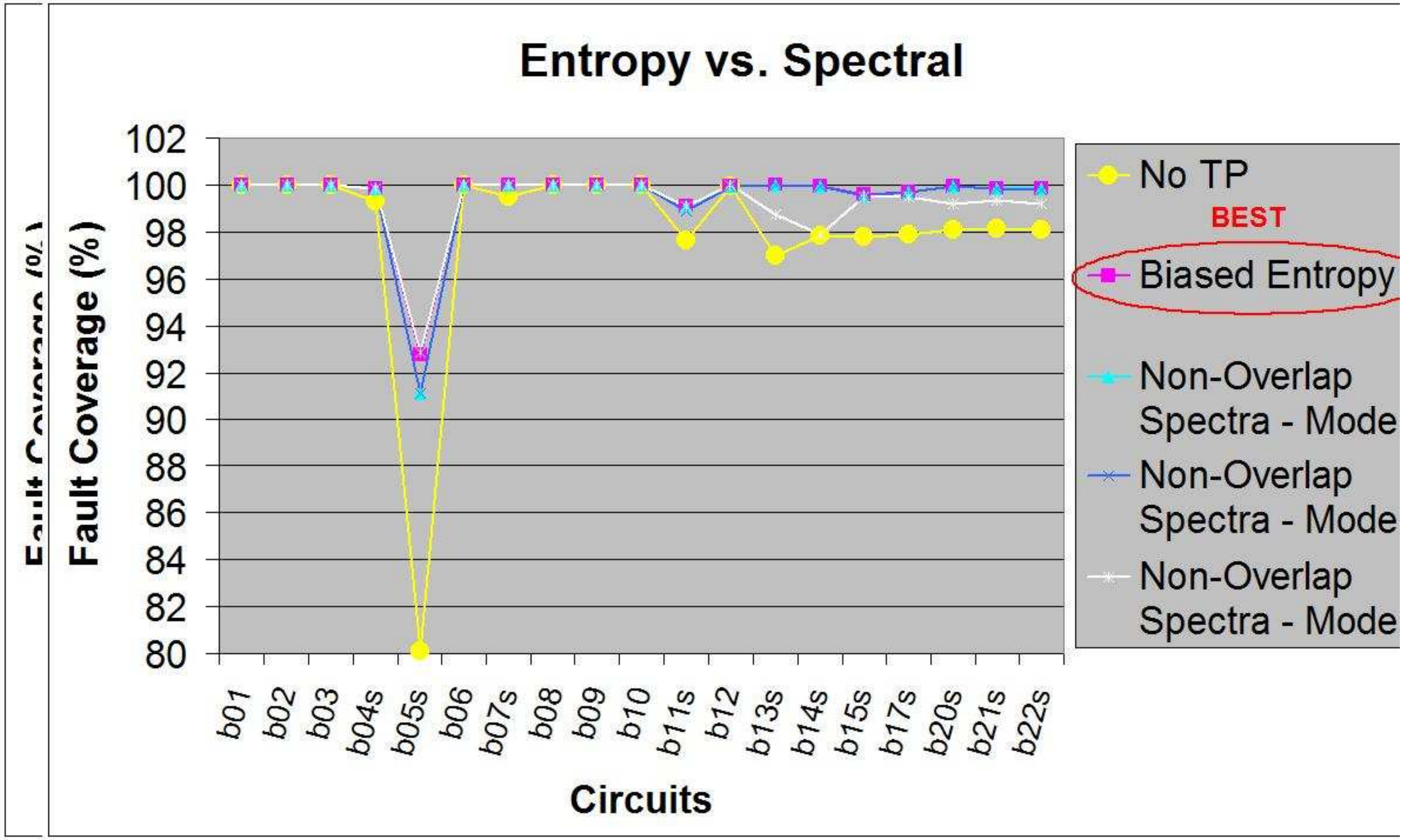

Figure 5.3: Fault Coverage comparison: Entropy vs. Spectral

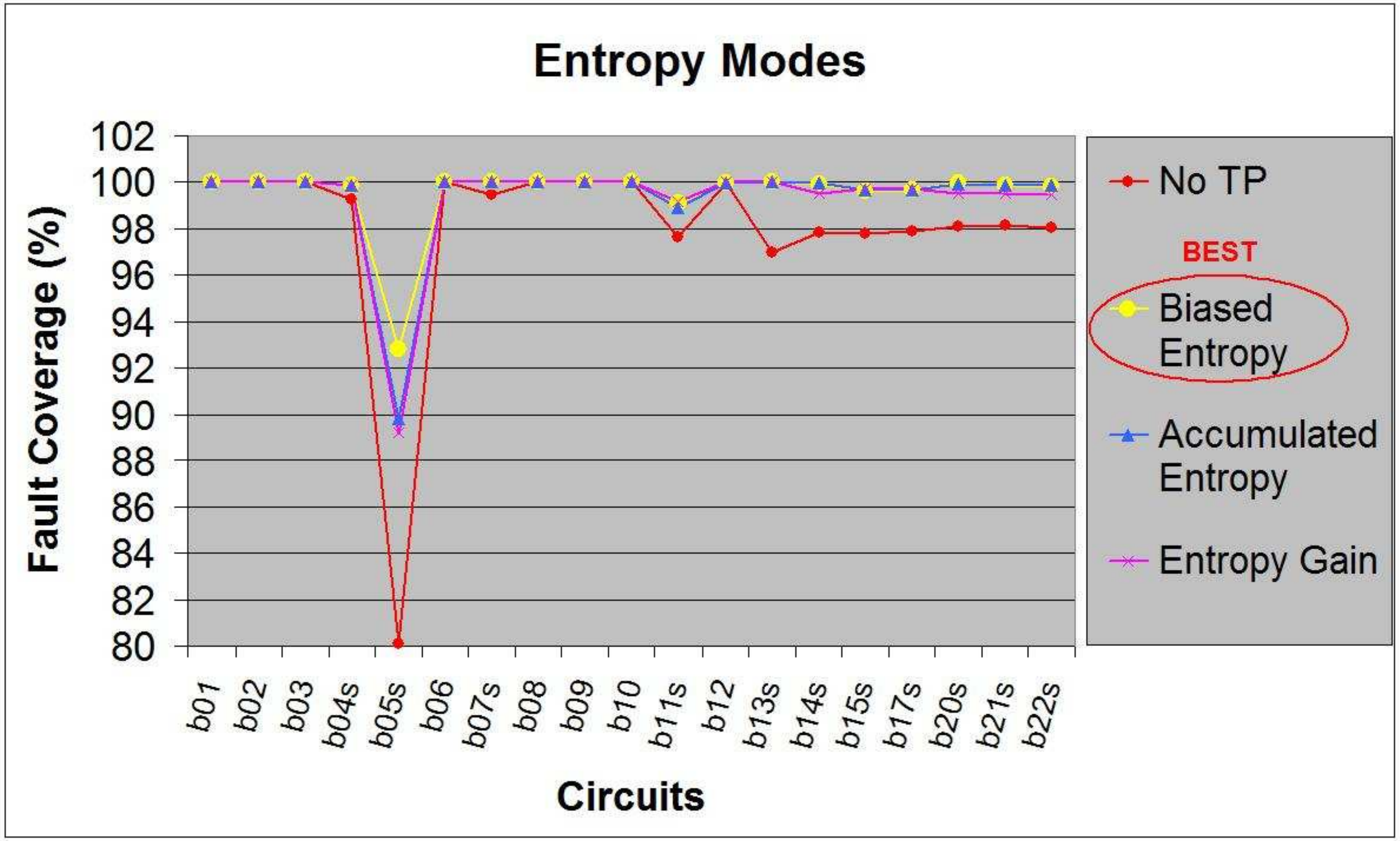

Figure 5.4: Fault Coverage comparison: Entropy Modes

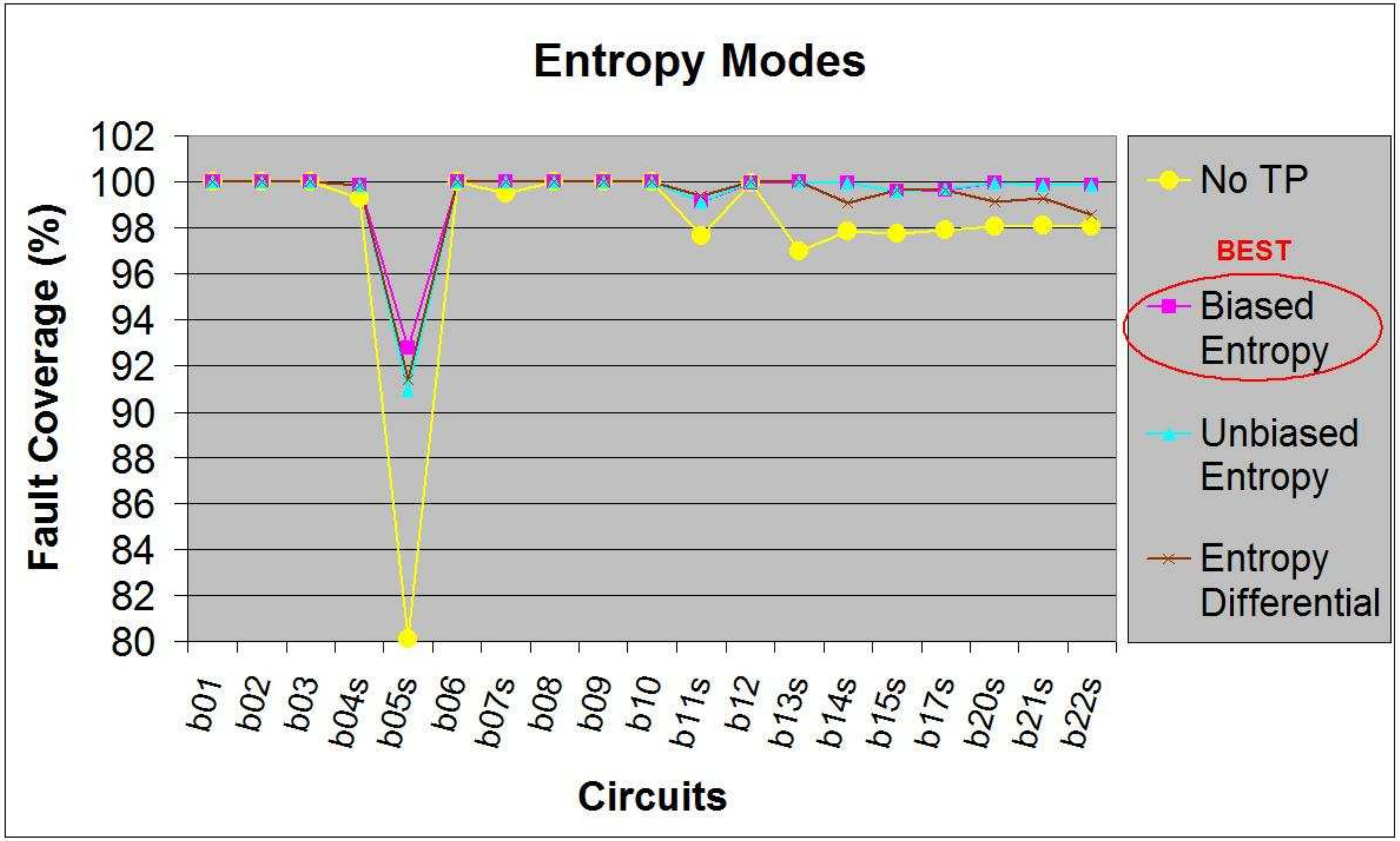

Figure 5.5: Fault Coverage comparison: Entropy Modes

5.2.3 Fault Efficiency Results

The FE results for various modes of operation of the algorithm is shown in Table 5.3. Inserting TPs using the proposed algorithm helps it to achieve 100% FE on of all the circuits, which is 0.21% more than the FE achieved without any TPs inserted into the circuits. Figure 5.6 compares the FE of Entropy Gain with Spectral-based results. The Entropy Gain measure is chosen to be compared with Spectral measures because it has the highest average FE among all of the Entropy modes. Figure 5.7 compares the FE of Overlap Spectra Mode 1 with other spectral modes. Figures 5.8 and 5.9 compare the FE of various Entropy-based results. From Table 5.3 and the figures, it can be observed that the highest FE is acheived by running the algorithm in Overlapped Spectra Mode 1.

Table 5.3: Fault Efficiency results of ITC ’99 Benchmark Circuits

Ckt.	No TP	Proba- -bility	Biased Entropy	Unbiased Entropy	Unbiased Entropy ILP mode	Accumu- -lated Entropy	Multi TP	Entropy Gain	Entropy Differen- -tial	Overlap Spectra Mode 1	Non-overlap Spectra				***Best***
											Mode 1	Mode 2	Mode 3	Mode 4	
b01	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b02	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b03	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b04s	99.60	99.87	99.87	99.87	99.87	99.87	99.87	99.87	99.87	**100.00**	99.87	99.87	99.87	99.87	***100.00***
b05s	99.91	99.24	99.92	99.24	99.28	99.32	99.24	99.61	**100.00**	99.86	99.24	99.32	99.32	**100.00**	***100.00***
b06	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b07s	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b08	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b09	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b10	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b11s	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b12	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b13s	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b14s	99.44	**100.00**	99.63	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	99.88	**100.00**	**100.00**	**100.00**	**100.00**	99.44	***100.00***
b15s	99.12	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	99.99	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	***100.00***
b17s	99.04	**100.00**	99.99	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	99.99	**100.00**	**100.00**	**100.00**	**100.00**	99.99	***100.00***
b20s	99.58	**100.00**	99.67	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	99.85	**100.00**	**100.00**	**100.00**	**100.00**	99.68	***100.00***
b21s	99.60	**100.00**	99.72	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	99.93	**100.00**	**100.00**	**100.00**	**100.00**	99.61	***100.00***
b22s	99.65	**100.00**	99.70	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	99.75	**100.00**	**100.00**	**100.00**	**100.00**	99.69	***100.00***
Avg.	99.79	99.95	99.92	99.95	99.96	99.96	99.95	99.97	99.96	**99.99**	99.95	99.96	99.96	99.91	***100.00***

Variability: 99.99% - 0.02% + 0.01%

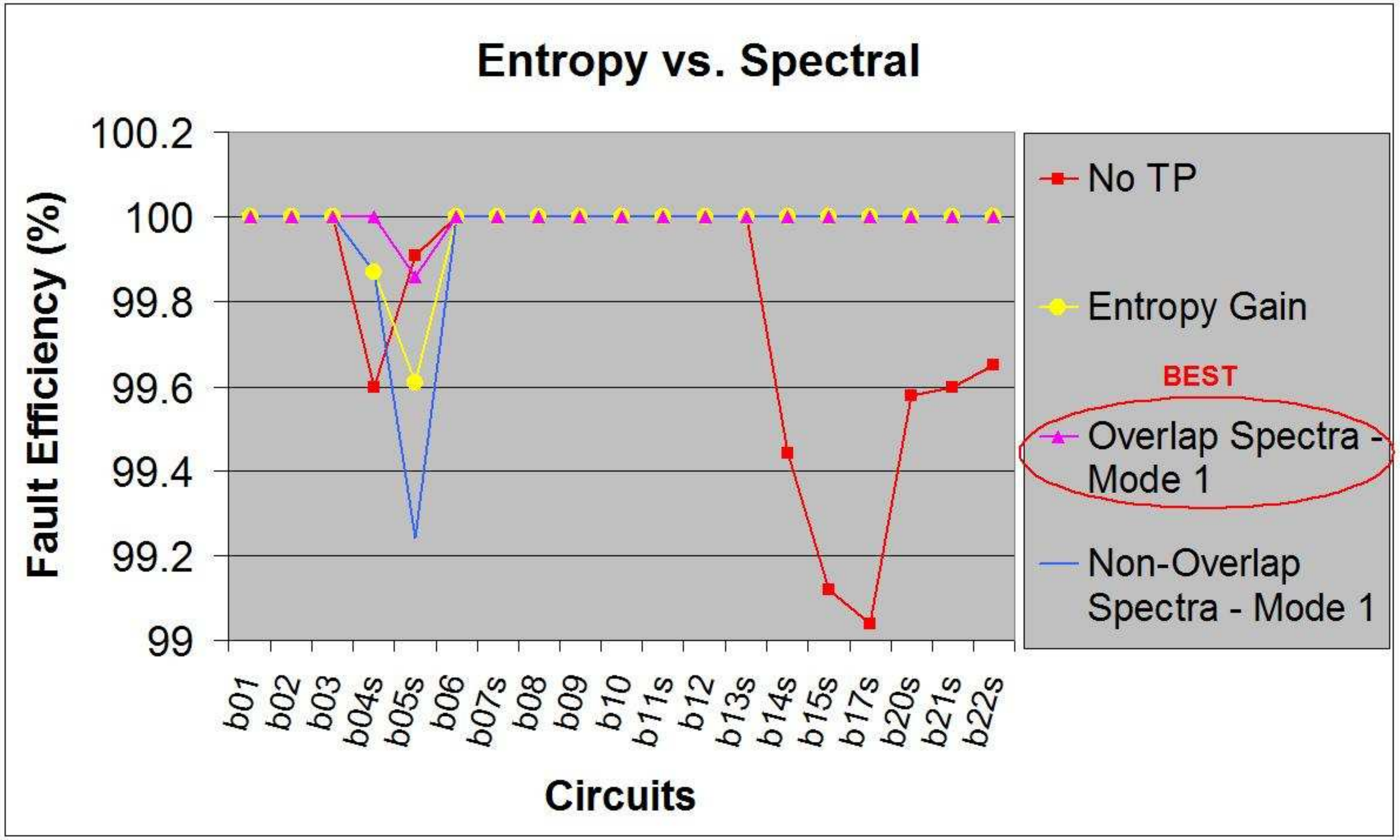

Figure 5.6: Fault Efficiency comparison: Entropy vs. Spectral

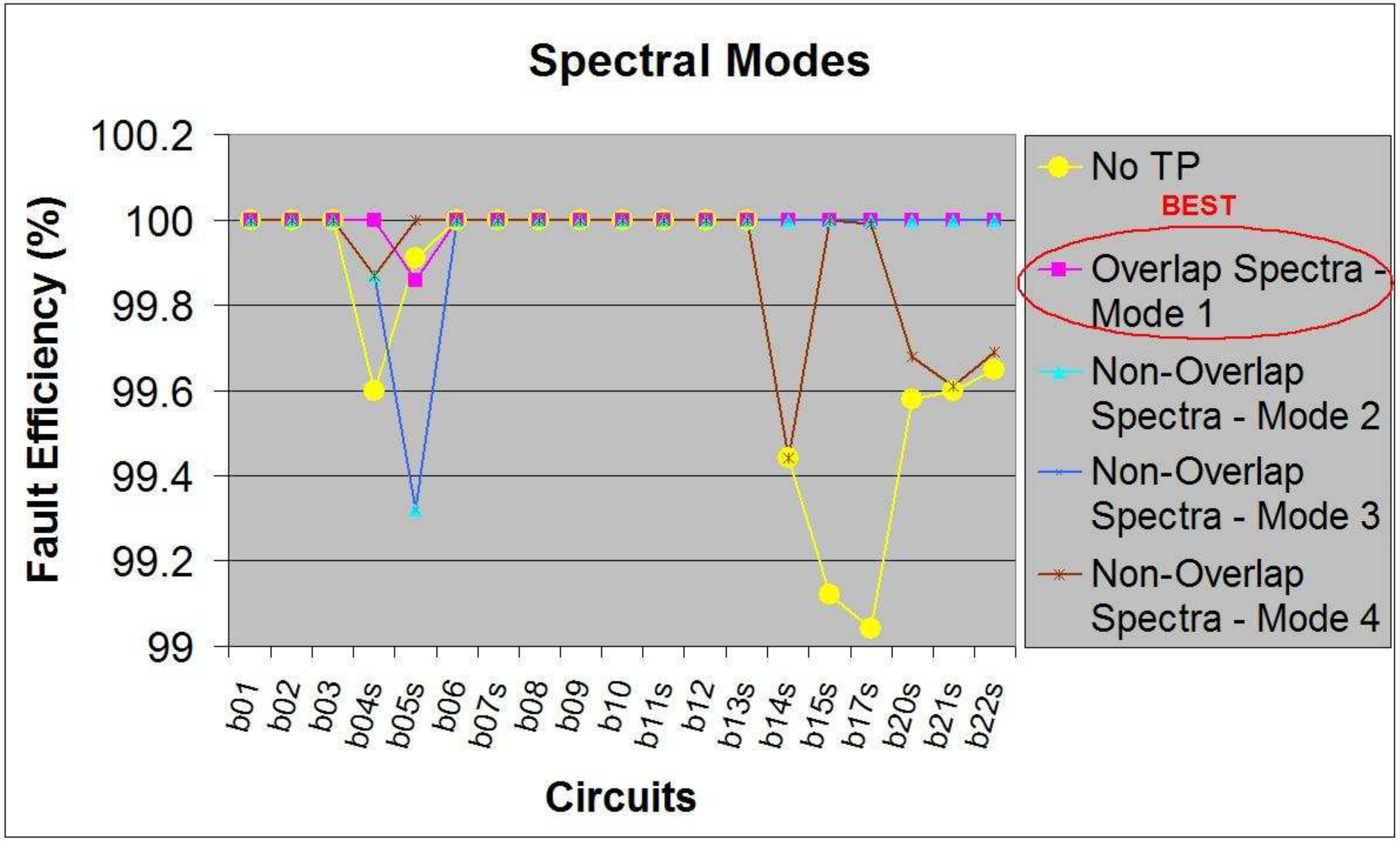

Figure 5.7: Fault Efficiency comparison: Spectral Modes

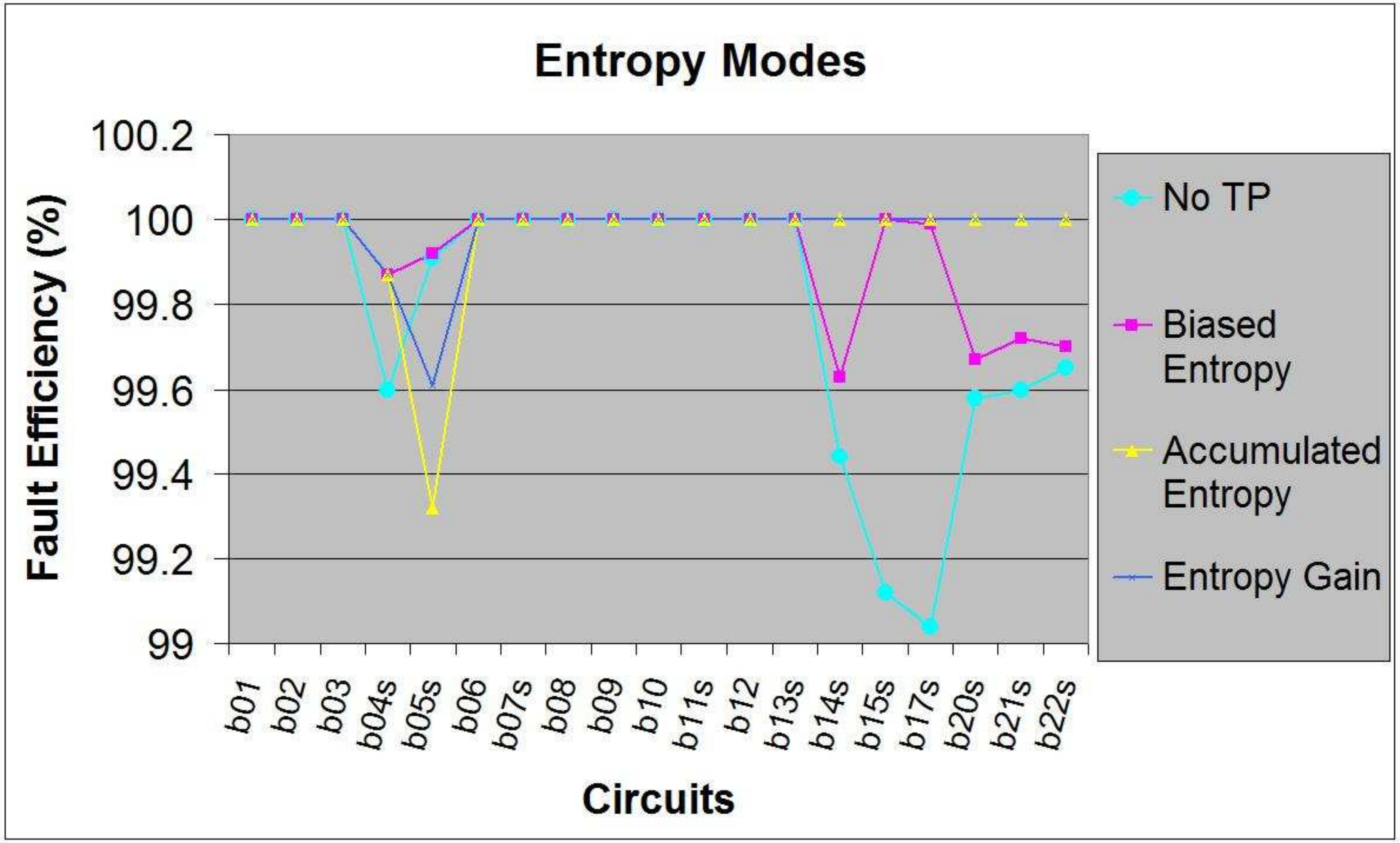

Figure 5.8: Fault Efficiency comparison: Entropy Modes

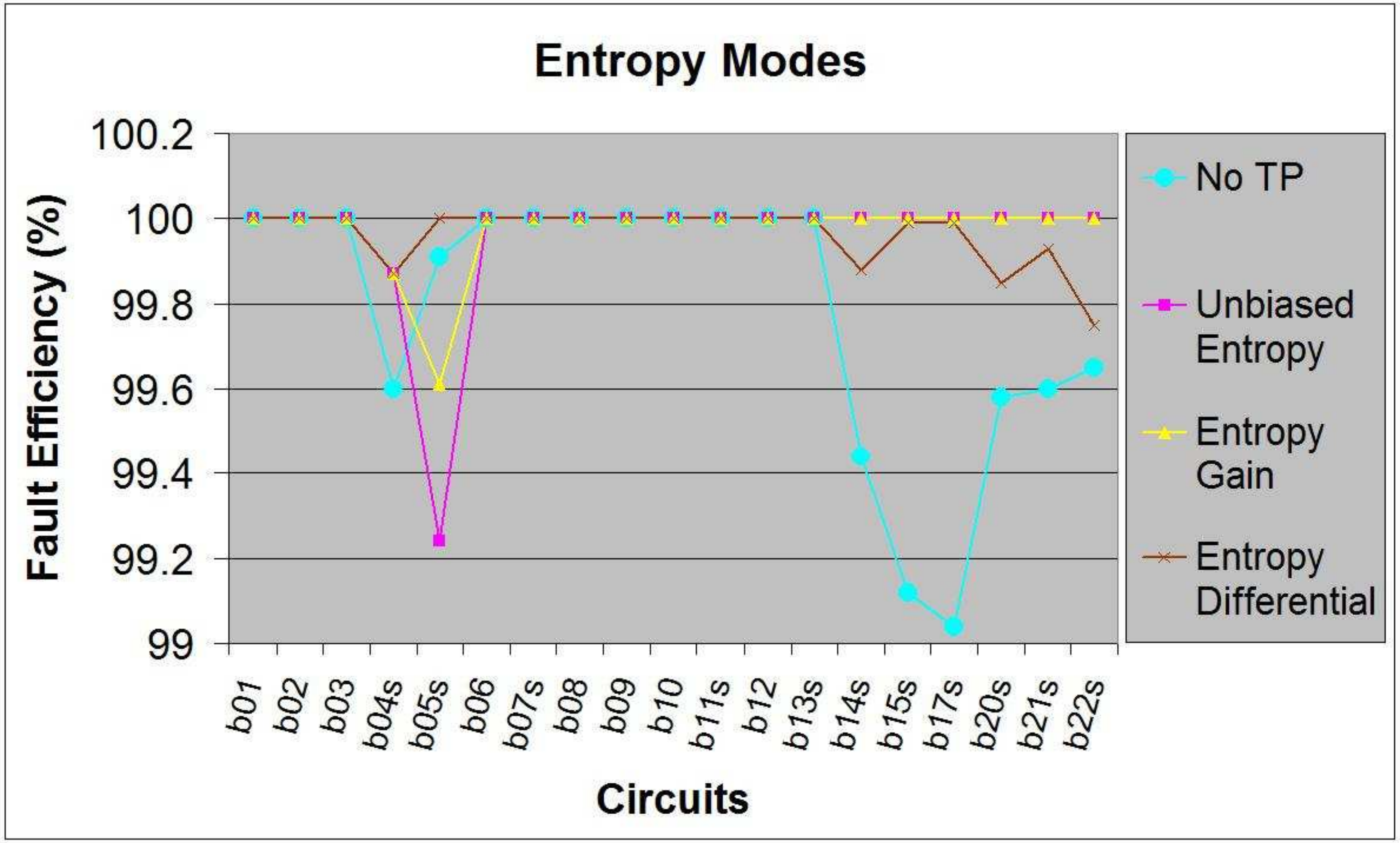

Figure 5.9: Fault Efficiency comparison: Entropy Modes

5.2.4 Test Vector Length Results

The Test Vector Length results for various modes of operation of the algorithm are shown in Table 5.4. The maximum average vector length reduction that can be achieved by using the various modes proposed is 48.68%. The last row in Table 5.4 represents the average V for the largest ITC '99 benchmark circuits of b14s to b22s. The maximum vector length reduction achieved on larger circuits is 59%. Figure 5.10 compares the V of Accumulated Entropy with spectral-based results. The Accumulated Entropy measure is chosen to be compared with Spectral measures because it has the highest average vector length reduction, 48.26% on all circuits and 54.04% on larger circuits, among all of the Entropy modes. Figure 5.11 compares V of various Entropy-based results. From Table 5.4 and Figures 5.10 and 5.11, it can be observed that the shortest vector length can be achieved by using the Accumulated Entropy as the testability measure.

Table 5.4: Test Vector Length results of ITC '99 Benchmark Circuits

Ckt.	No TP	Proba- -bility	Biased Entropy	Unbiased Entropy	Unbiased Entropy ILP mode	Accumu- -lated Entropy	Multi TP	Entropy Gain	Entropy Differen- -tial	Overlap Spectra Mode 1	Non-overlap Spectra				***Best***
											Mode 1	Mode 2	Mode 3	Mode 4	
b01	16	**15**	16	**15**	**15**	**15**	**15**	**15**	17	16	**15**	15	**15**	16	***15***
b02	11	**10**	11	**10**	**10**	**10**	**10**	11	12	**10**	11	**10**	11	11	***10***
b03	24	15	21	15	15	15	**14**	19	20	15	15	15	15	21	***14***
b04s	54	40	45	40	40	43	40	**38**	43	42	40	46	43	45	***38***
b05s	**65**	76	73	75	73	**65**	78	67	67	67	76	72	72	71	***65***
b06	16	**15**	16	**15**	**15**	**15**	**15**	**15**	17	**15**	**15**	**15**	**15**	16	***15***
b07s	44	**29**	34	**29**	**29**	32	34	32	39	39	33	32	31	38	***29***
b08	36	30	35	30	30	31	**28**	35	33	41	30	30	30	37	***28***
b09	29	16	21	16	16	16	18	26	**11**	19	26	16	26	23	***11***
b10	40	**32**	38	**32**	**32**	**32**	**32**	34	39	39	38	**32**	38	35	***32***
b11s	58	37	52	37	37	39	41	55	51	50	**34**	37	39	54	***34***
b12	89	74	82	74	74	79	75	**55**	86	75	74	80	80	82	***55***
b13s	29	26	28	26	26	26	25	24	29	29	**22**	25	23	30	***22***
b14s	410	195	348	196	196	**167**	193	233	315	252	203	193	210	351	***167***
b15s	436	213	356	213	218	237	217	196	253	**195**	229	307	299	383	***195***
b17s	531	258	361	260	260	274	269	228	**215**	225	258	265	257	389	***215***
b20s	419	203	400	203	203	**163**	209	276	391	219	207	192	252	404	***163***
b21s	420	175	365	176	**168**	199	192	276	319	217	216	180	246	385	***168***
b22s	445	213	413	220	219	**175**	217	282	421	266	205	231	279	403	***183***
Avg.	166.95	88.00	142.90	88.53	88.21	**85.95**	90.63	100.89	125.16	96.37	91.95	94.37	104.26	147.05	***85.69***
(b14s -b22s)	443.50	209.50	373.83	211.33	210.67	**202.50**	216.17	248.50	319.00	229.00	219.67	228.00	257.17	385.83	***181.83***

Variability: 202.50 - 28.67 + 241

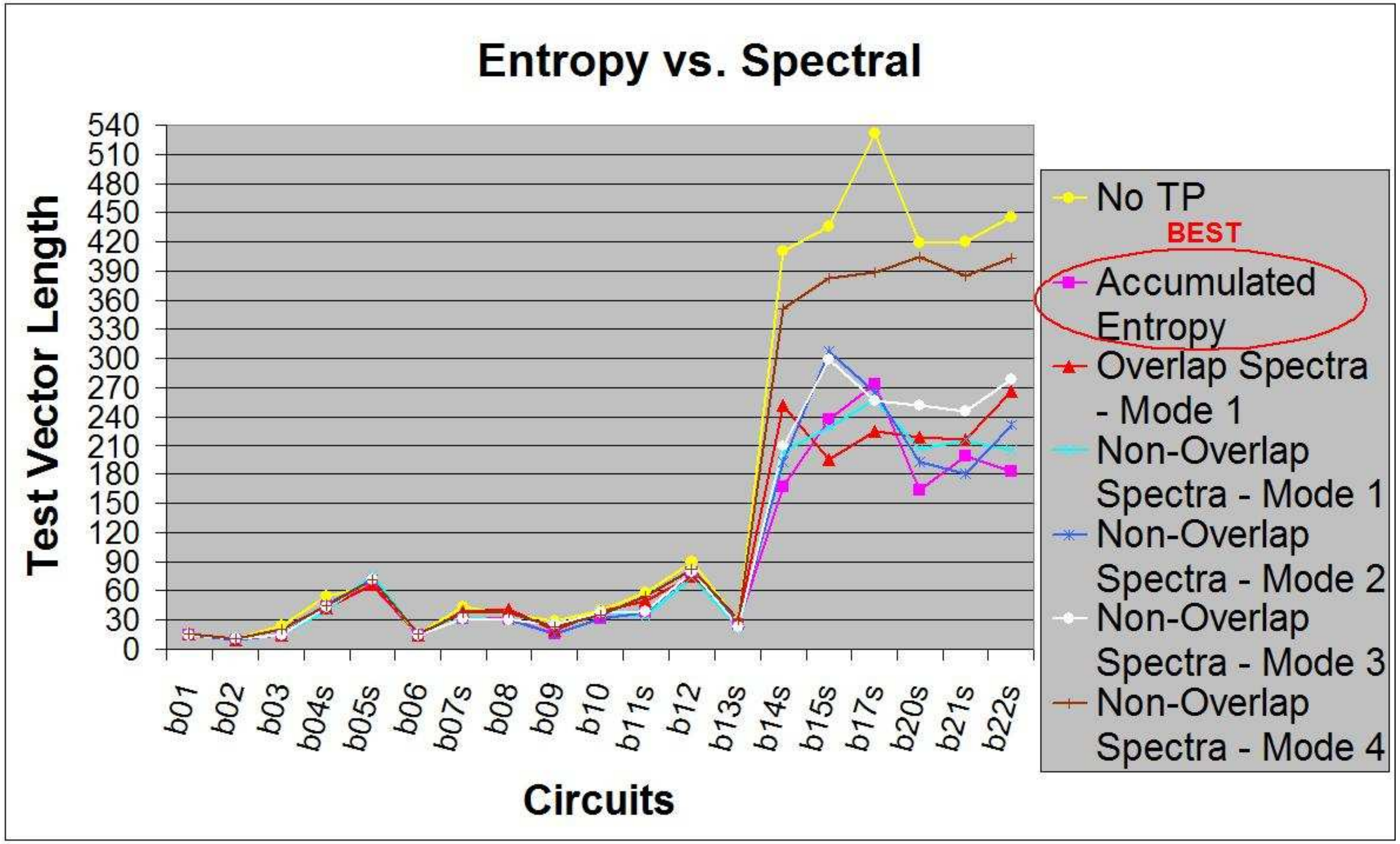

Figure 5.10: Test Vector Length comparison: Entropy vs. Spectral

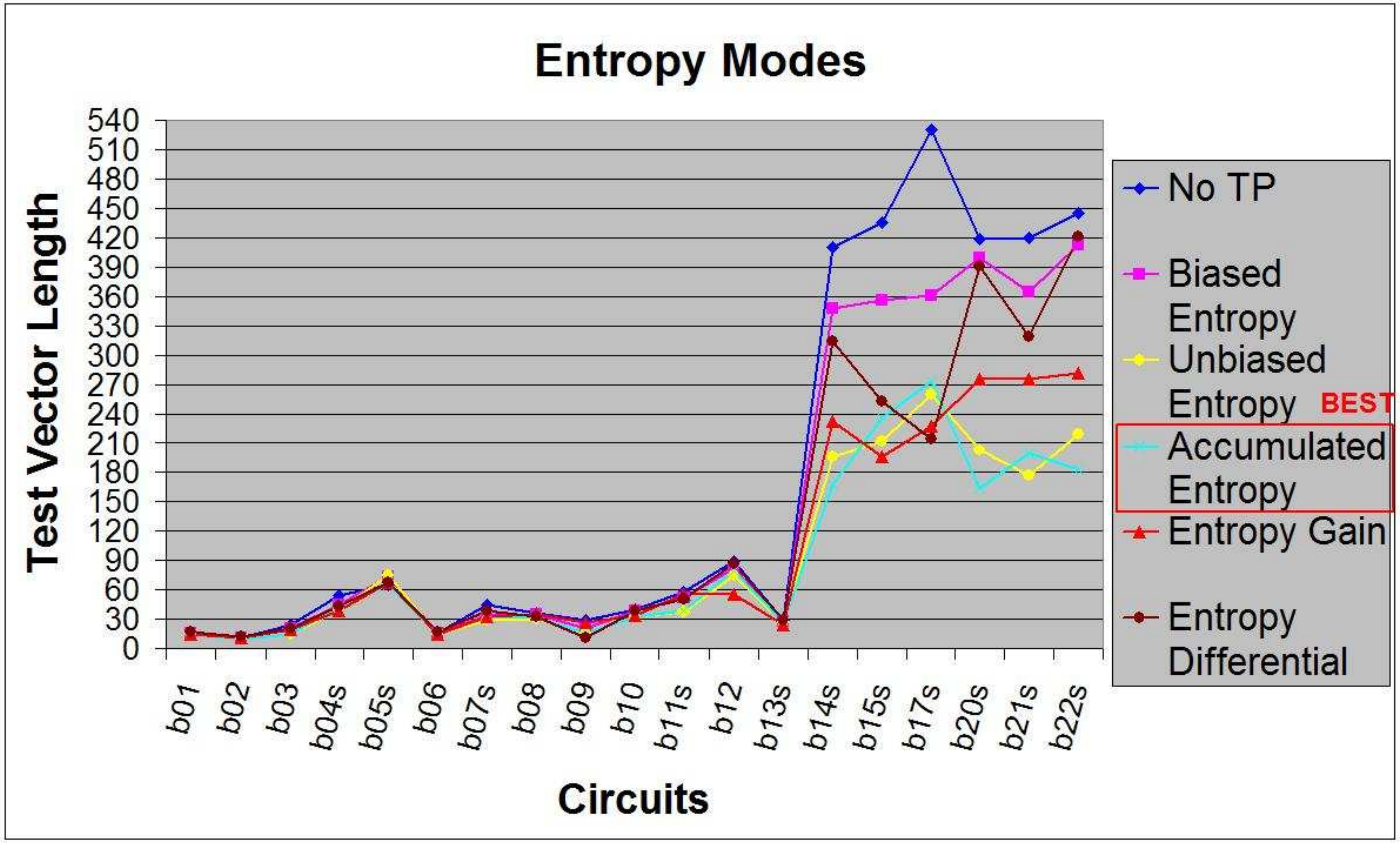

Figure 5.11: Test Vector Length comparison: Entropy Modes

5.2.5 Test Volume Results

The Test Volume results for various modes of operation of the algorithm are shown in Table 5.5. The TV is calculated using Equation 5.2. As TV depends on V and number of TPs inserted in the circuit, some circuits could have a better TV without any TPs in them, such as b01, b02, b05s and b06, as shown in the table. These circuits have a very low vector length to begin with and inserting TPs does not help in reducing the vector length further. The maximum average TV reduction, calculated using Equation 5.3, that can be achieved by using the various modes proposed is 40.05%. The last row in Table 5.5 represents the average TV for the largest ITC '99 benchmark circuits of b14s to b22s. The maximum TV reduction achieved on larger circuits is 40.47%. Figure 5.12 compares the TV of the larger ITC '99 circuits for Accumulated Entropy with spectral-based results, as the smaller circuits have very low TV compared to the larger circuits and do not show up well in the charts. The Accumulated Entropy measure is chosen to be compared with Spectral measures because it has the highest average TV reduction, 30.39% on all circuits and 30.95% on larger circuits, among all of the Entropy modes. Figure 5.13 compares the TV of various Entropy-based results for the larger ITC '99 circuits. From Table 5.5 and the Figures 5.12 and 5.13, it can be observed that the highest TV reduction can be achieved by using the Accumulated Entropy as the testability measure.

Table 5.5: Test Volume results of ITC '99 Benchmark Circuits

Ckt.	No TP	Proba--bility	Biased Entropy	Unbiased Entropy	Unbiased Entropy ILP mode	Accum--ulated Entropy	Multi TP	Entropy Gain	Entropy Differ--ential	Overlap Spectra Mode 1	Non-overlap Spectra				***Best***
											Mode 1	Mode 2	Mode 3	Mode 4	
b01	**128**	135	144	135	135	135	135	135	153	144	135	135	135	144	135
b02	**66**	70	77	70	70	70	70	77	84	70	77	70	77	77	70
b03	840	570	798	570	570	570	**532**	722	760	570	570	570	570	798	***532***
b04s	4212	3880	4365	3880	3880	4171	3880	**3686**	4171	4074	3880	4462	4171	4365	***3686***
b05s	**2340**	4332	4161	4275	4161	3705	4446	3819	3819	3819	4332	4104	4104	4047	3705
b06	**192**	210	224	210	210	210	210	210	238	210	210	210	210	224	210
b07s	2244	**1856**	2176	**1856**	**1856**	2048	2176	2048	2496	2496	2112	2048	1984	2432	***1856***
b08	1116	1080	1260	1080	1080	1116	**1008**	1260	1188	1476	1080	1080	1080	1332	***1008***
b09	870	576	756	576	576	576	648	936	**396**	684	936	576	936	828	***396***
b10	1160	**1088**	1292	**1088**	**1088**	**1088**	**1088**	1156	1326	1326	1292	**1088**	1292	1190	***1088***
b11s	2262	1887	2652	1887	1887	1989	2091	2805	2601	2550	**1734**	1887	1989	2754	***1734***
b12	11303	11914	13202	11914	11914	12719	12075	**8855**	13846	12075	11914	12880	12880	13202	***8855***
b13s	1856	1976	2128	1976	1976	1976	1900	1824	2204	2204	**1672**	1900	1748	2280	***1672***
b14s	113980	78195	139548	78596	78596	**66967**	77393	93433	126315	101052	81403	77393	84210	140751	***66967***
b15s	211896	152721	255252	152721	156306	169929	155589	140532	181401	**139815**	164193	220119	214383	274611	***139815***
b17s	771543	544896	762432	549120	549120	578688	568128	481536	**454080**	475200	544896	559680	542784	821568	***454080***
b20s	219137	155904	307200	155904	155904	**125184**	160512	211968	300288	168192	158976	147456	193536	310272	***125184***
b21s	219660	135625	282875	136400	**130200**	154225	148800	213900	247225	168175	167400	139500	190650	298375	***130200***
b22s	341760	245589	476189	253660	252507	**201775**	250201	325146	485413	306698	236365	266343	321687	464659	***201775***
Avg.	100345	70658	118775	71364	71159	**69849**	73204	78634	96210	73201	72798	75868	83075	123363	***60156***
(b14s -b22s)	312996	218821	370582	221066	220438	**216128**	226770	244419	299120	226522	225538	235081	257875	385039	***186336***

Variability: 216128 -29732 + 168911

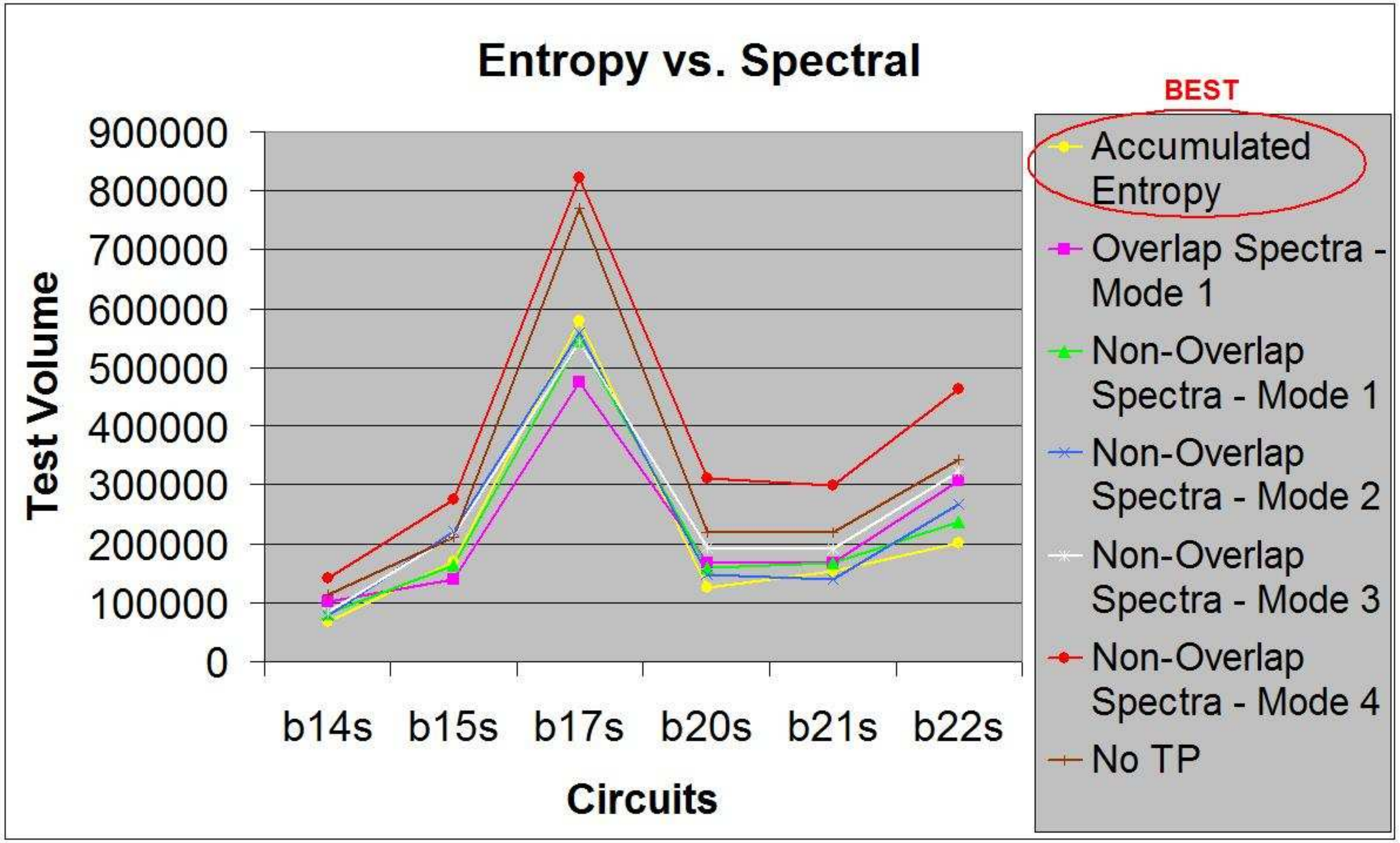

Figure 5.12: Test Volume comparison: Entropy vs. Spectral

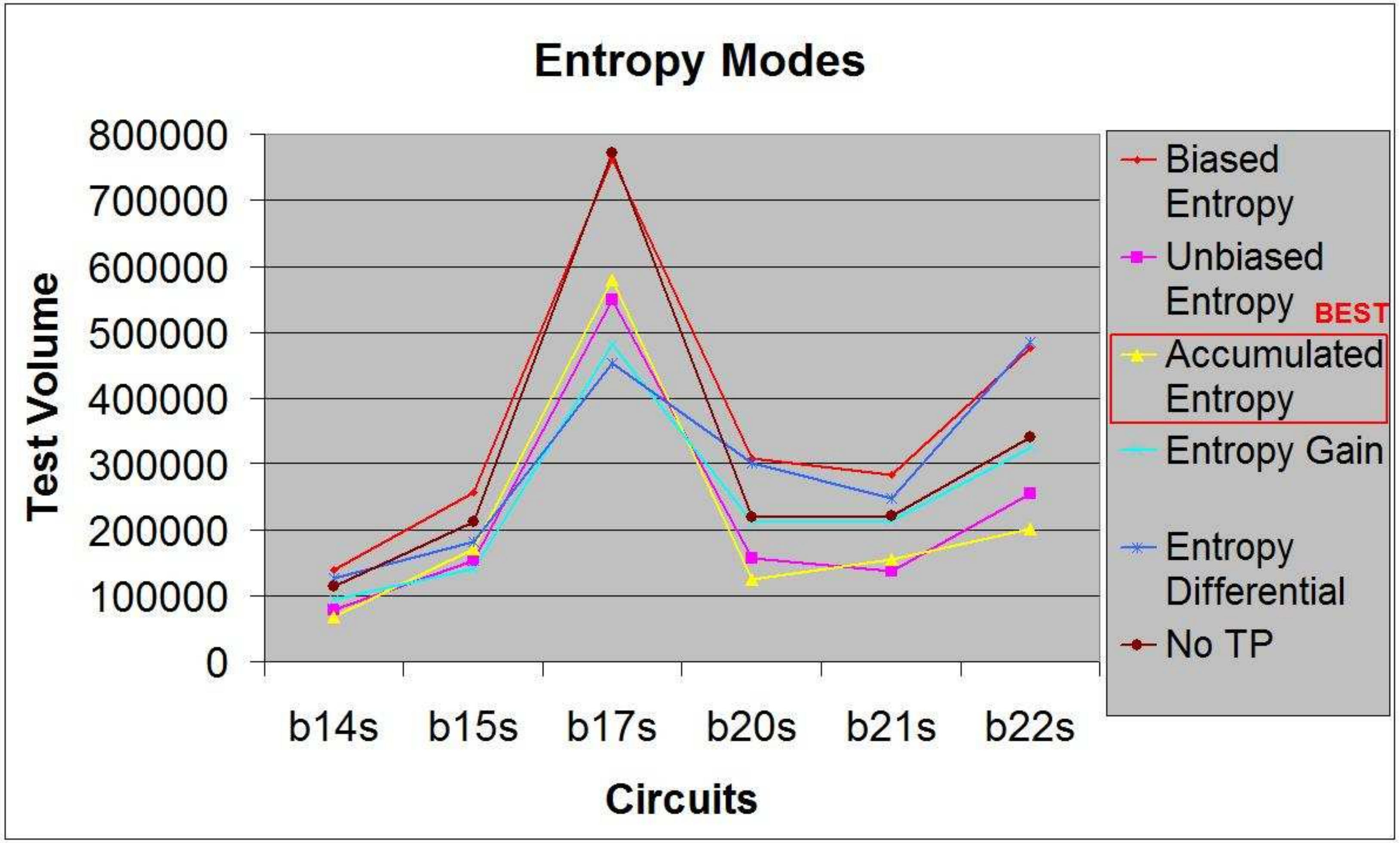

Figure 5.13: Test Volume comparison: Entropy Modes

5.2.6 Test Application Time Results

The Test Application Time results for various modes of operation of the algorithm are shown in Table 5.6. The TAT is calculated using Equation 5.4. Both TAT and TV depend on V and the number of TPs inserted in the circuit, so some circuits could have a better TAT without any TPs in them, such as b01, b02, b05s and b06, as shown in the table. These circuits have a very low vector length to begin with and inserting TPs does not help in reducing the vector length further. The maximum average TAT reduction, calculated using Equation 5.6, that can be achieved by using the various modes proposed is 54.24%. The last row in Table 5.6 represents the average TAT for the largest ITC '99 benchmark circuits of b14s to b22s. The maximum TAT reduction achieved on larger circuits is 58.77%. Figure 5.14 compares the TAT of the Accumulated Entropy mode with spectral-based results. The Accumulated Entropy measure is chosen to be compared with Spectral measures because it has the highest average TAT reduction, 48.38% on all circuits and 54.09% on larger circuits, among all of the Entropy modes. Figure 5.15 compares the TAT of various Entropy-based results for the larger ITC '99 circuits. From Table 5.6 and the Figures 5.14 and 5.15, it can be observed that the highest TAT reduction can be achieved by using the Accumulated Entropy as the testability measure.

Table 5.6: Test Application Time results of ITC '99 Benchmark Circuits

Ckt.	No TP	Proba--bility	Biased Entropy	Unbiased Entropy	Unbiased Entropy ILP mode	Accum--ulated Entropy	Multi TP	Entropy Gain	Entropy Differ--ential	Overlap Spectra Mode 1	Non-overlap Spectra				***Best***
											Mode 1	Mode 2	Mode 3	Mode 4	
b01	**110**	121	128	121	121	121	121	121	135	128	121	121	121	128	121
b02	**67**	74	80	74	74	74	74	80	86	74	80	74	80	80	74
b03	808	580	784	580	580	580	**546**	716	750	580	580	580	580	784	***546***
b04s	3642	2732	3057	2732	2732	2927	2732	**2602**	2927	2862	2732	3122	2927	3057	***2602***
b05s	**2347**	4370	4202	4314	4202	3754	4482	3866	3866	3866	4370	4146	4146	4090	3754
b06	**182**	206	218	206	206	206	206	206	230	206	206	206	206	218	206
b07s	2302	**1955**	2270	**1955**	**1955**	2144	2270	2144	2585	2585	2207	2144	2081	2522	***1955***
b08	838	866	1001	866	866	893	**812**	1001	947	1163	866	866	866	1055	***812***
b09	901	632	807	632	632	632	702	982	**457**	737	982	632	982	877	***457***
b10	758	**784**	922	**784**	**784**	**784**	**784**	830	945	945	922	**784**	922	853	***784***
b11s	1922	1718	2378	1718	1718	1806	1894	2510	2334	2290	**1586**	1718	1806	2466	***1586***
b12	5917	4942	5462	4942	4942	5267	5007	**3707**	5722	5007	4942	5332	5332	5462	***3707***
b13s	1676	1822	1952	1822	1822	1822	1757	1692	2017	2017	**1562**	1757	1627	2082	***1562***
b14s	26782	12807	22752	12872	12872	**10987**	12677	15277	20607	16512	13327	12677	13782	22947	***10987***
b15s	28472	13977	23272	13977	14302	15537	14237	12872	16577	**12807**	15017	20087	19567	25027	***12807***
b17s	34647	16902	23597	17032	17032	17942	17617	14952	**14107**	14757	16902	17357	16837	25417	***14107***
b20s	27367	13327	26132	13327	13327	**10727**	13717	18072	25547	14367	13587	12612	16512	26392	***10727***
b21s	27432	11507	23857	11572	**11052**	13067	12612	18072	20867	14237	14172	11832	16122	25157	***11052***
b22s	29057	13977	26977	14432	14367	**11507**	14237	18462	27497	17422	13457	15147	18267	26327	***11507***
Avg.	10275	5436	8939	5471	5451	**5304**	5604	6219	7800	5924	5664	5852	6461	9207	***4702***
(b14s -b22s)	28959	13749	24431	13868	13825	**13294**	14182	16284	20867	15017	14410	14952	16847	25211	***11939***

Variability: 13294 - 1355 + 15665

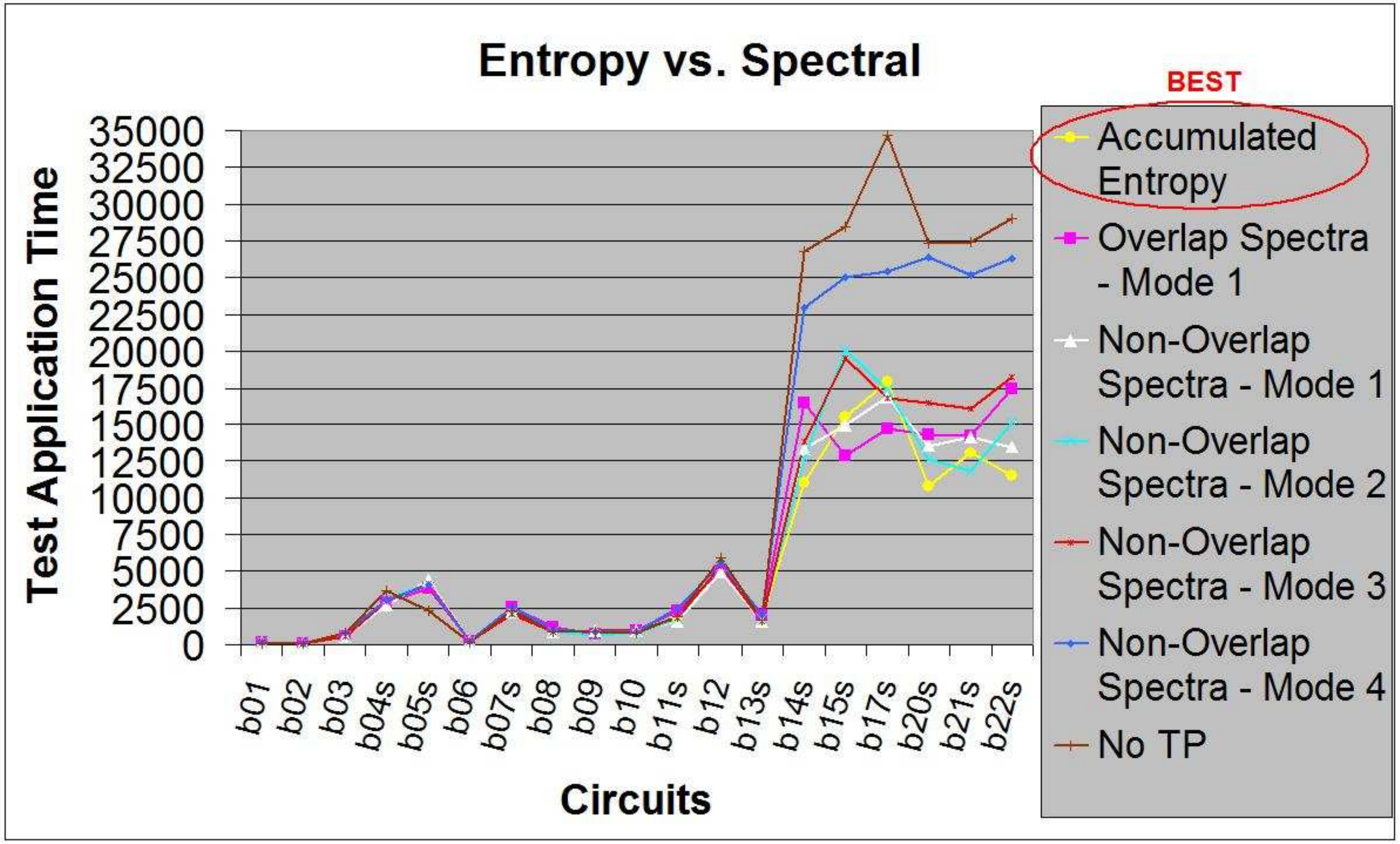

Figure 5.14: Test Application Time comparison: Entropy vs. Spectral

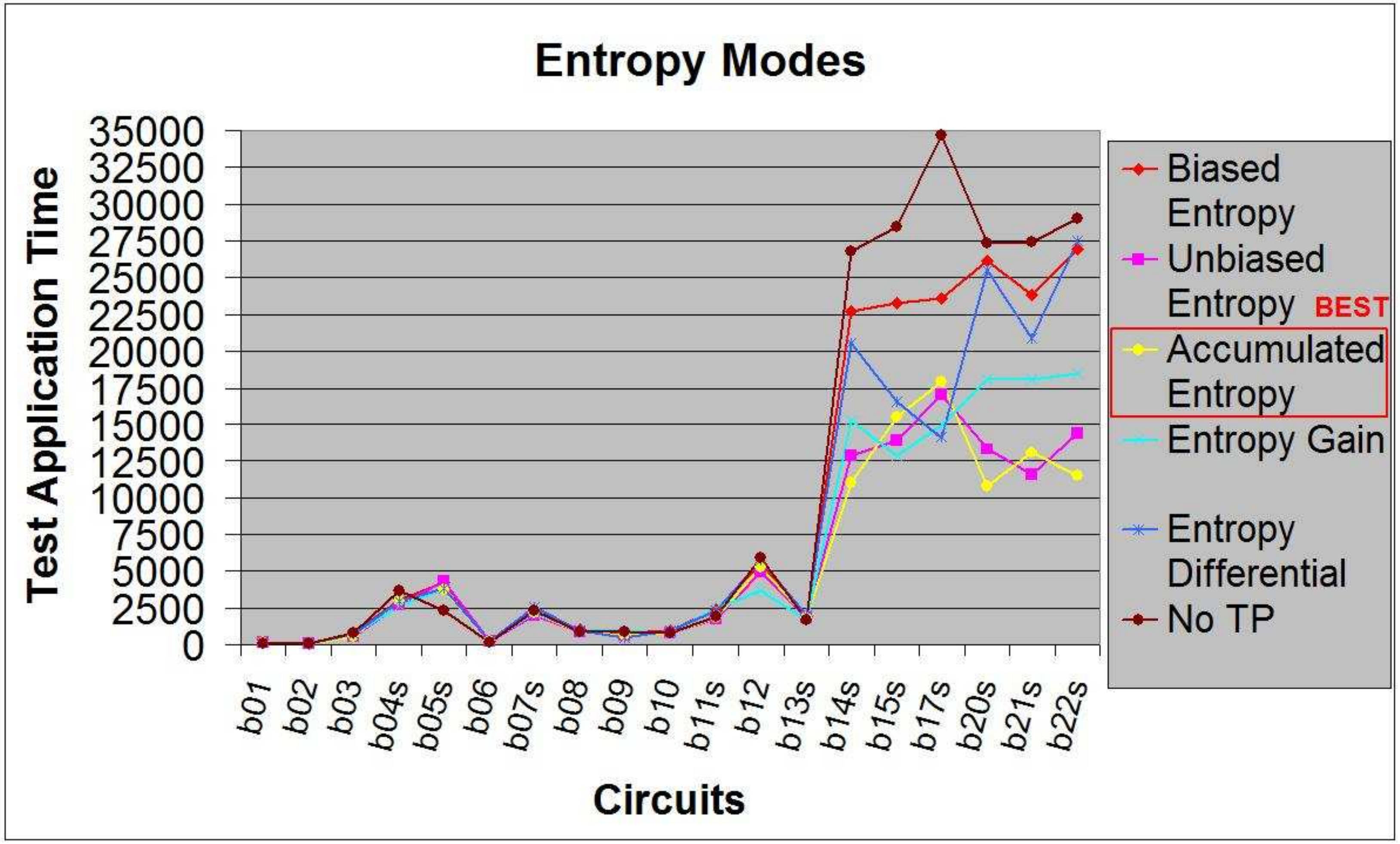

Figure 5.15: Test Application Time comparison: Entropy Modes

5.2.7 ATPG Time Results

The ATPG Time is the time taken by the TetraMAX ATPG to test and generate test vectors for the ITC '99 benchmark circuits. The ATPG time results for various modes of operation of the algorithm are shown in Table 5.7. The maximum average ATPG time reduction that can be achieved by using the various modes proposed is 90.24%. The last row in Table 5.7 represents the average ATPG time for the largest ITC '99 benchmark circuits of b14s to b22s. The maximum ATPG time reduction achieved on larger circuits is 90.32%. Figure 5.16 compares the ATPG of the Accumulated Entropy mode with spectral-based results for the circuits b05s and b14s to b22s. The ATPG times for other circuits are insignificantly low, and hence, are not included in the chart comparison. The Unbiased Entropy measure in ILP mode is chosen to be compared with Spectral measures because it has the highest average ATPG time reduction, 88.82% on all circuits and 89.35% on larger circuits, among all of the Entropy modes. Figure 5.17 compares the ATPG time of various Entropy-based results for b05s and b14s to b22s. From Table 5.7 and the Figures 5.16 and 5.17, it can be observed that the highest ATPG time reduction can be achieved by using the Unbiased Entropy as the testability measure in ILP mode.

Table 5.7: ATPG Time results (in sec) for ITC ’99 Benchmark Circuits

Ckt.	No TP	Proba- -bility	Biased Entropy	Unbiased Entropy	Unbiased Entropy ILP mode	Accum- -ulated Entropy	Multi TP	Entropy Gain	Entropy Differ- -ential	Overlap Spectra Mode 1	Non-overlap Spectra				***Best***
											Mode 1	Mode 2	Mode 3	Mode 4	
b01	0.07	0.09	0.07	0.08	0.07	0.07	0.08	0.09	0.07	**0.06**	0.10	0.07	0.07	0.08	***0.06***
b02	0.06	0.06	0.06	0.07	0.08	0.07	0.06	0.06	0.08	**0.05**	0.06	0.06	0.06	**0.05**	***0.05***
b03	0.12	0.09	0.10	0.10	0.09	0.10	**0.08**	0.11	0.15	0.10	0.09	**0.08**	0.11	0.12	***0.08***
b04s	0.83	0.45	0.50	0.44	0.43	0.50	**0.41**	0.50	0.45	0.42	0.52	0.45	0.53	0.57	***0.41***
b05s	2.15	4.97	**2.11**	7.43	5.27	7.11	4.74	5.36	1.75	6.96	11.84	4.37	6.39	2.19	***2.11***
b06	0.08	**0.06**	**0.06**	0.07	**0.06**	0.07	0.07	0.10	0.10	0.08	**0.06**	**0.06**	0.08	0.07	***0.06***
b07s	0.34	**0.25**	0.29	0.28	0.26	0.31	0.28	0.29	0.31	0.31	0.28	0.30	0.30	0.31	***0.25***
b08	0.18	0.16	0.16	0.15	0.15	0.15	0.17	**0.13**	0.15	0.16	0.15	0.14	0.17	0.17	***0.13***
b09	0.13	**0.10**	0.13	**0.10**	**0.10**	0.11	**0.10**	0.12	0.11	0.11	0.12	0.11	0.14	0.12	***0.10***
b10	**0.15**	0.16	0.16	0.17	0.16	0.17	**0.15**	0.17	0.20	0.16	**0.15**	0.16	0.17	0.18	***0.15***
b11s	0.61	0.39	0.52	0.39	0.39	0.44	**0.38**	0.50	0.45	0.48	0.40	**0.38**	0.47	0.60	***0.38***
b12	0.93	0.88	0.83	0.86	0.81	0.91	0.80	**0.77**	0.87	0.88	0.87	0.88	0.93	0.98	***0.77***
b13s	0.18	**0.17**	0.19	0.18	**0.17**	0.19	0.18	0.18	0.25	0.21	**0.17**	0.18	0.18	0.21	***0.17***
b14s	37.70	9.17	35.12	18.69	8.92	**8.78**	8.92	11.24	18.13	11.43	9.95	9.82	11.08	59.88	***8.78***
b15s	120.70	20.32	34.71	19.33	19.09	22.06	20.20	21.30	25.82	**15.86**	20.85	31.93	28.04	44.49	***15.86***
b17s	890.77	60.79	100.61	61.97	61.88	60.54	63.24	66.07	62.73	**53.78**	70.99	70.17	70.18	122.83	***53.78***
b20s	81.54	16.61	69.47	**15.94**	15.95	16.06	16.47	18.11	38.82	19.11	17.94	16.22	22.70	118.98	***15.94***
b21s	93.37	15.48	60.51	14.57	**14.43**	20.59	15.46	18.21	35.57	17.24	18.11	16.27	23.37	112.57	***14.43***
b22s	148.51	30.60	125.22	31.03	25.90	**24.07**	27.22	38.13	98.83	28.79	37.47	36.77	43.69	137.58	***24.07***
Avg.	72.55	8.46	22.67	9.05	**8.11**	8.54	8.37	9.55	14.99	8.22	10.01	9.92	10.98	31.68	***7.24***
(b14s -b22s)	228.77	25.50	70.94	26.92	**24.36**	25.35	25.25	28.84	46.65	24.37	29.22	30.20	33.18	99.39	***22.14***

Variability: 24.36 - 2.22 + 204.41

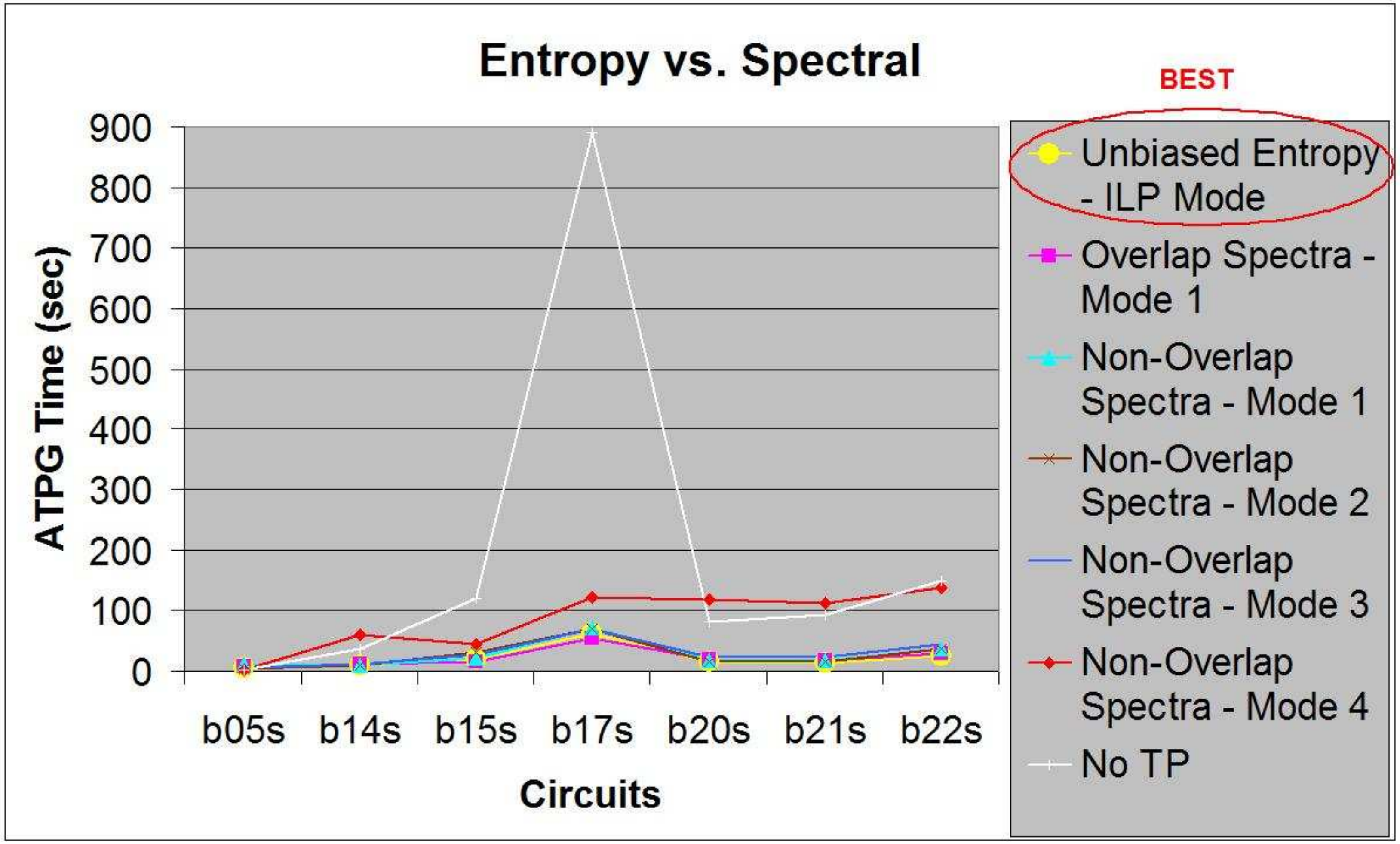

Figure 5.16: ATPG Time comparison: Entropy vs. Spectral

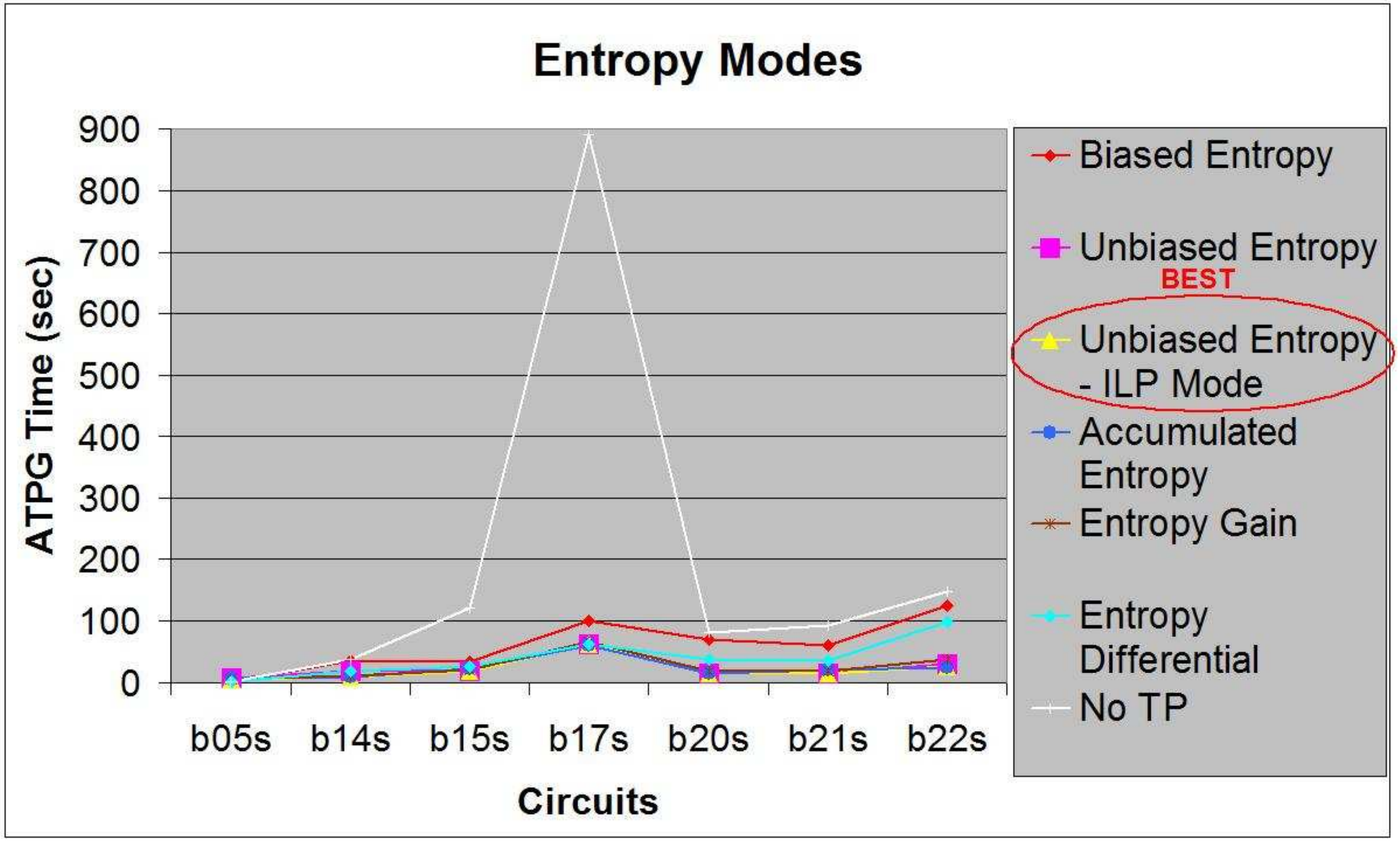

Figure 5.17: ATPG Time comparison: Entropy Modes

5.2.8 Entropy Run Time Results

The Run Time is the time taken by the proposed algorithm to insert the TPs in the circuit and it varies for each mode of the algorithm. The Run Time results for various modes of operation of the algorithm are shown in Table 5.8. The columns No TP and Best from the previous results tables have been omitted here, because No TP refers to circuits without running the algorithm on them. This table compares the run time for each mode, hence, highlighting the best run time for each circuit does not provide valuable information. The last row in Table 5.8 represents the average run time for the largest ITC '99 benchmark circuits of b14s to b22s. Figure 5.18 compares the run time of the Multi-TP mode with spectral-based results for the circuits b14s to b22s. The run times for other circuits are insignificantly low and the run time of Overlapping Spectra mode is very high compared to other modes, and hence, they are not included in the chart comparison. The Multi-TP mode is chosen to be compared with Spectral measures because it has the lowest average run time, 936.57 seconds on all circuits and 2940.11 seconds on larger circuits, among all of the Entropy modes. Figure 5.19 compares the run times of various Entropy-based results for b14s to b22s. From Table 5.8 and the Figures 5.18 and 5.19, it can be observed that the largest run time is achieved by using the Unbiased Entropy as the testability measure in Multi-TP mode.

Table 5.8: Run Time results (in sec) for ITC '99 Benchmark Circuits

Ckt.	Proba- -bility	Biased Entropy	Unbiased Entropy	Unbiased Entropy ILP mode	Accum- -ulated Entropy	Multi TP	Entropy Gain	Entropy Differ- -ential	Overlap Spectra Mode 1	Non-overlap Spectra			
										Mode 1	Mode 2	Mode 3	Mode 4
b01	0.29	0.63	0.63	2.61	0.62	0.62	0.68	0.59	302.33	10.10	3.38	3.39	3.44
b02	0.31	0.73	0.71	2.71	0.71	0.69	0.77	0.69	371.98	12.35	4.09	4.15	4.11
b03	2.20	4.79	4.78	9.63	4.80	4.63	5.11	4.68	2042.67	68.67	22.89	22.93	22.75
b04s	11.12	23.46	23.33	52.19	23.28	20.19	24.42	22.60	9282.91	315.55	105.61	105.74	104.69
b05s	27.82	52.94	53.05	84.90	52.79	45.45	54.89	50.99	14906.36	521.04	174.83	175.01	173.26
b06	0.69	1.57	1.57	5.07	1.57	1.52	1.70	1.51	775.52	26.50	8.81	8.79	8.71
b07s	9.17	18.58	18.52	38.33	18.59	16.75	19.29	18.02	6215.00	216.62	72.34	72.47	71.74
b08	2.85	5.89	5.91	13.87	5.90	5.65	6.26	5.84	2175.68	75.22	25.06	25.06	24.86
b09	2.27	4.95	4.94	14.37	4.92	4.61	5.27	4.88	2193.50	76.34	25.04	24.99	24.80
b10	2.95	6.17	6.20	14.16	6.17	5.78	6.64	6.23	2452.49	84.45	28.03	28.06	27.79
b11s	11.68	22.76	22.81	40.92	22.54	20.56	23.60	21.98	6962.88	244.69	81.58	81.77	80.92
b12	10.96	23.27	23.19	75.19	23.15	17.87	24.60	22.46	9054.62	320.64	105.99	106.12	105.12
b13s	4.87	10.69	10.72	29.44	10.66	9.79	11.37	10.35	4824.21	164.97	54.98	55.02	54.51
b14s	678.38	1278.51	1263.00	1461.96	1289.09	727.28	1160.13	1221.34	278617.26	10172.43	3413.36	3417.56	3386.54
b15s	542.95	1040.55	977.84	1349.22	985.66	504.39	955.39	942.53	183227.23	7384.09	2445.45	3452.48	2444.22
b17s	16251.75	29790.52	28175.29	29061.79	28141.68	9037.34	21611.97	27551.08	1594987.50	184533.72	60630.58	60673.97	60555.63
b20s	2334.49	4219.89	4162.06	4553.86	4101.91	1854.45	3766.20	4059.39	852636.19	31579.03	10522.05	10611.05	10432.53
b21s	2466.83	4554.80	4433.39	4843.18	4381.68	1985.36	3998.77	4451.83	905250.38	33519.42	11159.19	11283.12	11105.19
b22s	5063.65	7527.86	7685.72	10248.28	9605.13	3531.82	8102.33	7334.80	638325.44	56589.78	18698.81	18894.84	18482.93
Avg.	1443.43	2557.29	2467.04	2731.67	2562.15	**936.57**	2093.65	2406.94	237610.74	17153.45	5662.21	5739.29	5637.57
(b14s -b22s)	4556.34	8068.69	7782.88	8586.38	8084.19	**2940.11**	6599.13	7593.50	742174.00	53963.08	17811.57	18055.50	17734.51

Variability: 2940.11 + 739233.89

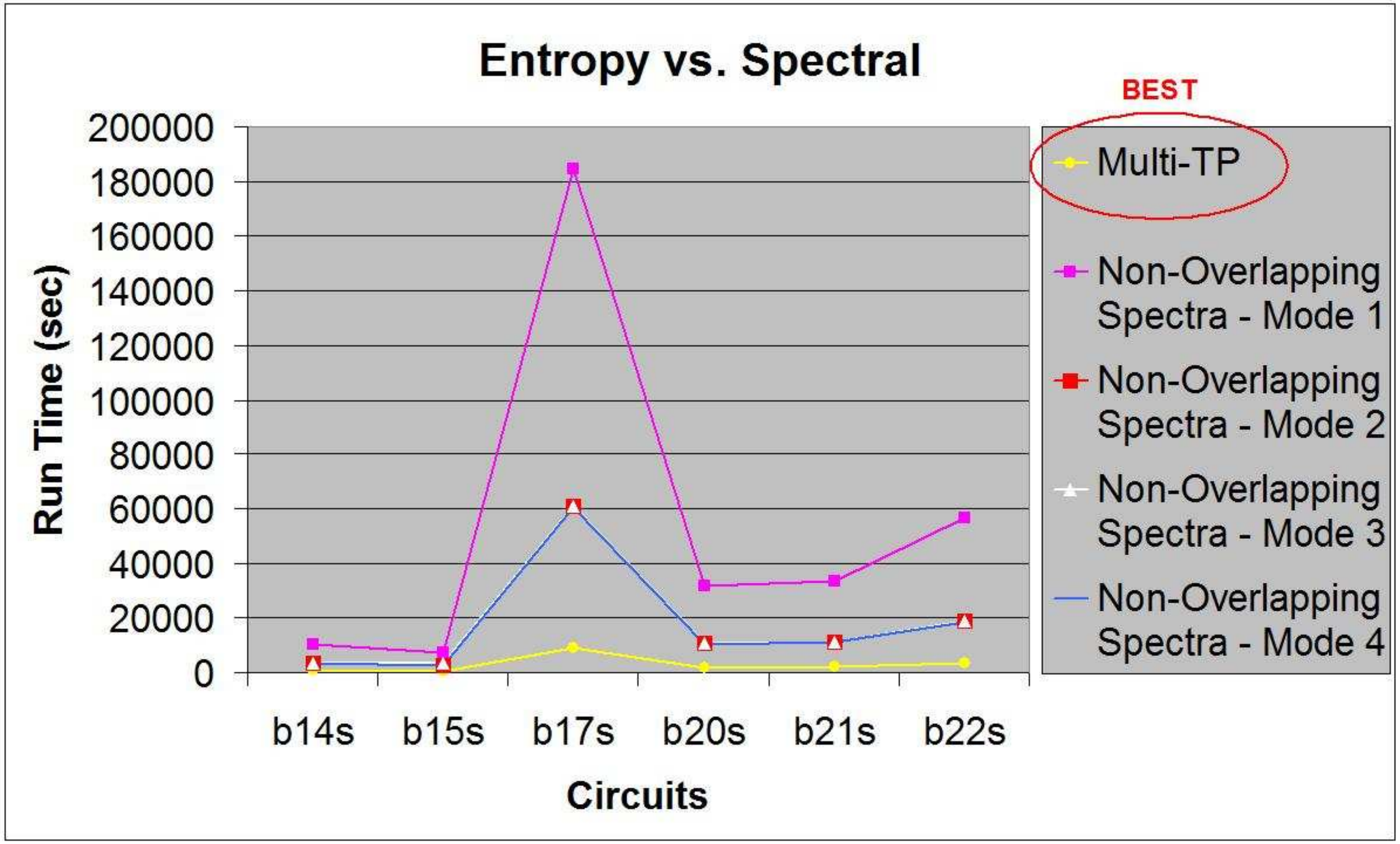

Figure 5.18: Run Time comparison: Entropy vs. Spectral

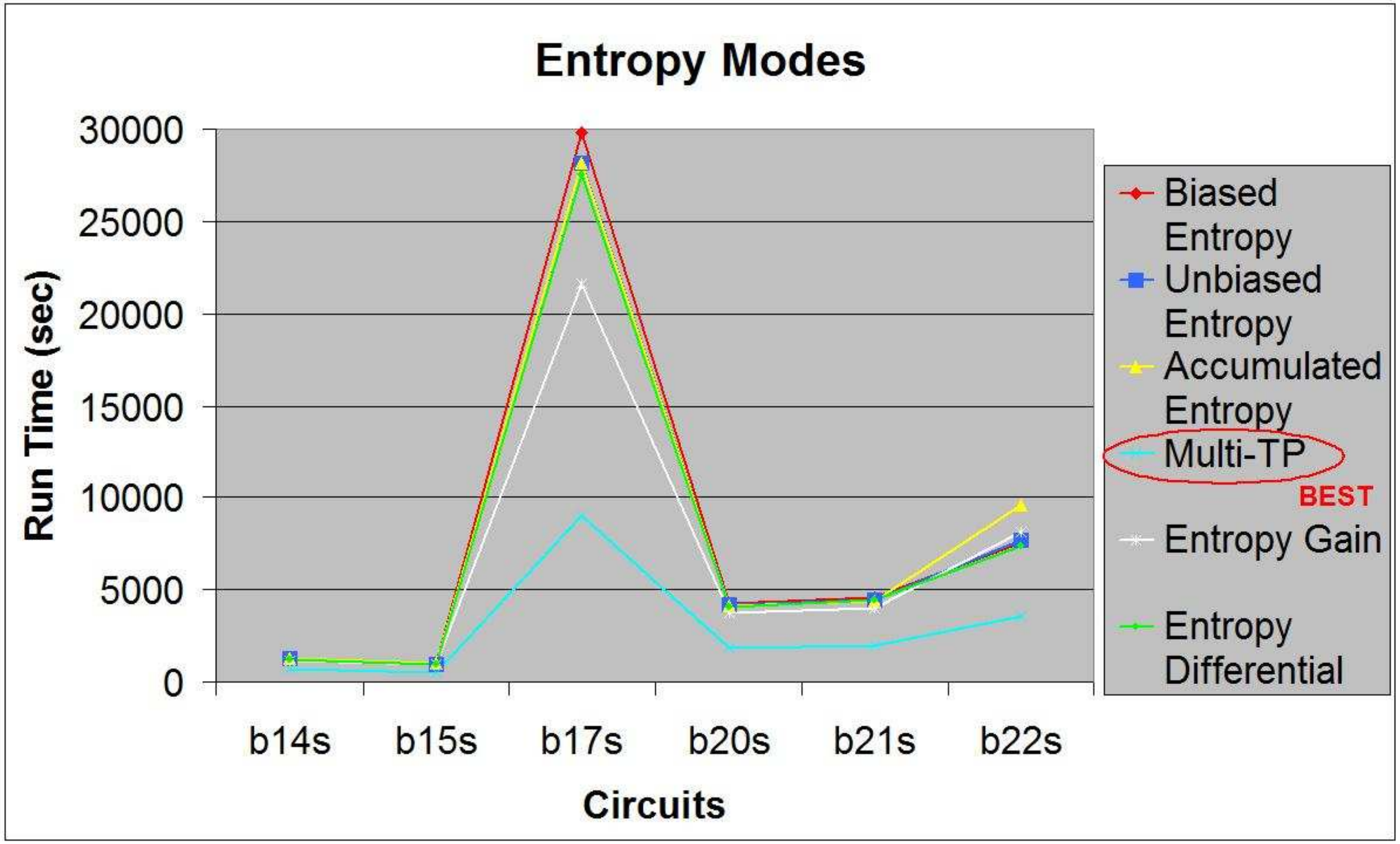

Figure 5.19: Run Time comparison: Entropy Modes

5.2.9 Calculation of Unbiasing factor

In the algorithm chapter, the calculation of the unbiasing factor to remove the statistical bias caused by pseudo-random vector simulation was discussed and is represented by Equation 4.6 and the unbiased probabilities, $p_u(1)$, were calculated using Equation 4.8. The equation for bias was derived after generating the above results for the full scan circuits. The values of θ chosen for the above experiments for various circuits are shown in Table 5.9. These values of θ were chosen empirically such that high performance is obtained from the algorithm. By choosing $\beta = 22$ in Equation 4.7, we get approximately the same values for θ as in Table 5.9.

Table 5.9: Unbiasing Factor for ITC '99 Circuits

Ckt.	θ
b01	0.01024
b02	0.00683
b03	0.00683
b04s	0.02
b05s	0.02
b06	0.02
b07s	0.02
b08	0.02
b09	0.02
b10	0.02
b11s	0.02
b12	0.01024
b13s	0.02
b14s	0.04
b15s	0.01024
b17s	0.01024
b20s	0.04
b21s	0.04
b22s	0.04

For calculating β and bias for new circuits, run the algorithm by choosing a θ and calculate β from the value of chosen θ. Then for future experiments with the same circuit, vary the calculated β to get better results.

5.2.10 Reasons for Algorithm Performance

Observing the results from the tables and figures, it is evident that the proposed algorithm performs well to reduce the test volume and test application time.

- Entropy is a superior testability measure that considers the amount of information flow in a circuit. By increasing the information flow, the testability of the circuit is increased and the results prove this point.
- Using the integer linear program solver enables the tool to select the best candidate from a random selection of candidates when there are more than one candidate with the same testability measure.
- We remove the bias caused by vector sampling, so that entropy becomes a more accurate measure, and the quality of test points chosen becomes better. This also reduces the number of pseudo-random vectors needed for simulation.

5.2.11 Summary of Full Scan with Test Point Results

Various results for different modes of operation of the algorithm were discussed. Analyzing from the tables and figures provided, the following observations can be drawn:

- For achieving the highest FC, the algorithm should be run in the Biased Entropy mode.
- For achieving the highest FE, the algorithm should be run in the Overlapped Spectra Mode 1.
- For achieving the shortest vector length, highest TV reduction and highest TAT reduction, the algorithm should be run in the Accumulated Entropy mode.
- For achieving the highest ATPG time reduction, the algorithm should be run in ILP mode using Unbiased Entropy as the testability measure.

- The algorithm runs fastest when run in Multi-TP mode.

The observations are based on ITC ’99 circuits, and the best result could be achieved by a different mode for other circuits. So it would be advisable to run the algorithm in all modes the first time on a new circuit and choose the best mode for further analysis.

5.3 Partial Scan with Test Point Insertion Results

The various options in the tool are described in Appendix A. The characteristics of the ISCAS ’89 benchmark circuits are shown in Table 5.10. It is assumed that the scan and test point chains are stitched into one single scan chain and for calculating TAT from Equation 5.5, $N = 1$ will be used. Each circuit was warmed up with a fixed number of warmup vectors and the testability measures were updated with a fixed number of update vectors. The PS penalty column in the table indicates the penalty of inserting a scan flip-flop in the ILP cost function and TP penalty column indicates the penalty for a TP.

5.3.1 Experimental Conditions

Experiments were run using the integer linear program on all of the ISCAS ’89 benchmarks, with the exception of s208 and s510, to generate the partial scan results. The circuit s208 is too small to show significant variation with and without DFT hardware and the circuit s510 is uninitializable with logic simulation. A C language program implemented the algorithm, and generated the constraints for the integer linear program. The integer linear programming algorithm was run using AMPL on an IBM PC, but the C program ran on a Sun workstation. Our partial scan with test point insertion results were generated using the sequential ATPG called GATEST [53] to keep our results consistent with the results reported by Khan *et al.* [35, 36, 37] and Xiang and Patel [70]. The parameters used for test generation using GATEST are listed in Table 5.11. The parameters that

Table 5.10: Characteristics and parameters of ISCAS '89 Benchmark Circuits

Ckt.	PIs	FFs	Gates	Warmup Vectors	Update Vectors	PS Penalty	TP Penalty
s298	3	14	119	51200	512	0.99	0.99
s344	9	15	160	51200	512	0.99	0.99
s349	9	15	161	51200	512	0.99	0.99
s382	3	21	158	51200	512	0.99	0.99
s386	7	6	159	51200	512	0.99	0.99
s400	4	21	162	51200	512	0.99	0.99
s444	3	21	181	51200	512	0.99	0.99
s526	3	21	193	51200	512	0.99	0.99
s641	35	19	379	51200	512	0.99	0.99
s713	35	21	393	51200	512	0.99	0.99
s820	18	5	289	51200	512	0.99	0.99
s953	16	29	395	51200	512	0.99	0.99
s1423	17	74	657	51200	512	0.99	0.99
s1488	8	6	653	51200	512	0.99	0.99
s1494	8	6	347	51200	512	0.99	0.99
s5378	35	179	2779	51200	512	0.99	0.99
s9234	19	228	5597	51200	512	0.99	0.99
s13207	31	669	7951	51200	512	0.99	0.99
s15850	77	534	9772	51200	512	0.99	0.99
s35932	35	1728	16065	51200	512	0.99	0.99
s38417	28	1636	22179	51200	512	0.99	0.99
s38584	12	1452	19253	51200	512	0.99	0.99

Table 5.11: GATEST parameters

Option	Description	Value	Circuits
-v	Maximum number of vectors in a test sequence	200	s298, s444
		default	others
-g	Number of generations used in genetic algorithm	8	s298, s444
		32	others

are not shown in the table use default values. The vectors generated by GATEST are compacted using a *reverse order restoration* (ROR) compactor [22].

5.3.2 Fault Coverage and Overhead Results

The results shown in Table 5.12 were generated with options *pensff* and *pentp* set to 0.99. The sixth column SFF under SPARTAN indicates the number of scan flip-flops inserted for all the three ideas of PS only, PS + TP1 and PS + TP2. The authors of *mpscan*, Xiang and Patel, did not provide the results for the circuit s38584 and hence, they are left blank in the corresponding columns of the table. The values shown in bold are the best result for that particular circuit. From Table 5.12 we can infer that the proposed algorithm has a better FC than SPARTAN and *mpscan*. The proposed idea has a slightly higher hardware overhead than the partial scan idea of Khan *et al.* [35, 36, 37], but very high overhead compared to *mpscan* [70]. The higher overhead is incurred to achieve a higher FC and better test vector length, TV and TAT as will be explained in the next section. When compared to the best average FC reported by Khan *et al.*, which is their PS + TP2 idea, the proposed entropy algorithm achieves higher FC with two-thirds of the incurred overhead. Figures 5.20 and 5.21 compare the FC and hardware overhead results for the seven biggest circuits in the ISCAS '89 circuits, respectively.

Table 5.12: Partial-Scan Scanned Flip-Flop Count, Test Point Count, Fault Coverage, and Hardware Overhead for ISCAS '89 Circuits

Ckt.	ILP+Entropy				SPARTAN [35, 36, 37]									mpscan		
	Partial-Scan				SFF	PS Only		PS + TP1			PS + TP2			[70]		
	SFF	TP	FC	O		FC	O	TP	FC	O	TP	FC	O	SFF	FC	O
s298	7	0	99.35	23.08	11	99.67	36.26	5	**99.68**	48.80	4	99.68	46.29	2	98.70	**6.59**
s344	5	10	99.45	33.24	7	99.71	18.46	1	**99.71**	20.47	3	99.71	24.48	3	99.40	**7.91**
s349	5	10	98.92	33.10	8	99.14	21.01	27	**99.75**	74.92	4	99.16	28.99	3	98.90	**7.88**
s382	10	9	99.52	40.02	17	99	40.40	1	99	42.20	16	**99.54**	69.31	6	99.00	**14.26**
s386	6	8	**100.00**	38.98	4	99.48	12.90	25	**100.00**	74.23	10	**100.00**	37.43	4	**100.00**	**12.90**
s400	10	7	**99.1**	35.84	16	94.6	37.43	1	94.63	39.21	16	95.42	65.89	4	95.80	**9.36**
s444	11	10	**99.19**	40.52	18	96.84	39.20	1	95.38	40.86	19	98.05	70.67	6	96.20	**13.07**
s526	12	4	99.28	31.39	19	99.64	39.65	1	**99.82**	41.24	3	99.82	44.41	10	99.30	**20.87**
s641	14	17	**100.00**	34.78	9	99.14	11.63	40	99.64	50.92	6	99.58	17.52	1	99.40	**1.29**
s713	11	11	94.36	23.83	9	92.94	11.08	39	**95.14**	47.58	6	93.09	16.69	1	92.90	**1.23**
s820	5	1	**100.00**	11.10	2	99.69	**3.85**	187	**100.00**	277.8	48	**100.00**	74.16	2	**100.00**	**3.85**
s953	5	18	**100.00**	21.34	23	**100.00**	26.26	3	**100.00**	28.87	3	**100.00**	28.87	3	**100.00**	**3.43**
s1423	41	5	97.97	27.15	63	98.75	38.18	19	**98.78**	46.94	3	98.75	39.56	41	98.10	**24.85**
s1488	5	5	**100.00**	7.77	5	**100.00**	4.41	389	**100.00**	265.4	58	**100.00**	43.33	2	**100.00**	**1.77**
s1494	6	9	99.21	20.61	4	99.2	6.42	399	**100.00**	493.2	26	99.42	38.14	2	99.10	**3.21**
s5378	63	61	99.01	18.31	114	98.81	19.08	2	**98.87**	19.34	44	98.78	24.68	50	97.20	**8.37**
s9234	124	148	**98.08**	21.43	199	93.32	18.03	7	93.27	18.51	102	94.81	25.06	97	93.00	**8.79**
s13207	196	179	90.54	18.18	496	97.77	27.15	21	**98.02**	28.02	20	97.82	27.98	58	85.60	**3.18**
s15850	198	113	94.65	13.99	407	93.08	20.06	30	92.74	21.18	7	93.33	20.32	180	**94.80**	**8.87**
s35932	565	0	89.76	14.22	477	89.8	12.00	27	**89.92**	12.52	18	89.83	12.35	150	89.80	**3.78**
s38417	499	331	97.48	15.25	1249	98.44	25.37	27	98.48	25.79	201	**98.82**	28.47	400	94.50	**8.12**
s38584	372	155	**94.83**	**11.40**	924	94.54	21.50	24	94.38	21.92	87	94.25	23.04	-	-	-
Avg.	98.6	50.5	**97.76**	24.34	185.5	97.43	22.29	58	97.60	79.08	32	97.72	36.71	48.81	96.75	**8.27**

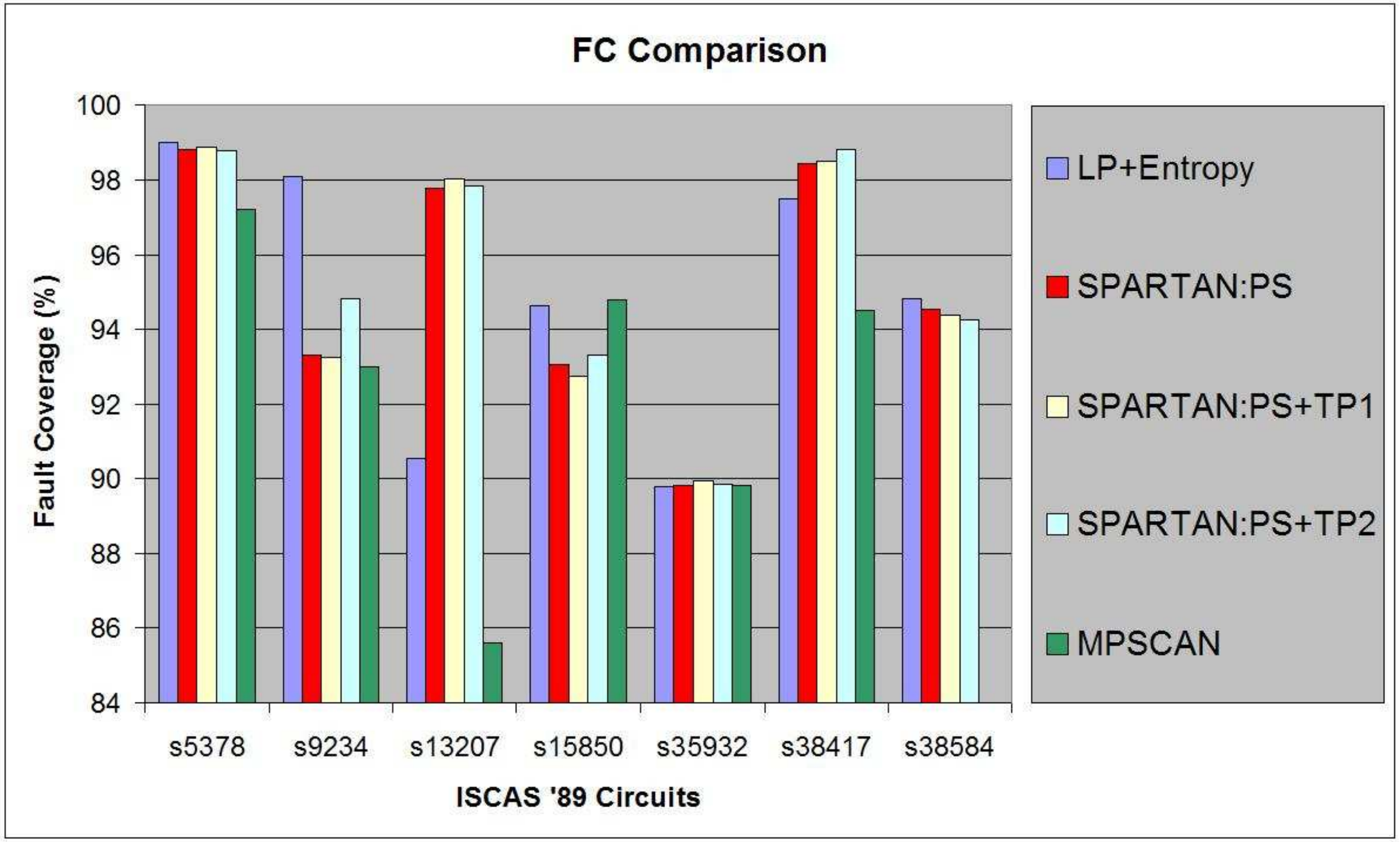

Figure 5.20: Fault Coverage comparison

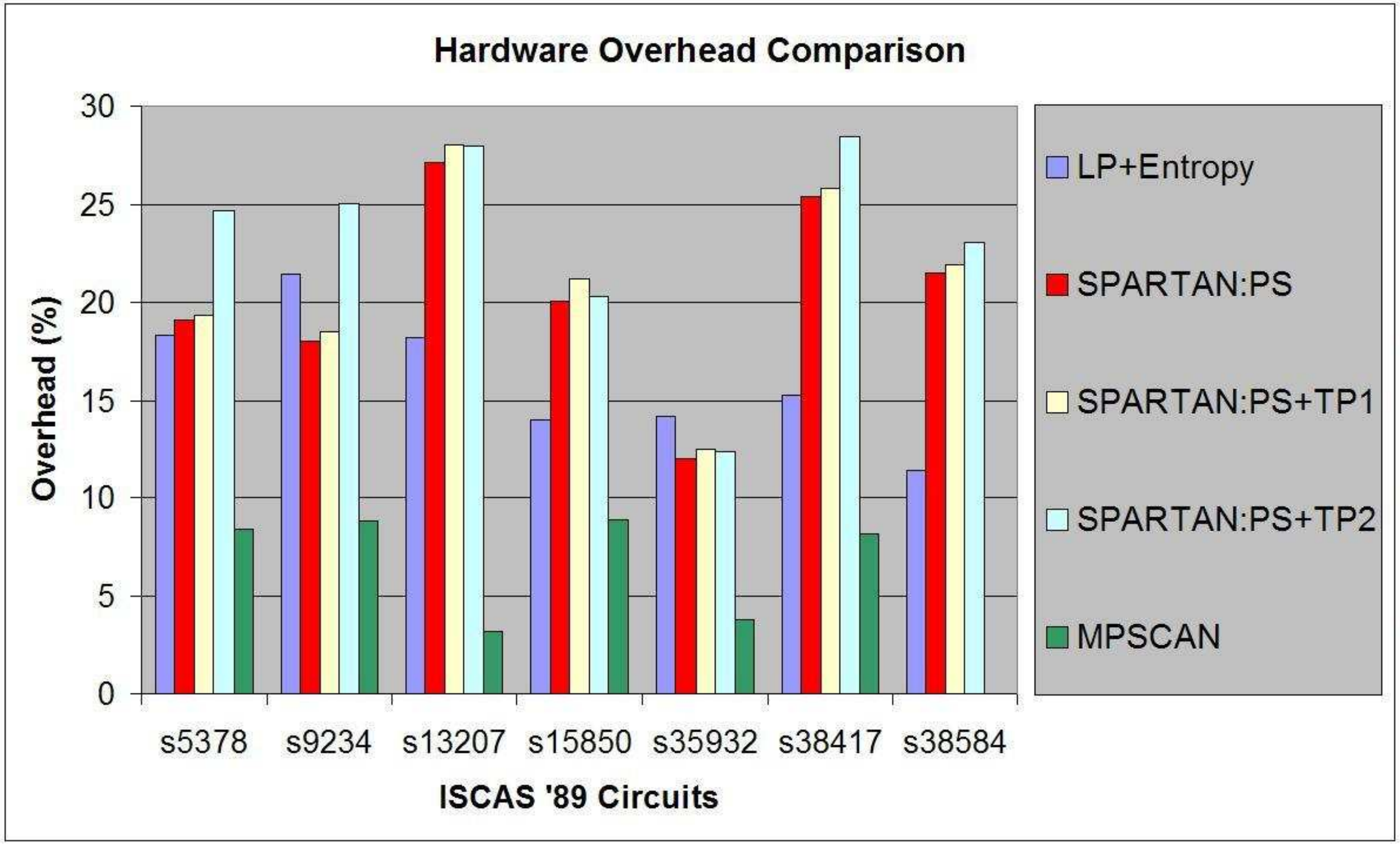

Figure 5.21: Hardware Overhead comparison

5.3.3 Test length, Test Volume and Test Application Time Results

Table 5.13 gives the results for Test Vector Length, Test Volume and Test Application Time. The authors of *mpscan*, Xiang and Patel, did not provide the results for the circuit s38584 and hence, they are left blank in the corresponding columns of the table. The values shown in bold are the best result for that particular circuit. Even though the average vector length for the proposed algorithm is higher than that of the SPARTAN PS+TP1 and PS+TP2 ideas, since we insert less DFT hardware than SPARTAN, we get TV reductions of 19.56% and 33.42% and TAT reductions of 21.63% and 31.23%, over the SPARTAN PS+TP1 and PS+TP2 ideas, respectively. As explained in previous section, more test hardware is inserted by the proposed algorithm than by *mpscan*, so that a better V, TV and TAT are obtained for the circuits. It can be observed from the table that we get 32.62% TV reduction and 25.39% TAT reduction over the *mpscan* algorithm. Figures 5.22, 5.23 and 5.24 compare the V, TV and TAT results for the seven biggest circuits in the ISCAS '89 circuits, respectively.

Table 5.13: Partial-Scan Test Vector Count, Test Volume and Test Application Time for ISCAS '89 Circuits

Ckt.	ILP+Entropy			SPARTAN [35, 36, 37]									mpscan		
	Partial-Scan			PS Only			PS+TP1			PS+TP2			[70]		
	V	TV	TAT	V	TV	TAT	V	TV	TAT	V	TV	TAT	V	TV	TAT
s298	39	**429**	**330**	37	555	470	**36**	720	648	41	779	690	160	960	488
s344	**43**	1075	722	65	1105	538	63	1134	587	50	**1000**	**574**	141	1833	**574**
s349	33	825	562	50	900	470	**22**	990	866	33	**726**	**457**	161	2093	654
s382	**54**	**1242**	**1122**	215	4515	3908	164	3608	3156	93	3441	3232	168	1680	1192
s386	**63**	**1386**	**977**	119	1428	607	89	3293	2732	94	2068	1442	205	2460	1037
s400	**35**	**770**	**668**	236	4956	4048	252	5544	4574	79	2923	2675	394	3546	1982
s444	**44**	**1100**	**1014**	158	3476	3042	116	2668	2362	91	3731	3536	193	1930	1367
s526	**67**	**1340**	**1175**	131	3013	2662	112	2688	2396	105	2730	2463	273	3822	3027
s641	**25**	**1675**	866	107	4815	1092	108	9180	5502	115	5865	1874	286	10582	**578**
s713	**52**	**3016**	1244	125	5625	1272	105	8820	5245	130	6630	2114	310	11470	**626**
s820	112	**2800**	**800**	349	7329	1055	**93**	19344	18052	171	11799	8825	602	12642	1814
s953	146	5840	3554	98	3920	2402	**90**	**3870**	2486	**90**	3870	2486	339	6780	**1366**
s1423	111	7104	**5313**	131	10611	8514	**68**	**6800**	5812	95	7980	6501	397	23423	16760
s1488	124	2356	1388	136	**1904**	**830**	**56**	22568	22912	131	9432	8514	639	7029	1925
s1494	110	2640	1794	179	**2327**	**907**	**66**	27192	27474	191	7449	5985	622	6842	1874
s5378	**485**	**77600**	60877	535	80250	61757	556	84512	65288	517	100298	82523	1023	87978	**52277**
s9234	**427**	**124684**	**117119**	958	209802	192002	717	162042	148835	860	276060	260326	3114	364338	305370
s13207	1478	601546	556482	879	**464112**	**437859**	**875**	480375	454288	1044	572112	540784	9805	882450	578615
s15850	1585	616565	495146	3906	1894410	1594466	**661**	**340415**	**290396**	832	409344	346112	4527	1167966	819751
s35932	218	131018	124522	261	133893	125716	166	89640	84842	**150**	79650	75394	252	**46872**	**38356**
s38417	**2694**	**2314146**	**2240378**	3113	3978414	3893752	3066	4001130	3917838	3433	5077407	4984187	11573	4964817	4641577
s38584	2729	1473660	1441970	1491	1397067	1381027	1460	**1403060**	**1387440**	**1449**	1483776	1468414	-	-	-
Avg.	485	**244219**	**229910**	603	373383	350836	**406**	303617	293351	445	366775	354959	1675	362453	308152

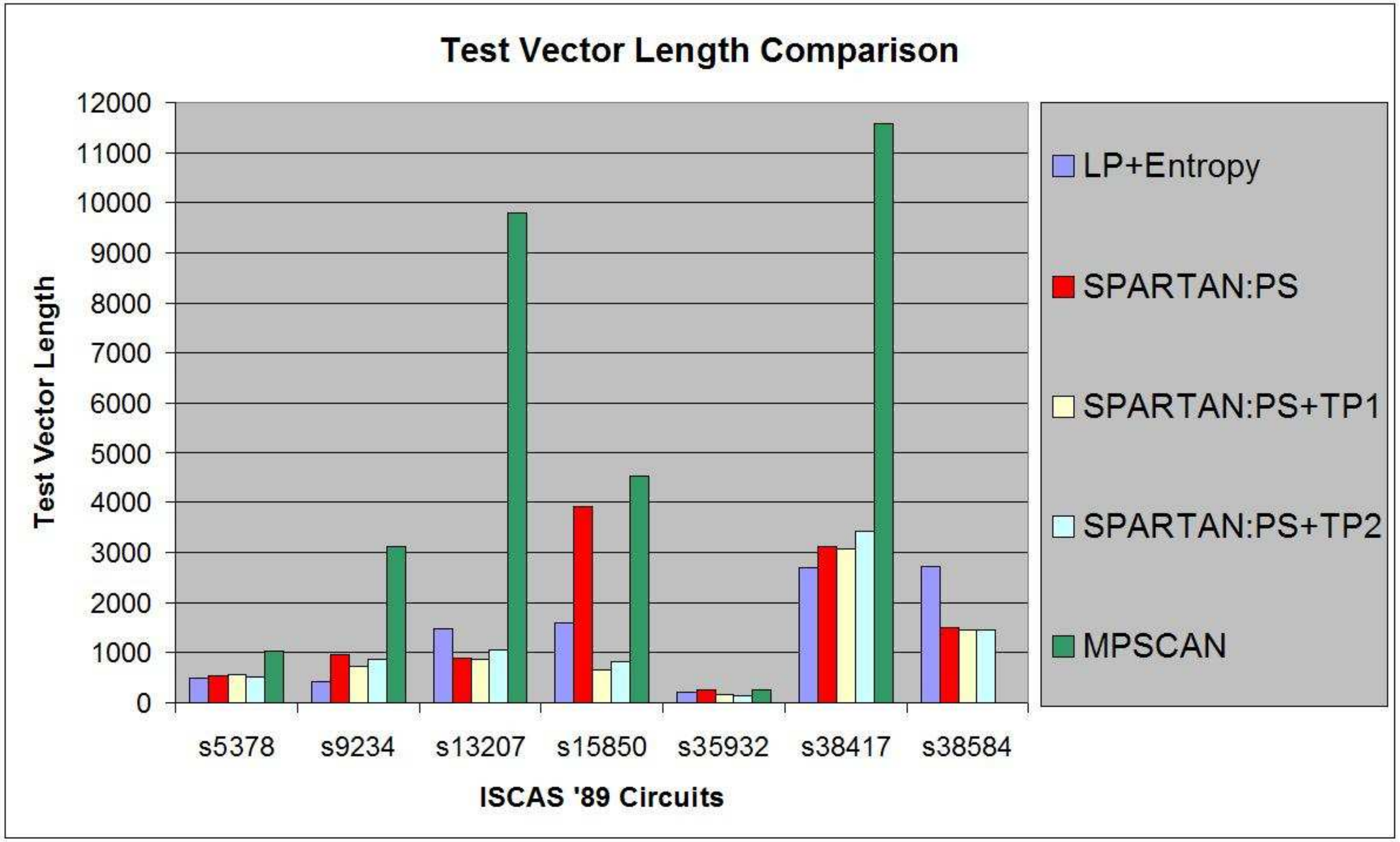

Figure 5.22: Test Vector Length comparison

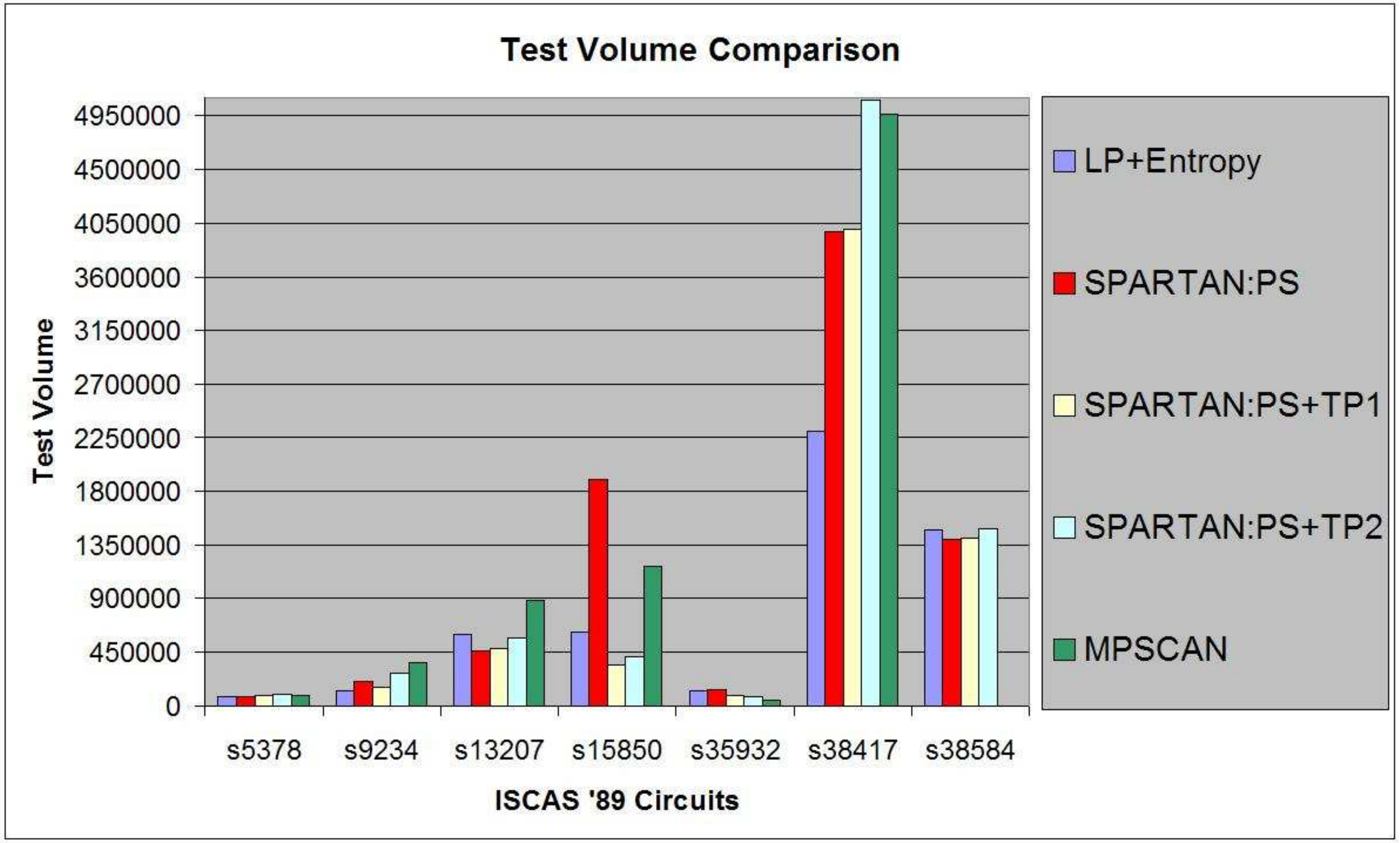

Figure 5.23: Test Volume comparison

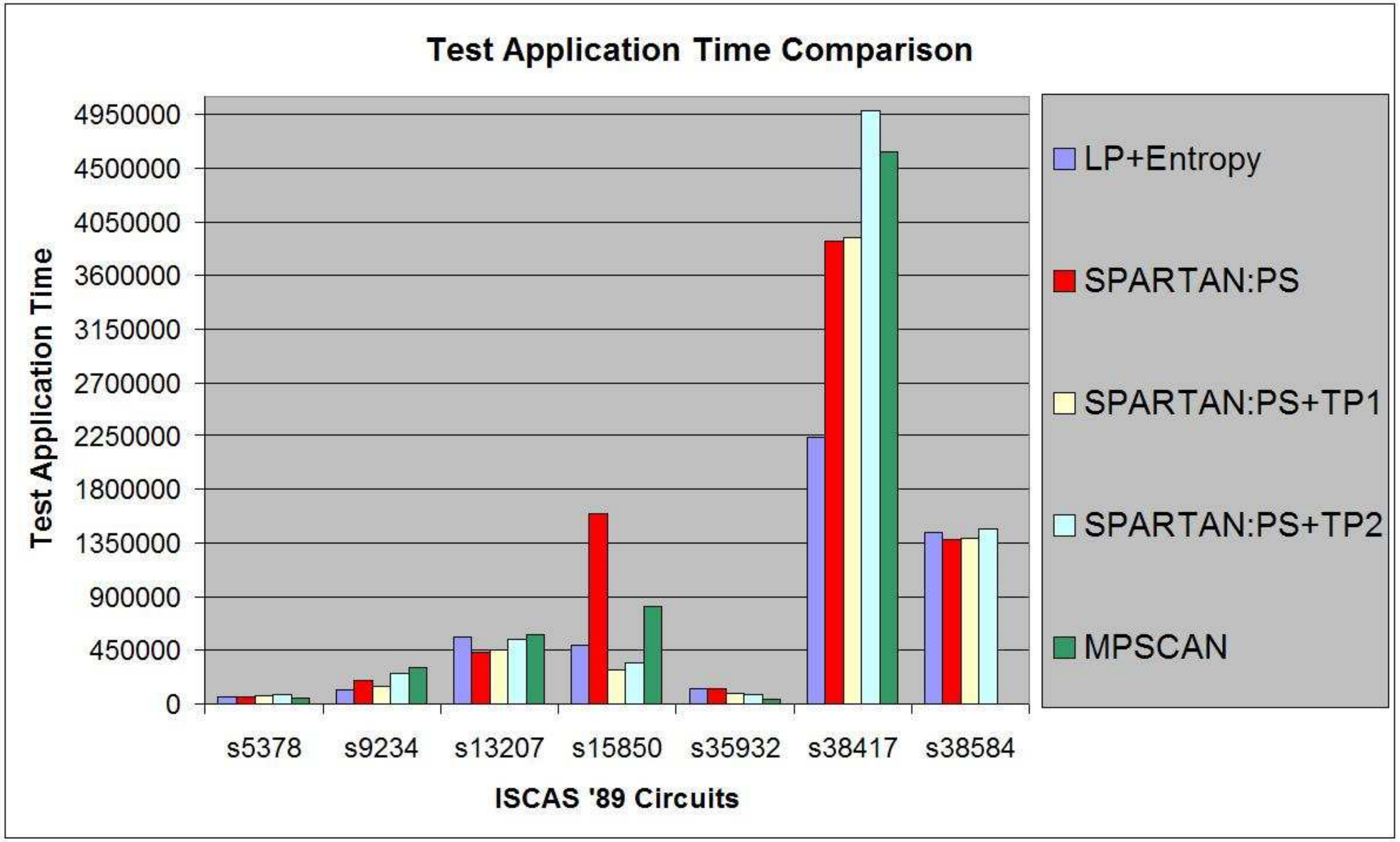

Figure 5.24: Test Application Time comparison

5.3.4 Reasons for Algorithm Performance

The ideas in the proposed entropy algorithm that make it perform better than the *SPARTAN* and *mpscan* algorithms are:

- By simultaneously selecting scan flop and test point candidates, the tool can evaluate the advantages of inserting a test point versus scanning a flip-flop at each step and make a superior decision. This way a circuit with better testability can be obtained by inserting less DFT hardware.
- Using the integer linear program solver enables the tool to select the best candidate from a random selection of candidates when there are more than one candidate with the same testability measure.
- We remove the bias caused by vector sampling, so that entropy becomes a more accurate measure, and the quality of scan flip-flops and test points chosen becomes better.

5.3.5 Summary of Partial Scan with Test Point Results

The proposed algorithm has a better FC than SPARTAN and *mpscan.* The proposed idea has a slightly higher hardware overhead than the partial scan idea of Khan *et al.*, but very high overhead compared to *mpscan.* When compared to the best average FC reported by Khan *et al.*, which is their PS + TP2 idea, the proposed entropy algorithm achieves higher FC with two-thirds of the incurred overhead. Even though the average vector length for the proposed algorithm is higher than that of the SPARTAN PS+TP1 and PS+TP2 ideas, since we insert less DFT hardware than SPARTAN, we get TV reductions of 19.56% and 33.42% and TAT reductions of 21.63% and 31.23%, over the SPARTAN PS+TP1 and PS+TP2 ideas, respectively. More test hardware is inserted by the proposed algorithm than by *mpscan*, so that a better V, TV and TAT are obtained for the circuits. We achieve 32.62% TV reduction and 25.39% TAT reduction over the *mpscan* algorithm.

Chapter 6
Implementation

The algorithm explained in the previous chapter has been implemented in the C language and uses the Rutgers *application programming interface* (API) for netlist parsing and modification. The algorithm reads in the netlist represented in the rutmod netlist language and writes out a rutmod file of the circuit containing DFT hardware that can be directly used for ATPG. When the circuit operates in ILP mode, the data file has to be created, sent to the AMPL software and the output file has to be read back. The following sections describe how these steps are implemented.

6.1 Writing out the Data file

A data file has to be written, which can be used by AMPL to select a test-point candidate or scan-flop candidate based on the testability measure. The data file has to be syntactically correct so that AMPL can parse it without any error. The syntax for the data file is provided by Gay *et al.* [24]. A sample data file can be found in Appendix A. The following are the parameters used in the data file:

- **gate:** Set of all the logic gates present in the circuit
- **flop:** Set of all the flip-flops present in the circuit
- **cycles:** Each SCC in the circuit is represented by a cycle number. Parameter *cycles* represents the set of all cycle numbers of the SCCs present in the circuit, with size greater than 1, that is, there are more than 1 flip-flops in that SCC.

- **L:** Parameter representing the total number of logic gates present in the circuit
- **F:** Parameter representing the total number of flip-flops present in the circuit
- **nCycles:** Parameter representing the total number of SCCs present in the circuit
- **E:** Parameter representing the testability measure of each gate
- **SCC:** Parameter representing the set of flip-flops present in each SCC
- **k:** Parameter representing the DFT weight. It governs the amount of DFT hardware being inserted into the circuit

The data file, "circuit.dat", will be generated with the above parameters. Whenever testability measures are recalculated, a new data file has to be generated.

6.2 Transferring Data and Output Files

After the data file is written out, it is sent to the AMPL software. The AMPL software is installed on a PC while the code to generate the data file runs on a Unix Workstation. Hence, we need to send the data file from the workstation to the PC and the output file of AMPL from the PC to the workstation.

6.3 Secure Shell

Secure Shell or SSH is a network protocol that allows data to be exchanged using a secure channel between two networked devices. SSH uses public-key cryptography to authenticate the remote computer and allow the remote computer to authenticate the user, if necessary. The following are some uses of SSH:

- For login to a shell on a remote host
- For executing a single command on a remote host
- For copying files from a local server to a remote host.

Another advantage of SSH is that is provides key-based authentication, which is very helpful for automated file transfer between a server and a remote host, as in this case. The data and output files have to be sent back and forth every time a new candidate is selected and testability measures are recalculated. It is not feasible for an administrator to enter a username and password for every file transfer. Hence, we choose SSH to automate the file transfer.

6.4 Reading the Output File

The output file from the AMPL solver consists of five candidates. This file is parsed to obtain the candidates and one best candidate is chosen depending upon the testability measure being used. A typical output file is shown below, where Y represents the gate being a flip-flop and X represents the gate being a logic gate.

```
:     Y   X   :=
16    1   .
59    .   1
64    .   1
92    .   1
102   .   1
;
```

6.5 Summary

This chapter explains how the algorithm is implemented and how the ILP mode is handled in the software. The data file required by the AMPL solver is created and sent to AMPL using the secure shell. The secure shell is also used for firing the AMPL solver to solve the model file using the data file. The output file from AMPL is copied back to the host where the algorithm is being run. The candidates from the output file are read and the best one is selected from among them depending on the testability measure used. Secure shell is chosen because it provides key-based authentication that is very helpful for automated file transfer.

Chapter 7
Conclusions and Future Work

In Chapter 3, we discussed the integer linear programming approach and the formulation used by the algorithm to select the test point candidates. Chapter 4 explained the algorithm in detail including the various testability measures used and provided a flow chart of the entire procedure. Chapter 5 discussed in detail the various results obtained for various runs of the algorithm. Chapter 6 explained the implementation of the algorithm. In this chapter we will conclude with the importance of this work and look ahead to the future work.

7.1 Conclusions

This work has provided these original ideas:

- The first use of a combined algorithm to make the trade-off between inserting a scan flip-flop and inserting a test point. Prior algorithms did these operations in separate passes, with the result that they inserted unnecessary test points and unnecessarily scanned flip-flops.
- The first use of entropy measures exclusively for test point and scan flip-flop insertion. The prior methods relied on a combination of entropy measures and spectral analysis.
- A novel way of calculating the statistical sample bias arising due to input vector sampling and a method to remove the bias. Our bias removal improves the algorithm performance greatly when compared to using biased entropy as the testability measure. Shorter vector sequences can be used, which greatly accelerates logic simulation.

- The second use of integer linear programming to select places in a circuit for scanning flip-flops and the first use of it for inserting test points. Because an integer linear program can find the optimal solution to a problem with many millions of variables, this approach scales up to the size of circuits in practical use today.
- Entropy derivative measures such as entropy gain and entropy differential that account for both difficult-to-control and difficult-to-observe gates.
- A high speed bit-wise computation to multiply the 32×32 Hadamard matrix with the output of each gate to calculate its spectral co-efficients. The matrix multiplication is replaced with a bit-wise XNORing operation.

The algorithm can insert test points into a full-scan circuit or it can insert test points and scan flip-flops into a partial scan circuit depending upon the need of the user.

Full-scan with Test Points

On full-scan circuits, our method achieves a dramatic 40.05% TV reduction and 54.24% TAT reduction while acheiving 99.60% FC and 100% FE with 10% hardware overhead on the ITC '99 benchmark circuits. Our method also reduces the time spent on TetraMAX ATPG by 90.24%. The following summarizes the best mode for operating the algorithm to acheive the best result for each characteristic of the circuit:

- For achieving the highest FC, the algorithm should be run in the Biased Entropy mode.
- For achieving the highest FE, the algorithm should be run in the Overlapped Spectra Mode 1.
- For achieving the shortest vector length, highest TV reduction and highest TAT reduction, the algorithm should be run in the Accumulated Entropy mode.

- For achieving the highest ATPG time reduction, the algorithm should be run in ILP mode using Unbiased Entropy as the testability measure.
- The algorithm runs fastest when run in Multi TP mode.

Partial scan with Test Points

On partial scan circuits, our method achieves TV reductions of 19.56% and 33.42% and TAT reductions of 21.63% and 31.23%, over the previous best SPARTAN PS+TP1 and PS+TP2 partial scan ideas [35, 36, 37], respectively, on ISCAS '89 benchmark circuits. We also get 32.62% TV reduction and 25.39% TAT reduction over the *mpscan* algorithm [70]. Our method acheives the highest FC among the above-mentioned methods and inserts less DFT hardware than SPARTAN but more than the *mpscan* algorithm.

7.2 Future Work

Several ideas can easily be applied to this method to improve it further. The designer can add a *weight* to each circuit line, which removes it from consideration for test hardware when the *weight* exceeds certain limit. The greater the number, the less likely the DFT algorithm is to insert DFT hardware on that line. This drives the DFT hardware off critical paths, and forces the algorithm to position DFT hardware on non-critical delay paths. This idea has been implemented in the industrial version of the same algorithm as a product of the company Spectral Design and Test, Inc.

Another improvement to the algorithm could be to add a fault simulator that adds faults at each line and the algorithm can be made to select candidates based on the number of faults that will be detected without needing to change the inputs at PIs and thus reducing the TV and vector length.

To make the algorithm faster, the SCC forming routine can be replaced with a faster algorithm such as Tarjan's Algorithm [63], which uses only one DFS to visit the nodes rather than using two DFS's. This idea has been implemented in the industrial version of the same algorithm as a product of the company Spectral Design and Test, Inc.

The algorithm could insert control-to-0, control-to-1 and observe-only test points into the circuit based on the probability values for each candidate. This can reduce the hardware overhead incurred by test point insertion. This idea has been implemented in the industrial version of the same algorithm as a product of the company Spectral Design and Test, Inc.

The TPI algorithm could drive the control-to-0 and control-to-1 test points using gates with high probability of logic 0 or logic 1, respectively. These test points then need not be stitched into a scan-chain or they need not be driven from a primary input.

References

[1] M. Abramovici, J. J. Kulikowski, and R. R. Roy. The Best Flip-Flops to Scan. In *Proc. of the Int'l. Test Conf.*, pages 166–173, 1991.

[2] V. D. Agrawal, K. T. Cheng, D. D. Johnson, and T. Lin. Designing Circuits with Partial Scan. *IEEE Design and Test of Computers*, 5(1):8-15, March 1988.

[3] V.D. Agrawal. An Information Theoretic Approach to Digital Fault Testing. *IEEE Transactions on Computers*, C-30, no. 8:582–587, August 1981.

[4] P. Ashar and S. Malik. Implicit Computation of Minimum-Cost Feedback Vertex Sets for Partial Scan and other Applications. In *Proc. of the ACM/IEEE Design Automation Conf.*, pages 77–80, 1994.

[5] G. Bell. Bell's Law for the Birth and Death of Computer Classes. *Commun. of the ACM*, 51(1):86–94, 2008.

[6] S. Bhawmik, C. J. Lin, K. T. Cheng, and V. D. Agrawal. Pascant: A Partial Scan and Test generation System. *Proc. of the Custom Integrated Circuits Conf.*, pages 17.3.1–17.3.4, 1992.

[7] V. Boppana and W. K. Fuchs. Partial Scan Design Based on State Transition Modeling. In *Proc. of the Int'l. Test Conf.*, pages 538–547, 1996.

[8] S. Boubezari, E. Cerny, B. Kaminska, and B. Nadeau-Dostie. Testability Analysis and Test Point Insertion in RTL VHDL Specifications for Scan-Based BIST. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 18(9):1327–1340, Sept. 1999.

[9] F. Brglez, D. Bryan, and K. Kozminski. Combinational Profiles of Sequential Benchmark Circuits. In *Proc. of the IEEE International Symposium on Circuits and Systems*, pages 1929–1934, vol. 3, May 1989.

[10] M. L. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits.* Kluwer Academic Publishers, Boston, 2000.

[11] S. Chakradhar, V. D. Agrawal, and S. Rothweiler. A Transitive Closure Algorithm for Test Generation. *IEEE Trans. on Computer-Aided Design*, 12(7):1015–1028, July 1993.

[12] S. Chakradhar, A. Balakrishnan, and V. D. Agrawal. An Exact Algorithm for Selecting Partial Scan Flip-Flops. In *Proc. of the 31st ACM/IEEE Design Automation Conference*, pages 81–86, 1994.

[13] K. T. Cheng. Single-Clock Partial Scan. *IEEE Design and Test of Computers*, 12(2):24–31, 1995.

[14] K. T. Cheng and V. D. Agrawal. A Partial Scan Method for Sequential Circuits with Feedback. *IEEE Trans. on Computers*, 39(4):544–548, April 1990.

[15] K. T. Cheng and C. J. Lin. Timing-Driven Test Point Insertion for Full-Scan and Partial-Scan BIST. In *Proc. of the Int'l. Test Conference*, pages 506–514, Oct. 1995.

[16] V. Chickermane and J. Patel. An Optimization Based Approach to the Partial Scan Problem. In *Proc. of the Int'l. Test Conf.*, pages 377–386, Oct. 1990.

[17] V. Chickermane and J. H. Patel. A Fault Oriented Partial Scan Design Approach. In *Proc. of the Int'l. Conf. on Computer-Aided Design*, pages 400–403, Nov. 1991.

[18] V. Chickermane, E. M. Rudnick, P. Banerjee, and J. H. Patel. Non-Scan Design-for-Testability Techniques for Sequential Circuits. In *Proc. of the Design Automation Conference*, pages 236–241, 1993.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* Boston: McGraw-Hill Book Company, 2006.

[20] F. Corno, P. Prinetto, M. Sonza Reorda, and M. Violante. Exploiting Symbolic Techniques for Partial Scan Flip-Flop Selection. In *Proc. of the IEEE Design, Automation, and Test in Europe Conf.*, pages 670–677, 1998.

[21] S. Davidson. ITC '99 Benchmark Circuits - Preliminary Results. In *Proc. of the International Test Conference*, pages 1125–1125, 1999.

[22] S. K. Devanathan, O. I. Khan, and M. L. Bushnell. PROR Compaction Scheme for Larger Circuits and Long Vector Sequences. Unpublished. 2007.

[23] J.A. Dussault. A Testabilty Measure. In *Proc. of IEEE 1978 Semiconductor Test Conf.*, pages 113–116, October 1978.

[24] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Boston, Massachusetts: Boyd and Fraser Publishing Company, 1993.

[25] P. Goel. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Transactions on Computers*, 30(3):215–222, March 1981.

[26] L. H. Goldstein and E. L. Thigpen. SCOAP: Sandia Controllability/Observability Analysis Program. In *Proc. of the Design Automation Conference*, pages 190–196, June 1980.

[27] R. Gupta, R. Gupta, and M. A. Breuer. The BALLAST Methodology for Structured Partial Scan Design. *IEEE Trans. on Computers*, 39(4):538–544, April 1990.

[28] K. Heragu, V. D. Agrawal, and M. L. Bushnell. Fault Coverage Estimation by Test Vector Sampling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(5):590–596, May 1995.

[29] M. S. Hsiao, G. S. Saund, E. M. Rudnick, and J. H. Patel. Partial Scan Selection Based on Dynamic Reachability and Observability Information. In *Proc. of the Int'l. Conf. on VLSI Design*, pages 174–180, 1998.

[30] V. Iyengar and D. Brand. Synthesis of Pseudo-Random Pattern Testable Designs. In *Proc. of the Int'l. Test Conf.*, pages 508–601, Aug. 1989.

[31] S. K. Jain and V. D. Agrawal. Statistical Fault Analysis. *IEEE Design and Test of Computers*, 2(1):38–44, Feb. 1985.

[32] N. Jiang, R. M. Chou, and K. K. Saluja. Synthesizing Finite State Machines for Minimum Length Synchronizing Sequences Using Partial Scan. In *Proc. of the IEEE Int'l. Fault-Tolerant Computing Symp.*, pages 41–49, 1995.

[33] P. Kalla and M. J. Ciesielski. A Comprehensive Approach to the Partial Scan Problem Using Implicit State Enumeration. In *Proc. of the IEEE Int'l. Test Conf.*, pages 651–657, 1998.

[34] P. Kalla and M. J. Ciesielski. A Comprehensive Approach to the Partial Scan Problem Using Implicit State Enumeration. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 21(7):810–826, 2002.

[35] O. Khan, M. L. Bushnell, S. Devanathan, and V. D. Agrawal. SPARTAN: A Spectral and Information Theoretic Approach to Partial Scan. In *Proc. of the 16th IEEE North Atlantic Test Workshop (NATW)*, pages 19–25, May 2007.

[36] O. Khan, M. L. Bushnell, S. Devanathan, and V. D. Agrawal. SPARTAN: A Spectral and Information Theoretic Approach to Partial Scan. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 21.1.1–21.1.10, Oct. 2007.

[37] O. I. Khan. Spartan: A Spectral and Entropy-based Partial-scan and Test Point Insertion Algorithm, 2007. Ph.D. Diss., Electrical and Computer Engineering Dept., Rutgers University.

[38] K. Kim and C. Kime. Partial Scan by Use of Empirical Testability. In *Proc. of the IEEE Int'l. Conf. on Computer-Aided Design*, pages 314–317, 1990.

[39] B. Krishnamurthy. A Dynamic Programming Approach to the Test Point Insertion Problem. In *Proc. of the* 24^{th} *ACM/IEEE Design Automation Conf.*, pages 695–705, June, 1987.

[40] A. Kunzmann and H. J. Wunderlich. An Analytical Approach to the Partial Scan Design Problem. *J. of Electronic Testing: Theory and Applications*, 1(22):163–174, 1990.

[41] D. Lee and S. Reddy. On Determining Scan Flip-Flops in Partial-Scan Designs. In *Proc. of the Int'l. Conf. on Computer Aided Design*, pages 163–174, 1990.

[42] H. C. Liang and C. L. Lee. Effective Methodology for Mixed Scan and Reset Design Based on Test Generation and Structure of Sequential Circuits. In *Proc. of the 8th IEEE Asian Test Symp.*, pages 173–178, 1999.

[43] X. Lin, I. Pomeranz, and S. M. Reddy. Full Scan Fault Coverage with Partial Scan. In *Proc. of the IEEE Design Automation and Test in Europe (DATE) Conf.*, pages 468–472, 1999.

[44] R. Mester and U. Franke. Spectral Entropy-Activity Classification in Adaptive Transform Coding. *IEEE J. of Selected Areas in Communications*, 20, no. 5:913–917, June 1992.

[45] G. E. Moore. Cramming More Components onto Integrated Circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114. *Solid-State Circuits Newsletter, IEEE*, 20(3):33–35, Sept. 2006.

[46] F. Muradali and J. Rajski. A Self-Driven Test Structure for Pseudorandom Testing of Non-Scan Sequential Circuits. In *Proc. of the VLSI Test Symposium*, pages 17–25, 1996.

[47] S. Narayanan, R. Gupta, and M. A. Breuer. Optimal Configuring of Multiple Scan Chains. *IEEE Trans. on Computers*, 42(9):1121-1131, Sept. 1991.

[48] T. Niermann and J. Patel. HITEC: A Test Generation Package for Sequential Circuits. In *Proc. of the European Conf. on Design Automation (EDAC)*, pages 214–218, Feb. 1991.

[49] P. S. Parikh and M. Abramovici. Testability-Based Partial Scan Analysis. *J. of Electronic Testing: Theory and Applications*, 7(11):47–60, August 1995.

[50] I. Park, D. S. Ha, and G. Sim. A New Method for Partial Scan Design Based on Propagation and Justification Requirements of Faults. In *Proc. of the Int'l. Test Conf.*, pages 413–422, Oct. 1995.

[51] S. Park. A Partial Scan Design Unifying Structural Analysis and Testabilities. *Int'l. J. on Electronics*, 88(12):1237–1245, Dec. 2001.

[52] K. P. Parker and E. J. McCluskey. Probabilistic Treatment of General Combinational Networks. *IEEE Trans. on Computers*, 24(6):668–670, June 1975.

[53] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann. A Genetic Algorithm Framework for Test Generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(9):1034–1044, Sep 1997.

[54] G. S. Saund, M. S. Hsiao, and J. H. Patel. Partial Scan beyond Cycle Cutting. In *Proc. of the IEEE Int'l. Symp. on Fault-Tolerant Computing*, pages 320–328, 1997.

[55] B. Seiss, P. Trouborst, and M. Schulz. Test Point Insertion for Scan-Based BIST. In *Proc. of the European Test Conf.*, pages 253–262, April 1991.

[56] S. C. Seth and V. D. Agrawal. A New Model for Computation of Probabilistic Testability in Combinational Circuits. *Integration, the VLSI Journal*, 7(1):49–75, April 1989.

[57] R. Sethuram, S. Wang, S. T. Chakradhar, and M. L. Bushnell. Zero Cost Test Point Insertion Technique for Structured ASICs. In *Proc. of the Int'l. Conf. on VLSI Design*, pages 357–363, Jan. 2007.

[58] R. Sethuram, S. Wang, S. T. Chakradhar, and M. L. Bushnell. Zero Cost Test Point Insertion Technique to Reduce Test Volume and Test Generation Time for Structured ASICs. In *Proc. of the Asian Test Symp.*, pages 339–348, Nov. 2006.

[59] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(2):379–423 and 623–656, July and October 1948.

[60] S. Sharma and M. Hsiao. Combination of Structural and State Analysis for Partial Scan. In *Proc. of the Int'l. Conf. on VLSI Design*, pages 134–139, Jan. 2001.

[61] S. E. Tai and D. Bhattacharya. A Three Stage Partial Scan Design Method to Ease ATPG. *J. of Electronic Testing: Theory and Applications (JETTA)*, 7(11):95–104, Nov. 1995.

[62] N. Tamarapalli and J. Rajski. Constructive Multi-Phase Test Point Insertion for Scan-Based BIST. In *Proc. of the Int'l. Test Conference*, pages 649–658, Oct. 1996.

[63] R. E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, 1983.

[64] K. Thearling and J. Abraham. An Easily Computed Functional Level Testability Measure. In *Proc. of the Int'l. Test Conf.*, pages 381–390, Oct. 1989.

[65] N. Touba and E. J. McCluskey. Test Point Insertion Based on Path Tracing. In *Proc. of the VLSI Test Symp.*, pages 2–8, April, 1996.

[66] E. Trischler. Incomplete Scan Path with an Automatic Test Generation Methodology. In *Proc. of the Int'l. Test Conf.*, pages 153–162, Oct. 1980.

[67] H. C. Tsai, K. T. Cheng, C. J. Lin, and S. Bhawmik. An Efficient Test Point Selection for Scan-Based BIST. *IEEE Trans. on VLSI Systems*, 6(4):667–676, Dec. 1998.

[68] H. J. Wunderlich and S. Hellebrand. The Pseudo-Exhaustive Test of Sequential Circuits. In *Proc. of the Int'l. Test Conf.*, pages 19–27, Oct. 1989.

[69] D. Xiang and J. H. Patel. A Global Partial Scan Design Algorithm Using Circuit State Information. In *Proc. of the IEEE Int'l. Test Conf.*, pages 548–557, October 1996.

[70] D. Xiang and J. H. Patel. Partial-Scan Design Based on Circuit State Information and Functional Analysis. *IEEE Trans. on Computers*, 53(3):276-287, March 2004.

[71] D. Xiang, S. Venkataraman, W. K. Fuchs, and J.H. Patel. Partial Scan Design Based on Circuit State Information. In *Proc. of the ACM/IEEE Design Automation Conf.*, pages 807–812, June 1996.

[72] D. Xiang, Y. Xu, and H. Fujiwara. Non-Scan Design for Testability for Synchronous Sequential Circuits Based on Conflict Resolution. In *Proc. of the Int'l. Conf. on Computer Aided Design*, pages 400–403, November 1991.

[73] W. Yang and J. D. Gibson. Coefficient Rate and Significance Maps in Transform Coding. In *Record of the Thirty-First Asilomar Conf. on Signals, Systems and Computers*, volume 2, pages 1373–1377, November 1997.

[74] W. Yang and J. D. Gibson. Spectral Entropy, Equivalent Bandwidth and Minimum Coefficient Rate. In *Proc. of the Int'l. Symp. on Information Theory*, page 181, June-July 1997.

Appendix A
Entropy User Guide

The algorithm is invoked by the following command line:

entropy [–a *input_file_name*] [–d *file_name*] [–c *file_name*] [–l *file_name*] [–v] [–s *value*] [–t *value*] [–p *value*] [–w *value*] [–u *value*] [–f] [–r *file_name*] [–q *value*] [-k *beta*] [–h] [–g] [–z *value*] [–i *mode*] [–b *file_name*] [–o *file_name*] [–x *value*] [–y *value*]

The options in the tool are explained below:

Option	**Definition**
–a	Name of the *input_file_name*
–d	Name of the netlist *file_name* for ATPG, with the DFT hardware inserted.
–c	Name of the constraints *file_name*
–l	Name of the library *file_name*
–v	Tells the parser that the input is in verilog format
–s	Floating point *value* indicates the percentage of flip-flops to be scanned. The range is from 0.0 to 1.0 with a default value of 0.5.
–t	Integer *value* indicates the number of test points that can be inserted. *value* should not be greater than total the number of logic gates in the netlist and the default value is 50.

Option	**Definition**
–p	Floating point *value* indicates the percentage of total DFT sites that can be test points. The range is from 0.0 to 1.0 with a default value of 0.5.
–w	Integer *value* indicates the number of vectors that will be used to initialize the circuit by logic simulation. The default value is 51200.
–u	Integer *value* indicates the number of vectors that will be used to update the entropies of the circuit by logic simulation. The default value is 512.
–f	If present, causes all the flip-flops to be scanned and the tool to run in full-scan mode. Otherwise, the tool runs in partial-scan mode.
–r	Name of the results *file_name*. By default, the results are printed on *stdout*.
–q	Interger *value* to unbias the probability values from pseudo-random vector logic simulation. Unbiasing greatly improves the performance of the tool by reducing the number of test vectors.
–k	Interger *beta* to calculate the unbias for the probability values from pseudo-random vector logic simulation.
–h	If present, uses entropy gain as the testability measure.
–g	If present, operates in gradient descent mode, otherwise, operates in LP mode.
–z	Floating point *value* indicates the entropy threshold value, if all the gates in the circuit have higher entropy than this *value*, then the tool terminates. The range is from 0.0 to 1.0 and default is 1.0.
–i	*mode* indicates the spectral mode of operation.

Option	**Definition**
–b	LP data *file_name* that should be sent to the AMPL.
–o	LP output *file_name* that should be copied from the AMPL.
–x	Floating point *value* indicates the penalty of inserting a test point in the LP cost function. The range is from 0.0 to 1.0 and the default is 0.95.
–y	Floating point *value* indicates the penalty of scanning a flip-flop in the LP cost function. The range is from 0.0 to 1.0 and the default is 0.95.