# Test Pattern Compaction with Boolean Satisfiability Attack of Logic Locking

Yadi Zhong

# Test Pattern Compaction with Boolean Satisfiability Attack of Logic Locking

Yadi Zhong

***Abstract*—Test time per chip plays an essential role in manufacturing tests. Keeping a low number of test patterns becomes one of the prime objectives in concurrent with achieving the desired fault coverage. Unfortunately, finding an optimum set is a NP-hard problem. Today's commercial ATPG tools have significantly reduced the number of test patterns to achieve a high fault coverage. However, there is still a huge scope for reducing the total pattern count, equivalent to minimizing test costs in the production phase. In this paper, we propose two novel methods to lower the test pattern count to detect all stuck-at faults in a circuit with the same or higher fault coverage as in the commercial ATPG tool, *e.g.,* TetraMAX II. The first approach begins by applying a small set of random patterns to solve easy-to-detect faults. The remaining faults are detected by the SAT-based attack on logic locking with converting all the remaining faults into one locked circuit. Each stuck-at fault is modeled with its equivalent key gate. The second approach selects the first few patterns generated by the ATPG tool and applies the SAT attack of logic locking to determine the test patterns for detecting the undetected faults. By exploiting the overall linear iteration complexity and the exponential removal of incorrect key combinations per each SAT attack iteration, it is feasible to significantly reduce the total pattern count while maintaining the same or higher fault coverage as the ATPG counterpart. We demonstrate the effectiveness of both approaches and show that we are able to achieve more compact test pattern set compared to commercial ATPG tool.**

***Index Terms*—Automatic test pattern generation (ATPG), test pattern compaction, Boolean satisfiability (SAT), logic locking.**

## I. Introduction

Integrated circuits (ICs) are everywhere and they are at the heart of every technology ranging from commercial, industrial, and Department of Defense (DoD) spaces. Semiconductor manufacturing is a booming industry and it is projected to become the trillion-dollar asset by 2030 [1]. Innovations in aggressive technology scaling and drastic shrinking of device geometries have propelled the complex fabrication, packaging, and test processes. As demand for more computational capabilities and capacity for electronics grow, *i.e.,* more transistors in a single chip or package, it is increasingly important for aggressive testing to reduce manufacturing defect escape to a minimum. For chips deployed in critical infrastructures, achieving high-quality tests through IC, package and system-level testing are crucial for the reliability of systems. To ensure the reliability of ICs, a desirable fault coverage is required during test pattern generation process [2]. Besides high fault coverage, the total test patterns used for attaining the specified fault coverage is an important factor to support massive production. For targeting the expected manufacturing quality, the number of test vectors to meet the quality assurance of a device can influence the production of such products as the manufacturing test applies all test vectors on every chip it produced. Optimizing test compaction and minimizing test pattern counts before the production phase will reduce test time bottleneck. Today's commercial automatic test pattern generation (ATPG) tools have significantly reduced the number of test patterns needed to detect faults with very high fault coverage. However, it is agnostic to the tool itself if it has found the minimum set of test vectors to detect all or the maximum number of faults in the complete fault list. It is common knowledge that finding the minimum pattern count is difficult and in NP-hard in computation complexity. Nevertheless, it is possible and beneficial to further shrink test counts, getting one step closer to the computationally-hard optimal solution for a given circuit.

Over the past few decades, we have witnessed a steady increase in fault coverage of digital circuits due to the novel techniques developed in combinational and sequential ATPG techniques, *i.e.,* D-Algorithm [3], [4], PODEM [5], FAN [6], SOCRATES [7], TRAN [8], *etc*. Along with these test pattern generation techniques, SAT-based solutions targeting high fault coverage have also been proposed [9]–[13]. In addition, recent progress of SAT-based techniques examines test compactions [14]–[18]. Concurrently, SAT has been actively applied as effective attacks in breaking various logic locking methods of the past decade. The effectiveness and efficiency of SAT attack [19] rendered the earlier locking schemes of insufficient security against oracle-guided adversaries. Since then, multiple works have focused on adapting SAT attack to exploit previously SAT-resistant designs, including recent study on complexity analysis of SAT attack and how it can be constructively applied for test pattern generation [20]–[22].

In this paper, we build upon the prior work in test pattern generations, which achieved 100% fault coverage with SAT attack of logic locking, and optimize the SAT attack-based method for test pattern compaction. Our goal is to minimize the overall test pattern count as it will reduce the accumulated test time bottleneck after batch manufacturing. We proposed two SAT attack-based approaches to optimize the pattern set compared to commercial ATPG tool counterpart. The approach I leverages random pattern generation to first solve easy-to-detect faults before using SAT attack to determine the patterns for the remaining faults. Random patterns are determined based on $(i)$ maximizing entropy reduction and $(ii)$ minimizing the selected patterns from a set of randomly generated set beforehand. Approach II targets improved test compaction from commercial ATPG tools. Patterns derived from ATPG are partially included while the SAT attack is applied to

solve the rest of the undetected faults. Both approaches have been demonstrated using ISCAS'85 benchmarks and they outperforms TetraMAX II in minimizing test count with c6288 having the highest test set reduction percentage.

The contributions of this paper are described as follows:

- *SAT attack-based test compaction:* We propose two novel test pattern reduction methods using SAT attack for logic locking. The first approach combines both random pattern generation and SAT attack-based method to complete the test set with 100% fault coverage. The random patterns are optimized with the proposed entropy reduction approach by discarding patterns with low impact of fault coverage increment through adaptive threshold. SAT attack-based pattern generation kicks in after optimized random patterns are chosen. The second approach improve test compaction of TetraMAX II directly by SAT attack-based method. Test pattern generation with SAT attack returns more compact pattern subset than ATPG tool for reaching fault coverage of 100% due to the observed linear attack complexity shown in logic locking.
- *Better Test Reduction than TetraMAX II:* Overall, both approaches needs fewer patterns than commercial ATPG tool TetraMAX II to achieve 100% fault coverage. We achieved an average test pattern count reduction of 14.97% (Approach I) and 23.66% (Approach II), and the maximum reduction of 37.68% (Approach I) and 43.48% (Approach II) compared to TetraMAX II. We believe our work provides insight for minimizing test set through maximizing the efficiency of SAT attack against logic locking with average linear attack complexity.

The rest of the paper is organized as follows: An overview of SAT-based attack on logic locking, test pattern generation, and SAT-based test compaction techniques are provided in Section II. The two proposed SAT-attack-based approaches for test compaction are described in Section III. We present the results for the implementation of the proposed methods with analysis on ISCAS'85 benchmarks in section IV. Finally, we conclude our paper in Section V.

## II. Background

This section presents the prior work of test pattern generation and test compaction techniques using Boolean satisfiability (SAT). We describe briefly the overview of logic locking and SAT attack, and the recent adaption of SAT attack for test pattern generation of hard-to-detect faults [22].

### A. SAT-based Test Pattern Generation and Test Compaction

Boolean satisfiability (SAT) was first explored for test pattern generation by Larrabee [9] and Stephan et al. [10], where a fault-free circuit and a faulty circuit with stuck-at fault are XORed at the output for deriving a test. With the advances of SAT solvers, the SAT solving time got expedited. Since early 2000s, multiple test generation and test compaction techniques have considered SAT to achieve higher fault coverage, and/or more compact test sets [11]–[13], [22]. Fujita et al. [12] have observed minimal test set increase for multiple stuck-at faults from patterns of single stuck-at faults. Their approach, however, considers faults occurred only at the output of gates, which includes a limited locations for faults. Zhong et al. [22] apply SAT attack for test generation of hard-to-detect faults and redundant faults from commercial ATPG tool. It also adopts the more generic notion which faults can appear at any line, fanout or segment of the circuit.

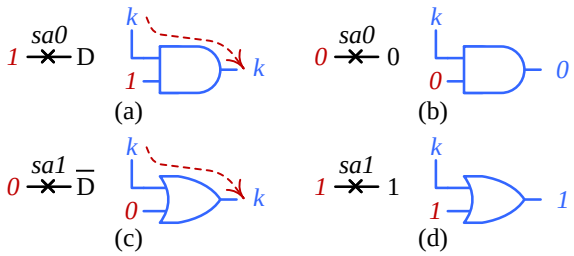


Figure 1: Logic locking-based modeling of a *saf*: transform a *sa0* to AND key gate ($k = 1$), (a) successful propagation of key $k$ with logic 1, (b) failed propagation of $k$ with 0; a *sa1* to OR key gate ($k = 0$), (c) successful propagation of $k$ with logic 0, and (d) failed propagation of $k$ with 1. [22]

SAT-based test compaction has also been evolving in the past two decades [14]–[18]. Eggersglüß et al. [14] focused on increasing the unspecified bits in patterns generated from SAT solvers, where post-processor identifies input bits necessary to make fault observable. However, this post-processing step has the multiplicative complexity for both circuit gate count and number of primary outputs. PASSAT 2.0 [17], a multi-functional testing framework based on SAT, apply both static and dynamic test compaction as well as fault simulation, but its test set size for c5315 and c7552 of ISCAS'85 is still large and take significantly more patterns than our proposed approaches (see Section IV, Table I). Eggersglüß et al. [18] uses both external and internal necessary assignments for static compaction of test sets. It works on pre-computed deterministic test set for industrial circuit, however, neither sets nor circuits are available for comparison.

### B. SAT attack in Logic Locking and Test Pattern Generation

Proposed by Subramanyan et al. [19], SAT attack has been both effective and efficient against logic locking available by then. The attack correctly derives the secret key by constructing a miter of two locked circuit, where a distinguishing input pattern is determined in each iteration and the corresponding oracle response helps the SAT solver "learns" additional information about the correct key. Zhong et al. [21] showed linear iteration complexity of SAT attack breaking single logic cones of secure logic locking with reconvergent fanout.

Besides, the same SAT attack is demonstrated effective in finding test sets for hard-to-detect faults from commercial ATPG tool. Zhong et al. [22] applied the same SAT attack to achieve 100% fault coverage for test pattern generation. Each hard-to-detect fault is converted to an equivalent locked gate. Figure 1(a,b) shows the modeling of *sa1* fault with AND key gate with key $k = 1$, where SAT attack's propagation of key value to the miter output only when logic 1 is placed at the AND's input. Logic 1 is also the value to make *sa1* fault observable. The *sa0* fault is modeled as OR gate with

$k = 0$. Again, OR gate ensures the identical conditions for key propagation of SAT attack and fault observability in VLSI testing, as in Figure 1(c,d). This transformation of fault to key gate preserves controllability and observability conditions for generating test patterns so that SAT attack's distinguishing input patterns are essentially tests for the stuck-at faults.

## III. Proposed SAT Attack-based Approaches for Test Compaction

The proposed approaches target test pattern generation of stuck-at faults, where any line in the circuit could have stuck-at 0 (*sa0*), stuck-at 1 (*sa1*) faults. The goal is to find better optimized test pattern set with lower test count while maintaining high fault coverage. We assume that the stuck-at faults in a circuit are collapsed faults, which is commonly used during pattern generations in various ATPG tools.

### A. Complexity Equivalence of Set-Covering Problem

Let us first look at the NP-hard set-covering problem [23], [24] and how test compaction is a natural fit into set-covering.

**Definition 1.** *Given a finite set $\mathcal{F}$ of elements $\{f_i\}$ and a family $\mathfrak{S}$ of non-empty subsets of $\mathcal{F}$, there exists element $f_i \in \mathcal{F}$ which belongs to one or more subsets of family $\mathfrak{S}$, $f_i \in \mathcal{S}_j \in \mathfrak{S}$; and $\mathcal{F}$ can be represented as combining all elements from every set $\mathcal{S}_j$ of $\mathfrak{S}$,*

$$\mathcal{F} = \bigcup_{\mathcal{S}_j \in \mathfrak{S}} \mathcal{S}_j$$

*The set-covering problem of $(\mathcal{F}, \mathfrak{S})$ is defined as the minimum-size subset $\mathfrak{M} \subseteq \mathfrak{S}$ whose members contain all elements in $\mathcal{F}$:*

$$\mathcal{F} = \bigcup_{\mathcal{S}_j \in \mathfrak{M}} \mathcal{S}_j$$

It is straightforward that finding the fewest subsets from $\mathfrak{S}$ to reconstruct set $\mathcal{F}$ is combinatorial problem in the worst case, while its decision problem, set-covering, is NP-hard. From VLSI testing perspective, we consider all stuck-at faults in a circuit as elements $f_i$ in finite set $\mathcal{F}$ since total faults is linear to the lines in the given circuit. A test pattern $p_j$, on the other hand, can detect one or more faults in $\mathcal{F}$, and the collection of these faults $\mathcal{S}_j$ is a subset of $\mathcal{F}$, where different test patterns form the set family $\mathfrak{S}$. Please note that test patterns $p_j$s are, in practice, fewer than all possible primary input patterns, which are subject to exponential increase with input size.

Centered around test compaction, we ask the optimization questions as follows: *how to (i) obtain these test patterns $\{p_j\}$, and (ii) heuristically optimize test compaction from $\{p_j\}$ such that it needs fewer patterns, a step closer to optimal solution, while fault coverage are kept same (or higher)?*

### B. Proposed Approach I

Proposed approach I consider first leveraging random pattern generation (RPG) to solve easy-to-detect faults and let the remaining faults to be determined by SAT attack-based technique. However, not all random patterns contribute equally to the fault coverage. Random pattern testing exhibits a logarithmic growth of fault coverage, where huge number of patterns are required as coverage growth diminish towards 100% [2]. Therefore, blindly accepting all random patterns is far from a solution for test compaction.

In order to maximize benefit from random pattern generation with its initial fast-rise fault coverage, we selectively choose patterns from randomly generated ones with greedy heuristic and adaptive thresholding (P1-Opt. RPG):

1) *Intermediate fault coverage per pattern:* Pre-compute $n$ random patterns $(p_1, ..., p_n)$ for a given circuit with a pseudo-random number generator. Patterns $p_1, ...p_n$ are simulated on a one-by-one manner through fault simulator. The intermediate fault coverage after running pattern $p_j$, $j = 1, ...n$, is recorded. Note that the order of applying these $n$ patterns affects the intermediate fault coverage except the last one as some fault may be detected earlier or later during fault simulation. In this method, we do not considering reordering $p_1, ..., p_n$ for optimization as it is combinatorial problem with nonlinear complexity.
2) *Greedy heuristic:* Finding the optimal threshold is equivalent to the previously discussed set-covering problem with set $\mathcal{F}$ as the faults covered by $p_1, ..., p_n$. However, here we consider a variant of set-covering problem, namely, the minimum-size subset of $\mathfrak{S}$ including the majority of elements in $\mathcal{F}$, but not all. We apply the greedy heuristic approach for choosing patterns giving the large stuck-at fault coverage increase with the given fault simulation result of previous step. We leave the rest of undetected faults for SAT attack-based approach in latter step. However, the boundary for deciding which patterns to keep while others to drop need to be determined on a circuit-by-circuit basis.
3) *Adaptive thresholding:* Due to the probabilistic nature of random patterns, the same fault may have already been detected by an earlier pattern, and thus fault coverage increases non-uniformly. We compute the delta $\Delta_j$ fault coverage between pattern $p_{j-1}$ and $p_j$, $j \geq 1$. All $\Delta_j$ values are sorted in descending order. By excluding the outliers, we compute the mean $\mu$, median $\eta$, and standard deviation $\sigma$ of fault increments $\Delta_j$. The threshold for selecting patterns from this pre-computed set is based on $(i)$ maximizing entropy reduction (or undetected faults) and $(ii)$ minimizing the selected patterns from randomly generated set beforehand.

$$\mathsf{threshold} = \mu + \mathsf{c}_0 \cdot \eta - \mathsf{c}_1 \cdot \sigma,$$

   where $c_0$ and $c_1$ are constants for tuning the threshold which are partially influenced by removed outliers.
4) *Optimized set of random patterns:* Patterns with $\Delta_j$ increment exceeding the threshold is selected and others with minimal contributions to fault coverage increase are discarded. We rerun fault simulation on the selected patterns to obtain fault coverage, both complete and incremental, along with a list of undetected faults.

The remaining undetected faults may still have some easy-to-detect fault excluded during optimizing random pattern set, but the majority faults should be random-pattern-resistant

faults. SAT-based ATPGs outperform classical ATPG in generating tests for hard-to-detect faults and identifying redundant faults [14], [22]. In addition, [22] showed mapping multiple faults to a single locked circuit also provided pattern reduction. We apply SAT attack-based approach to generate the remaining test set for faults undetected by (P1-Opt. RPG), denoted as P1–SAT. Note that faults are grouped into a locked circuit (Second approach in [22]), where each fault is transformed to its SAT-attack-based key gate equivalence as in Figure 1. The details are explained in Algorithm 1 of Section III-C

### C. Proposed Approach II

The proposed approach II improves test compaction of TetraMAX II by partially removing tests obtained by it with SAT attack-based method. The rationale behind replacing patterns of TetraMAX II with SAT attack-based ones are due to the observed linear attack complexity for SAT attack breaking logic locking designs. We include patterns determined from commercial ATPG tool and neglected the rest of the patterns based on a fault coverage cutoff percentage. We denote this approach as ATPG + P2–SAT. Our goal in this approach is to demonstrate the effectiveness of SAT attack on generating compact test sets on hard-to-detect faults (or faults in general) than TetraMAX II's compact set. Considering the preliminary results in [22] of tests reduction among hard-to-detect faults, it is feasible to reduce pattern counts while achieving equivalent fault coverage than ATPG tool given the overall linear iteration complexity and exponential removal of incorrect key combinations of SAT attack for every iteration.

Algorithm 1 shows SAT attack-based test compaction of ATPG + P2–SAT. Since P1–SAT and ATPG + P2–SAT are similar in regard to SAT attack-based test compaction except with different input fault list, we also include description of P1–SAT (Lines 15, 18-36) and consider it as a sub-function/procedure inside ATPG + P2–SAT (Lines 1-17). Algorithm first initialize fault coverage list $FC$, ATPG pattern list $L$, trimmed list $L_{tr}$, undetected fault list $UF$, SAT attack generated patterns $L_{sat}$, and final test set $P_2$ to empty sets (Line 2). First, netlist $C_N$ and standard cell library are loaded into ATPG; TetraMAX II is set with all stuck-at faults, maximum abort limit, and high merge effort for test pattern generation (Lines 3-5). ATPG return faults are stored in list $L$ (Line 6) and incremental fault coverage $FC_j$ is recorded (Lines 7-10) since fault simulation is accumulative by removing faults once detected. With a given cutoff percentage $co$, only patterns contributed to reaching fault coverage $co$ are included while the rest is discarded, and are saved in trimmed list $L_{tr}$ (Line 11). Fault simulation is again invoked with entire set $L_{tr}$ to find out the remaining undetected faults $UF$. Both netlist $C_N$ and faults $UF$ are passed to function P1–SAT to obtain SAT attack generated pattern set $L_{sat}$ (Lines 12-15). The complete test set for approach II is known by combining the trimmed set from ATPG $L_{tr}$ and SAT attack-based $L_{sat}$, and this test compaction set is return to the user (Line 16).

Function P1–SAT accepts netlist $C_N$ and to-be-detected fault list $UF$ as inputs and begins with initializing SAT attack results $SP$, final pattern set $Lsat$, and converting $C_N$ to *bench* file $C_{BN}$ (Lines 18-19). SAT attack-based test compaction is performed dynamically, where all faults in $UF$ are converted to their equivalent locked gates with keys (Line 21). SAT attack runs on technology-independent locked (*.bench*) circuit (Line 22) to generate distinguishing input patterns $SP$ and key $k$ (Line 22). We invoke fault simulation on each of the pattern $sp$ in set $SP$ and select pattern $sp_{max}$ producing the largest fault coverage increase $fc_{max}$ (Lines 24-31). Fault list with respect to this $sp_{max}$ is also recorded in Line 29. Pattern $sp_{max}$ is included in the final SAT pattern set $L_{sat}$, and the remaining faults $UF$ are updated based on it (Lines 29, 32-33). This selection of $sp_{max}$ maximizing the current choice of fault coverage $fc_{max}$ is essentially a greedy approximation for finding the compact test set, equivalent to greedy heuristic approaches in solving set-covering problem. This process

**Algorithm 1:** SAT attack-based test compactions.

**Input :** Synthesized circuit (.v) ($C_N$), standard cell library ($stdlib$), cutoff percentage $co$.
**Output:** Test pattern sets $P_2$.

```
1  function ATPG + P2–SAT (D) is
2      FC, L, L_tr, UF, L_sat, P_2 ← ∅ ;
3      loadATPG(C_N, stdlib) ; addFaults(sa0, sa1, 'all') ;
4      setATPG(AbortLimit = max, Merge = high);
5      runATPG ;
6      L ← reportPatterns() ;
7      addFaults(sa0, sa1, 'all') ;
8      foreach (element p_j ∈ L) do
9          FC_j ← faultSim(p_j) ;
10     end
11     L_tr ← removeByCutoff(L, FC, co) ;
12     addFaults(sa0, sa1, 'all') ;
13     faultSim(L_tr) ;
14     UF ← reportFault(undetected) ;
15     L_sat ← P1–SAT(C_N, UF) ;
16     Return P_2 ← {L_tr, L_sat} ;
17 end
18 function P1–SAT (C_N, UF) is
19     SP, L_sat ← ∅ ; C_BN ← toBench(C_N) ;
20     while UF ≠ ∅ do
21         C_L ← faultModeling(C_DN, f, [22]) ;
22         C_BL ← toBench(C_L) ;
23         [SP, k] ← SAT-attack(C_BL, C_BN) ;
24         fc_max, sp_max ← 0 ; U ← ∅ ;
25         foreach (element sp ∈ SP) do
26             fc ← faultSim(UF, sp) ;
27             if fc > fc_max then
28                 fc_max ← fc ; sp_max ← sp ;
29                 U ← reportFault(undetected) ;
30             end
31         end
32         L_sat ← append(L_sat, sp_max) ;
33         UF ← U ;
34     end
35     Return L_sat ;
36 end
```

(Lines 19-33) of picking one pattern producing the maximum fault coverage in a SAT attack run is repeated until all faults in $UF$ is solved. Test set $L_{sat}$ produced with SAT attack is returned in the end (Line 35).

## IV. Experimental Result

The proposed methods dynamically select one pattern giving the maximum delta increase of fault coverage after every SAT attack run. To evaluate the proposed approaches, our implementation is developed around the original SAT attack paper by Subramanyan et al. [19]. Both approaches are analyzed using ISCAS'85 benchmarks. Synopsys 32nm SAED32 library is used for circuit synthesis and for technology-dependent library for TetraMAX II ATPG [25]. Any advanced technology nodes can be applied as SAT attack requires technology-independent *bench* file. Conversion of *bench* file from standard cell library is enabled for locking of fault-equivalent gates. For comparison purposes, we also run TetraMAX II on all ISCAS'85 benchmark and record pattern set of each circuit. Total number of collapsed faults (TC), test pattern count (C) generated by TetraMAX II, and the corresponding fault coverage (FC) for each benchmark are recorded in Columns 2-4 of Table I. Note that after obtaining the TetraMAX II's pattern set, we rerun the fault simulation with all test patterns one by one from the set to collect the incremental fault coverage (see Figures 3 and 5 for fault coverage curves of TetraMAX II).

For our proposed approach I, we first randomly generate 100 input patterns for each circuit, and then obtained the incremental fault coverage for each from fault simulation. The delta $\Delta_j$ increment is the fault coverage difference before and after applying pattern $p_j$ to the simulator. Then, all fault increment $\Delta_j$s, excluding two of the largest and smallest values (four outliers), are sorted in descending order, as shown in Figure 2, where mean, median, and standard deviation of each $\{\Delta_j\}$ ensemble of ISCAS'85 circuits is included. Optimized random patterns (P1-Opt. RPG) is selected with $c_0, c_1 = 0$ due to median $\eta \approx 0$ and standard deviation $\sigma \gg \mu$. The remaining undetected faults are solved by SAT attack-based method (P1–SAT) as described in Section III-B. Figure 4 shows proposed approach I (P1-Opt. RPG + P1–SAT) in comparison with the initial 100 random patterns (green curve) using benchmarks c1908 and c7552 as examples. For c1908, 20 out of 100 random patterns are selected as P1-Opt. RPG (orange curve), along with 47 patterns determined by SAT attack (P1–SAT) (purple curve). For c7552 benchmark, proposed approach I determines 19 optimized random patterns (P1-Opt. RPG) with 52 patterns from SAT attack (P1–SAT). Figure 5 visualizes the fault coverage of approach I with P1-Opt. RPG (orange curve) and P1–SAT (purple) with TetraMAX II (blue) as reference, where all benchmarks except c880 outperform TetraMAX II. TetraMAX II has 1 fewer patterns than approach I. Table I reports the summary of approach I on ISCAS'85, where the optimized random pattern count ($C_R$), SAT attack-based patterns ($C_{S1}$), total pattern count ($C_{P1}$), fault coverage (FC), accumulated SAT attack running time ($t_{SAT}$), and the test set reduction (%) compared with commercial ATPG (Column 3)

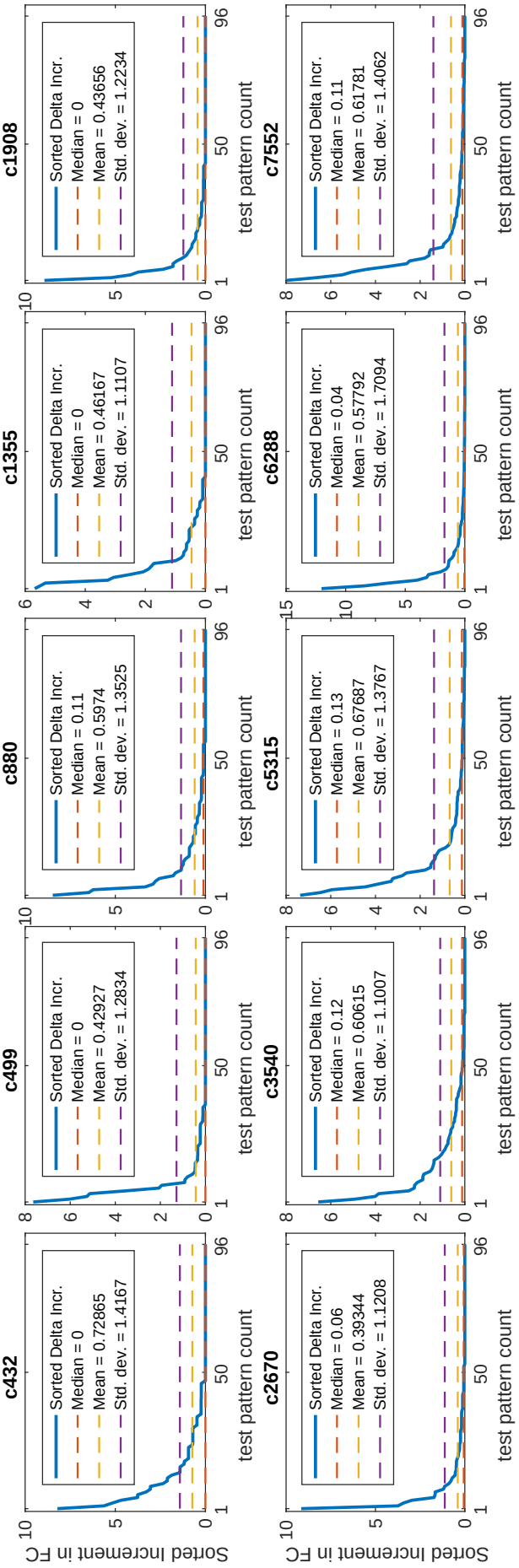


Figure 2: Sorted $\Delta_j$ increment with mean, median, and standard deviation in random pattern generation after removing outliers on ISCAS'85.

Table I: Summary of test compaction for TetraMAX II, proposed Approach 1 (P1-Opt. RPG + P1–SAT), and Approach 2 (ATPG + P2–SAT) on ISCAS'85 benchmarks. Total pattern count ($C_{P1} = C_R + C_{S1}$, $C_{P2} = C_T + C_{S2}$), SAT attack time ($t_{SAT}$), and pattern set reduction percentage are summarized for both proposed approaches.

| Bench-mark | TF | ATPG | | Proposed Approach I | | | | | | Proposed Approach II | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | C | FC | $C_R$ | $C_{S1}$ | $C_{P1}$ | FC | $t_{SAT}$ | Reduct. | $C_T$ | $C_{S2}$ | $C_{P2}$ | FC | $t_{SAT}$ | Reduct. |
| c432 | 1026 | 53 | 99% | 25 | 19 | 44 | 100% | 0.69s | 16.98% | 20 | 22 | 42 | 100% | 1.24s | 20.75% |
| c499 | 1304 | 75 | 100% | 17 | 37 | 54 | 100% | 2.35s | 28.00% | 15 | 37 | 52 | 100% | 5,45s | 30.67% |
| c880 | 1872 | 46 | 100% | 24 | 23 | 47 | 100% | 2.36s | -2.17% | 10 | 25 | 35 | 100% | 5.57s | 23.91% |
| c1355 | 2172 | 118 | 100% | 24 | 62 | 86 | 100% | 7.81s | 27.12% | 16 | 66 | 82 | 100% | 19.78s | 30.51% |
| c1908 | 1544 | 72 | 100% | 20 | 47 | 67 | 100% | 6.93s | 6.94% | 15 | 44 | 59 | 100% | 9.58s | 18.06% |
| c2670 | 3328 | 67 | 100% | 17 | 48 | 65 | 100% | Ab(5)+38.29s | 2.99% | 9 | 46 | 55 | 100% | 31.93s | 17.91% |
| c3540 | 5606 | 126 | 100% | 29 | 78 | 107 | 100% | 96.64s | 15.08% | 35 | 79 | 114 | 100% | 108.83s | 9.52% |
| c5315 | 7956 | 63 | 100% | 21 | 39 | 60 | 100% | 36.97s | 4.76% | 10 | 43 | 53 | 100% | 65.48s | 15.87% |
| c6288 | 14928 | 69 | 100% | 18 | 25 | 43 | 100% | Ab(24) | 37.68% | 10 | 29 | 39 | 100% | Ab(29) | 43.48% |
| c7552 | 7352 | 81 | 100% | 19 | 52 | 71 | 100% | 96.86s | 12.35% | 8 | 52 | 60 | 100% | 94.67s | 25.93% |

are listed in Columns 5-10. Note that we set the limit of SAT attack duration to 1 minute before aborting the attack program and analyzing those distinguishing input patterns generated during the 1-min interval, as described in Section III-B. For benchmarks c2670 and c6288, which have inherent complexity for SAT solvers [19], [21], each of them have 5 aborts (Ab(5)) and 24 aborts (Ab(24)), respectively. The abort time is not converted to seconds but is listed explicitly in the SAT attack running time $t_{SAT}$, as shown in (Ab(n)) for $n$ aborts in Table I. With proposed approach I, we can observe an average of 14.97% test set reduction and a maximum of 37.68% reduction (on c6288) on ISCAS'85 benchmarks than TetraMAX II.

Proposed approach II reuses a few patterns from TetraMAX II before asking SAT attack to solve the remaining faults. In our experiment, we selected the initial patterns returned by TetraMAX II that reach fault coverage threshold of 80% (about the same level as RPG achieves). We examine how many patterns the proposed approach II requires to fill the final 20% fault coverage compared with TetraMAX II's remaining patterns. Figure 5 visualizes the fault coverage of approach II (orange curve) and TetraMAX II (blue curve), where all benchmarks outperform TetraMAX II. The details of pattern count are summarized in Table I. Columns 11-16 comprise of selected TetraMAX II pattern count ($C_T$), SAT attack-based patterns ($C_{S2}$), total pattern count ($C_{P2}$), fault coverage (FC), SAT attack total running time ($t_{SAT}$), and test set reduction (%) compared to TetraMAX II (Column 3), respectively. For example, approach II on c499 needs only 37 patterns ($C_{S2}$) to reach 100% fault coverage while TetraMAX II would require $C - C_T = 75 - 15 = 60$ additional tests. Circuit c6288 takes $C_{S2} = 29$ patterns to fill the final 20% fault coverage than $C - C_T = 69 - 10 = 59$ for TetraMAX II, where all 29 tests are generated from 1-min time limit of SAT attack (Ab(29)). For proposed approach II, we obtain an average of 23.66% test set reduction and a maximum of 43.48% reduction than TetraMAX II. In both approaches, the multiplier benchmark (c6288) has the largest test reduction. Both approaches offer further optimized test compaction from commercial ATPG tool, *i.e.,* TetraMAX II, and we emphasize the application of SAT attack achieve this test set reduction.

## V. Conclusion

In this paper, we proposed two novel techniques to optimize test compaction utilizing the efficiency in Boolean Satisfiability attack on logic locking. All stuck-at faults of a circuit are transformed to their equivalent locked gates. As test compaction problem is equivalent to NP-hard set-covering problem, both approaches consider the greedy approximation algorithm which allows finding a solution within the logarithmic approximation of the optimal solution. Our methods are validated on ISCAS'85 benchmarks, demonstrating better test compaction than commercial ATPG counterpart. We achieved an average test pattern count reduction of 14.97% (Approach I) and 23.66% (Approach II), and the maximum reduction of 37.68% (Approach I) and 43.48% (Approach II) compared to TetraMAX II. We believe our work offers additional insight for minimizing total test patterns with the average linear complexity of SAT attack on logic locking. Our future work will focus on test compaction of large industrial circuits and improving both strategies with more time efficiency.

## References


[1] Semiconductors and the Semiconductor Industry, prepared by Congressional Research Service, April 19, 2023. https://crsreports.congress.gov/product/pdf/r/r47508.

[2] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, vol. 17. Springer Science & Business Media, 2004.

[3] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM journal of Research and Development*, pp. 278–291, 1966.

[4] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. on Electronic Comp.*, pp. 567–580, 1967.

[5] P. Goel, "An Implicit Enumeration Algorithm to Generate," *IEEE Transactions on Computers*, vol. 30, no. 3, 1981.

[6] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Transactions on Computers*, vol. 32, no. 12, pp. 1137–1144, 1983.

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

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

[9] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, 1992.

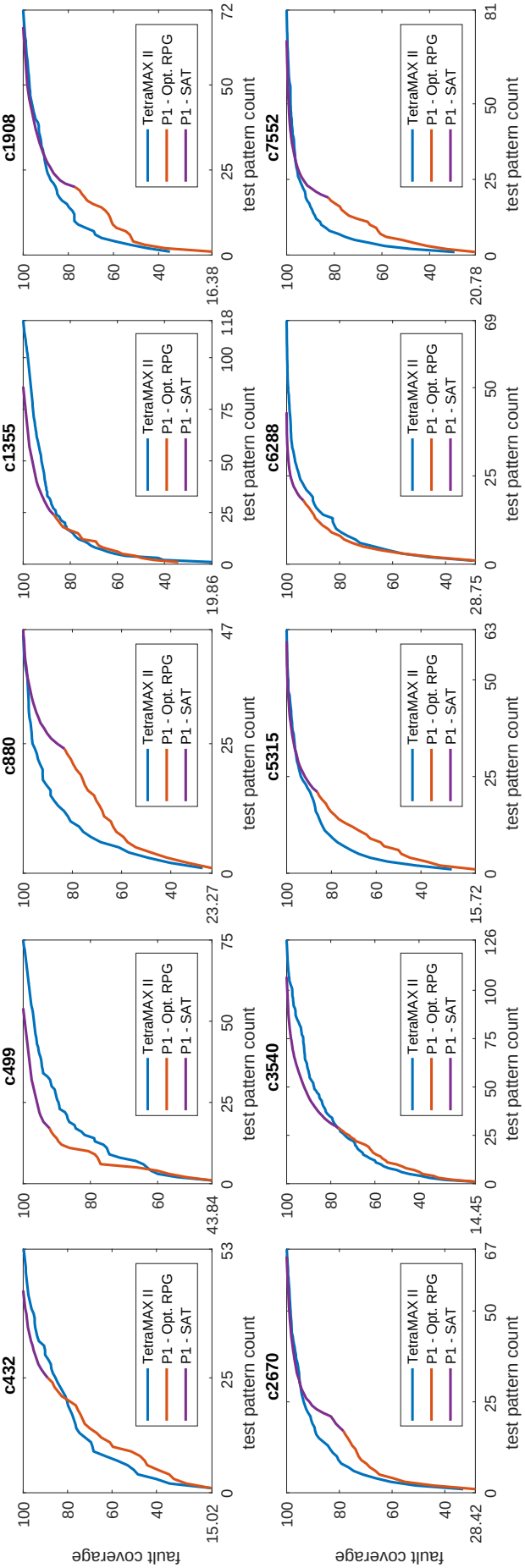


Figure 3: Fault coverage comparison between TetraMAX II and the proposed test compaction approach I on ISCAS'85 benchmarks.

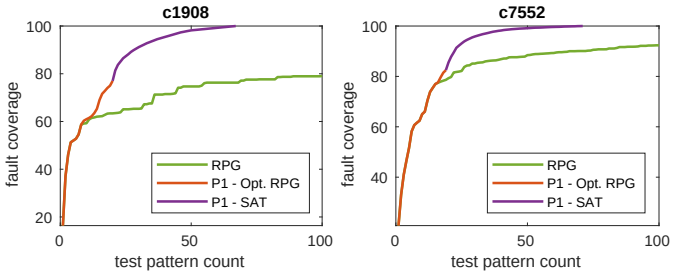


Figure 4: Comparison between unoptimized random pattern generation (RPG) and the proposed Approach I (P1-Opt. RPG + P1–SAT) for c1908 and c7552 benchmarks.


[10] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Combinational Test Generation Using Satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1167–1176, 1996.

[11] R. Drechsler, S. Eggergluss, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille, "On Acceleration of SAT-Based ATPG for Industrial Designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1329–1333, 2008.

[12] M. Fujita and A. Mishchenko, "Efficient SAT-based ATPG techniques for all multiple stuck-at faults," in *2014 International Test Conference*, pp. 1–10, IEEE, 2014.

[13] J. Balcarek, P. Fiser, and J. Schmidt, "Techniques for sat-based constrained test pattern generation," *Microprocessors and Microsystems*, vol. 37, no. 2, pp. 185–195, 2013.

[14] S. Eggersgluss and R. Drechsler, "Improving test pattern compactness in SAT-based ATPG," in *Asian Test Symposium (ATS)*, pp. 445–452, 2007.

[15] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved SAT-based ATPG: More constraints, better compaction," in *International Conference on Computer-Aided Design (ICCAD)*, pp. 85–90, 2013.

[16] S. Eggersglüß, R. Krenz-Bååth, A. Glowatz, F. Hapke, and R. Drechsler, "A new SAT-based ATPG for generating highly compacted test sets," in *15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 230–235, 2012.

[17] R. Drechsler, M. Diepenbeck, S. Eggersglüß, and R. Wille, "Passat 2.0: A multi-functional sat-based testing framework," in *2013 14th Latin American Test Workshop-LATW*, pp. 1–1, IEEE, 2013.

[18] S. Eggersglüß, S. Milewski, J. Rajski, and J. Tyszer, "A New Static Compaction of Deterministic Test Sets," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 4, pp. 411–420, 2023.

[19] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 137–143, 2015.

[20] H. M. Kamali, K. Z. Azar, F. Farahmandi, and M. Tehranipoor, "Advances in Logic Locking: Past, Present, and Prospects," *Cryptology ePrint Archive*, 2022.

[21] Y. Zhong and U. Guin, "Complexity Analysis of the SAT Attack on Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits (TCAD)*, pp. 1–14, 2023.

[22] Y. Zhong and U. Guin, "A Comprehensive Test Pattern Generation Approach Exploiting SAT Attack for Logic Locking," *IEEE Transactions on Computers*, pp. 1–13, 2023.

[23] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. Computer science, MIT Press, 2009.

[24] P. F. Flores, H. C. Neto, and J. P. Marques-Silva, "On applying set covering models to test set compaction," in *Proceedings Ninth Great Lakes Symposium on VLSI*, pp. 8–11, 1999.

[25] Synopsys Inc., Mountain View, CA, USA, "TetraMAX II ATPG: Automatic Test Pattern Generation," 2017.

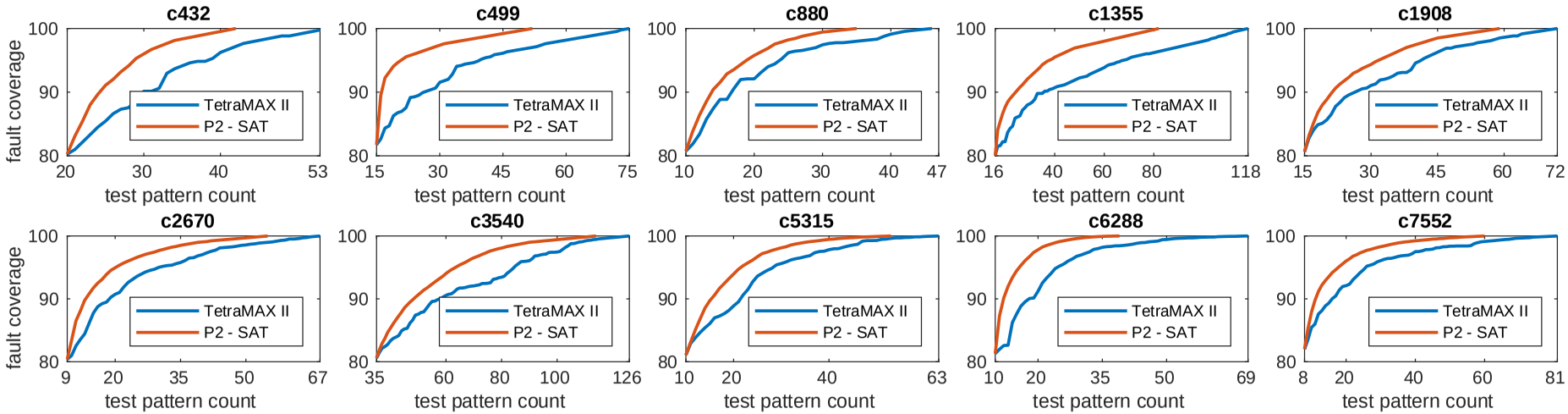


Figure 5: Fault coverage comparison between TetraMAX II and the proposed test compaction approach II on ISCAS'85 benchmarks.