Why is ATPG easy?*

Mukul R. Prasad Philip Chong Kurt Keutzer
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720
{mukul,pchong,keutzer}@eecs.berkeley.edu

Abstract

Empirical observation shows that practically encountered instances of ATPG are efficiently solvable. However, it has been known for more than two decades that ATPG is an NP-complete problem. This work is one of the first attempts to reconcile these seemingly disparate results. We introduce the concept of circuit cut-width and characterize the complexity of ATPG in terms of this property. We provide theoretical and empirical results to argue that an interestingly large class of practical circuits have cut-width characteristics which ensure a provably efficient solution of ATPG on them.

1 Introduction

Automatic test pattern generation (ATPG) techniques find widespread use in a number of CAD applications. In addition to the important task of generating test patterns for testing digital hardware, for which they were originally proposed, they have found application in verification techniques [3, 17] and logic optimization [6, 9]. It has been known for more than two decades that the ATPG problem is NP-complete [15]. This means that there cannot exist an algorithm which solves an arbitrary instance of this problem in polynomial time, unless $P = NP$. However, as early as 1979, Williams and Parker [25] claimed that for practically encountered instances of the problem the complexity of ATPG is only $O(n^3)$. In fact, the widespread use of ATPG-based techniques can largely be attributed to the relative ease with which large instances of the problem are solved in practice.

We corroborated the claim that ATPG is easily solvable in practice by performing the following experiment. ATPG was carried out on the combinational circuits from the **MCNC91** [26] and **ISCAS85** [5] benchmark suites, using **TEGUS** [24], a SAT-based ATPG tool. The time to solve each SAT instance was recorded as a function of the size of the instance and plotted in Figure 1. Of the 11,000 SAT instances generated, some with over 15,000 variables, over 90% were solved in less than 1/100th of second; the remaining exhibited roughly a cubic growth in execution time. Thus, the theoretical worst case complexity of ATPG, *i.e.* the fact that it is NP-complete, would seem to be a poor indicator of the practical ease of the problem. This work is one of the first attempts to offer a theoretical explanation for the practical ease of ATPG.

The practical ease of ATPG suggests that there is some underlying property common to real-life ATPG instances which makes them tractable. These instances are usually derived from practical VLSI circuits. Therefore, we develop a characterization of the complexity of solving ATPG in terms of a topological circuit property, namely *cut-width*. We also demonstrate, through theoretical arguments and experiments on practical circuits, that a large class of interesting circuits have small *cut-widths*, provably permitting efficient solution of ATPG on them.

We use a popular formulation based on SAT (also a well known NP-complete problem) as our working model of the ATPG algorithm. This formulation was originally proposed by Larrabee [18] and later developed by Stephan *et al.* [24]. The reason for this choice is twofold. First, SAT is a well researched problem (see [12] for an excellent survey), and offers a clean and general framework for analyzing a wide range of search techniques and heuristics. Second, SAT is a solution platform for a variety of other CAD applications. Therefore, understanding the functioning of SAT on ATPG might offer insight into several other related SAT applications.

The remainder of the paper is organized as follows. We begin with some definitions and notation in Section 2. In Section 3 we discuss some seemingly promising approaches for analyzing the complexity of ATPG instances. Unfortunately, these approaches provide only an incomplete or inconclusive answer to the practical easiness of ATPG. Section 4 presents our model of the backtracking based algorithm for solving SAT, the *cut-width* property of circuits, and an analysis of the complexity of ATPG in terms of cut-width. In Section 5 we present both theoretical arguments and empirical results to show that a cut-width based argument does in fact predict a polynomial runtime of ATPG on a large class of practical circuits. In Section 6 we present interesting parallels and points of contrast between our results and published work addressing bounds on the size of binary decision diagrams (BDDs). We conclude with directions for future research in Section 7.

2 Definitions and Notation

Given a combinational Boolean network C [4], a *single stuck-at fault* $\psi = \psi(X, \mathcal{B})$ is one which causes a net X in C to be permanently stuck at logic value $\mathcal{B}$ (where $\mathcal{B} \in \{0,1\}$). The *faulted circuit*, denoted by C_ψ, is then C with the fault ψ operative *i.e. fault-net* X asserted to value $\mathcal{B}$. The ATPG problem $\text{ATPG}(C, \psi(X, \mathcal{B}))$, is defined with respect to the *single stuck-at fault* ψ on circuit C and has the answer *YES* if and only if there exists an assignment of Boolean values to the primary inputs of C (and C_ψ) such that fault net X has complementary logic values in C and C_ψ and at least one of the primary outputs of C (and C_ψ) have complementary logic values. If such a Boolean assignment exists it is said to be a test for the fault ψ. Otherwise the fault is said to be untestable.

A conjunctive normal form (CNF) Boolean formula f on n Boolean variables $x_1, x_2, \ldots, x_n$ is a set of m clauses $\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_m$. Each clause $\mathcal{C}_i$ is a set of k_i literals $l_1, \ldots l_{k_i}$. A lit-

*This work was supported in part by California MICRO and NSERC Canada

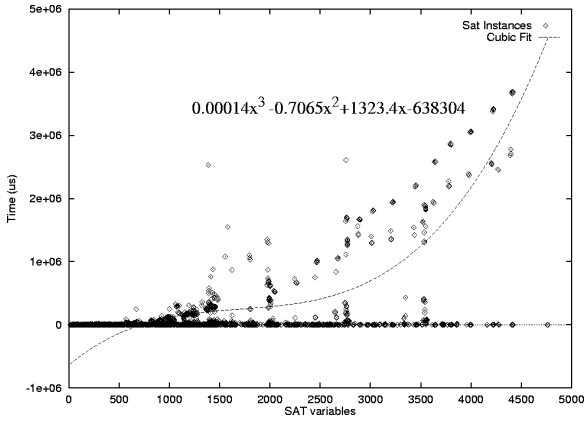


Figure 1: Results of TEGUS on ATPG-SAT instances

eral is an instance of a variable or its complement. f is interpreted as the conjunction (logical AND) of the clauses $\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_m$, each of which is interpreted as the disjunction (logical OR) of its constituent literals. For example, the formula $f = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_3)$ is represented as $f = \{\mathcal{C}_1, \mathcal{C}_2\}$, where $\mathcal{C}_1 = \{x_1, \overline{x_2}\}$ and $\mathcal{C}_2 = \{x_2, x_3\}$. The Boolean satisfiability problem $\mathrm{SAT}(f)$, defined on the CNF Boolean formula f has an answer YES iff there exists an assignment of Boolean values to the variables $x_1, x_2, \ldots, x_n$ under which f evaluates to 1. A similar satisfiability problem, *CIRCUIT-SAT* can be posed on a Boolean circuit C. The problem $\mathrm{CIRCUIT-SAT}(C)$ has an answer YES iff there exists an assignment of Boolean values to the primary inputs of C which sets at least one of the primary outputs of C to logic value 1. This assignment is called a satisfying assignment of C.

SAT and CIRCUIT-SAT are well known instances of *NP-complete* problems [11]. A CIRCUIT-SAT problem on C can be posed as a SAT problem on an appropriate Boolean formula $f(C)$. $f(C)$ has one variable for each signal net in C and a set of clauses for each gate (of the form shown in Figure 2). Additionally, there is a clause asserting that at least one output needs to be 1. In the following treatment we will make no distinction between the CIRCUIT-SAT problem on a circuit C and the Boolean satisfiability problem on its corresponding Boolean formula $f(C)$. The set of variables of $f(C)$ will be denoted by V_C.

The ATPG problem can be naturally cast as a satisfiability problem by formulating it as a CIRCUIT-SAT problem on a suitable circuit derived from the original circuit C and the fault ψ. Define C_ψ^{fo} as the subcircuit of C_ψ corresponding to the *transitive-fanout* of X in C_ψ and C_ψ^{sub} as the subcircuit of C containing all gates, inputs and outputs in the transitive fanin of the transitive fanout of the fault-point X. Then C_ψ^{ATPG} is the circuit corresponding to the pairwise *XOR* of the outputs of C_ψ^{sub} and C_ψ^{fo}. C_ψ^{fanout} derives its inputs from appropriate signal points in C_ψ^{sub} (see Figure 3). The set of all satisfying assignments for the

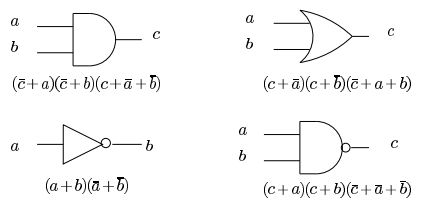


Figure 2: SAT formulas for some simple gates

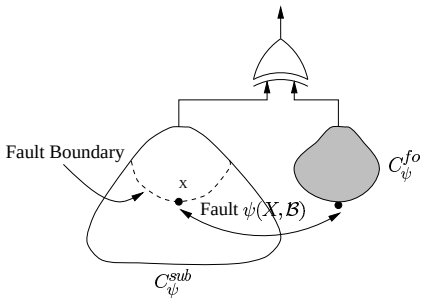


Figure 3: Circuit C_ψ^{ATPG} used for ATPG-SAT

CIRCUIT-SAT instance C_ψ^{ATPG} gives precisely the set of all input vectors that test the fault ψ (see [18] for details). Thus the ATPG problem $\mathrm{ATPG}(C, \psi(X, C))$ can be formulated as an instance of Boolean satisfiability, namely $\mathrm{CIRCUIT-SAT}(C_\psi^{ATPG})$. Henceforth, we will refer to this special instance of the satisfiability problem as *ATPG-SAT*. Specifically, $\mathrm{ATPG-SAT}(C, \psi) \equiv \mathrm{CIRCUIT-SAT}(C_\psi^{ATPG})$.

Throughout this discussion, we assume the circuits we deal with have gates with fanin and fanout bounded by k_{fi} and k_{fo}, respectively. We also assume the circuits are mapped to simple AND and OR gates, allowing inversions. The former restriction is enforced for practicality; design and technology constraints prohibit unlimited fanin and fanout. The latter restriction is enforced to facilitate the construction of the corresponding SAT formulas; it is difficult in practice to derive SAT formulas for arbitrary gates. TEGUS [24] enforces this latter condition for exactly this reason.

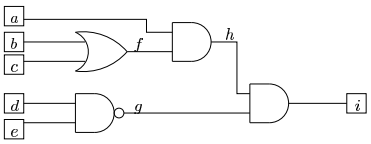
3 Applying Existing Techniques

One approach to exploring the easiness of ATPG is to try to determine what classes of ATPG problems can be efficiently solved. Alternatively, one could consider the SAT formulation of ATPG and attempt to show that ATPG-SAT instances are easy to solve. Here we discuss three such approaches based on applications of existing results and techniques. We show that none of them is capable of offering a conclusive or sufficiently general explanation for the easiness of ATPG. The reason for this is that these approaches cannot exploit the fact that practical circuits are considerably more regular than arbitrary Boolean circuits. In our analysis (Section 4) we address this shortcoming by characterizing the complexity of ATPG-SAT in terms of topological circuit properties.

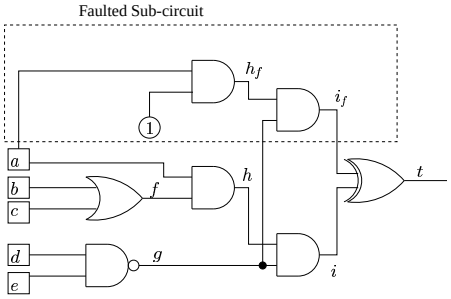
3.1 Simple SAT Classes

Some classes of SAT problems are known to be solvable in polynomial time; efficient algorithms are known for solving SAT formulas of a particular form. Horn-SAT is one such widely known class. Boros *et al.* [2] identify an even more general class of SAT formulas known as *q-Horn*; the set of q-Horn problems include Horn-SAT problems as well as several other polynomial time classes, such as 2-SAT, Hidden-Horn-SAT and Extended-Horn-SAT.

If we could show that an interestingly large class of ATPG-SAT instances fall into one of the known polynomial time solvable SAT classes it would imply that the corresponding class of ATPG problems are efficiently solvable. However, we can demonstrate simple circuits with corresponding ATPG-SAT formulas that are not q-Horn [7]. Thus it is unlikely that any ATPG-SAT instances of practical significance lie in one of the polynomial SAT classes.



(a) CIRCUIT-SAT example



(b) ATPG circuit example

Figure 4: Example Circuits

3.2 k-bounded Circuits

Fujiwara [10] introduced the notion of *k-bounded* circuits and showed that ATPG can be efficiently performed on this class of circuits. This class of circuits was shown to contain some circuits of practical interest such as ripple-carry adders, decoders, and one- and two-dimensional cellular arrays.

Briefly, a circuit is k-bounded if its nodes can be partitioned into disjoint blocks such that each block has at most k inputs, and the blocks form a DAG with **no reconvergent paths**. Simply put this means that all the reconvergence of the circuit is of a *local* nature, *i.e.* confined within k-input blocks. Practical circuits with deep reconvergent paths are abundant. Hence, k-boundedness seems to be too restrictive a property to be applied to general VLSI circuits.

3.3 Average-Time Analysis

Another approach to assessing the complexity of ATPG-SAT is to perform an average running time analysis on the the population of ATPG-SAT instances. Purdom and Brown [21] take this approach in modeling backtracking algorithms for arbitrary SAT problems. In their study, SAT formulas are parameterized based on the relationships between the number of variables, the number of clauses and the length of the clauses in the SAT instance. Based on this parameterization, classes of SAT formulas are shown to be solvable in polynomial time on average. Under reasonable assumptions, it can be shown that the SAT formulas generated from ATPG instances fall into a class known to be easily solvable on average [7].

Note, however, that while our ATPG-SAT formulas are in an easy class, there are SAT formulas in this class which have no corresponding ATPG formulation. Thus, while problems in the overall class are easily solvable on average, the smaller subset of ATPG problems need not be solvable in polynomial time on average. Therefore this analysis *suggests* ATPG-SAT is easy, but we cannot bring any hard conclusions to bear.

4 Analysis of ATPG-SAT

A number of approaches for solving *Boolean satisfiability* have been proposed in the literature [12]. Among these, backtracking based approaches are the most popular. Hence, for our analysis of ATPG-SAT we chose to model the SAT algorithm by a "caching based" variant of *simple backtracking* [12]. This algorithm is described in Section 4.1. We introduce the notion of *cut-width* of a circuit and characterize the worst case complexity of solving ATPG-SAT instances in terms of the cut-width of circuits from which the instances were derived.

To illustrate the salient results, we will use the circuit shown in Figure 4(a) as our working example. As per the discussion in Section 2 the CIRCUIT-SAT instance corresponding to this circuit is:

$$(\overline{b}+f)(\overline{c}+f)(b+c+\overline{f})(d+g)(e+g)(\overline{d}+\overline{e}+\overline{g})(a+\overline{h})$$
$$(f+\overline{h})(\overline{a}+\overline{f}+h)(h+\overline{i})(g+\overline{i})(\overline{h}+\overline{g}+i)(i) \qquad (4.1)$$

The ATPG problem we consider is a **stuck-at-1** fault on the net f. The ATPG-SAT instance generated by this fault corresponds to the circuit shown in Figure 4(b).

4.1 Caching-Based Backtracking for CIRCUIT-SAT

Most popular backtracking based algorithms, especially those proposed in the CAD literature [23, 24], provide some feature to reduce *conflicts* during backtracking. This may be in the form of a pre-processed set of *global implications* [24] or in the form of generating and storing *conflict-induced clauses* by "learning" from conflicts [23]. Our *caching based version* of simple backtracking is a simplified way of modeling this feature.

The essential idea of caching based backtracking is to perform simple backtracking with a fixed variable order, except that whenever the algorithm backtracks from an unsatisfiable sub-formula, the sub-formula is cached. Correspondingly, before a sub-formula is taken up for a satisfiability check, it is looked up in the cache. If found, it can be diagnosed immediately as being unsatisfiable and the algorithm can backtrack from it without trying any further variable assignments. The pseudo-code for the algorithm appears below. In Algorithm 1, f is the CNF Boolean formula for the satisfiability check, h is a function that orders the variables of f, and $\mathcal{T}$ is a hash table for storing the set of **unsatisfiable** sub-formulas of f encountered during the backtracking search.

Algorithm 1 Satisfiability through Caching-Based Backtracking

```
procedure Sat(T, h, f)
T ← ∅
if Cache_Sat(v_first, 0, f) = "UNSAT" and Cache_Sat(v_first, 1, f) = "UNSAT"
then
   return "UNSAT"
else
   return "SAT"
end if

procedure Cache_Sat(v_current, B, f_sub)
{v_current : Variable currently chosen for assignment,
B :Value assigned to v_current}
f_sub ← Assign(f_sub, v_current, B)
if Null_Clause(f_sub) then
   return "UNSAT"
else {f_sub has no NULL clauses}
   if Table_Lookup(T, f_sub) then
      return "UNSAT"
   end if
   v_next ← Next_Var(v_current, h)
   if Cache_SAT(v_next, 0, f_sub) = "SAT" then
      return "SAT"
   end if
   if Cache_SAT(v_next, 1, f_sub) = "SAT" then
      return "SAT"
   end if
   {Both Subtrees UNSAT}
   Insert_Table(T, f_sub)
   return "UNSAT"
end if
```

Figure 5 shows an example run of this algorithm on Formula 4.1. The variable ordering $A = (b < c < f < a < h < d < e < g <$

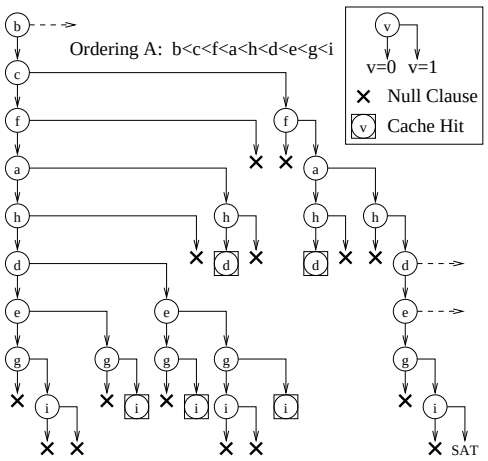


Figure 5: Caching-based backtracking for Formula 4.1

$i)$ is used for the backtracking search. Note there are several places where the caching strategy works to prune the search. For example, consider the partial assignment $b = 0, c = 0, f = 0, a = 0, h = 0$; this leaves the sub-formula $(d+g)(e+g)(\overline{d}+\overline{e}+\overline{g})(g+\overline{i})(\overline{i})(i)$. This same sub-formula is obtained under the assignment $b = 0, c = 0, f = 0, a = 1, h = 0$, and so we can prune this branch of the search without further computation.

The running time of Algorithm 1 on a given formula f, is denoted by $\mathcal{R}(f)$ and can be analyzed as follows. A *sub-formula* of f is obtained by setting a subset of the variables of f to certain values. Define a *consistent sub-formula* of f as a *sub-formula* having no *empty clauses*[1] (i.e. a clause where all the literals have been set to $false$ under the partial assignment).

We assume that the sub-formulas are cached as sets of clauses. Thus, from our point of view two sub-formulas are identical if and only if they have the same set of clauses.[2] $\mathcal{R}(f)$ is upper bounded by the product of the size of the backtracking tree and the worst case time for a single cache access (insertion, lookup or deletion). Since each cache access can be at worst linear in the size of the backtracking tree, the specific cache access time cannot alter the asymptotic nature of the running time (*i.e.* cause the difference between a polynomial and exponential run time). Hence, for the purpose of this analysis we assume that the caching is perfect; cache lookups and insertions can be done in constant time. Under this scenario, $\mathcal{R}(f)$ is upper-bounded by the size of the backtracking tree, which in turn is bounded by the number of *distinct consistent sub-formulas (DCSFs)* of f that can be generated under a particular static ordering of the formula variables. Thus, under the ordering h, $\mathcal{R}(f) = O(\mathcal{F}(\mathcal{P}_h(V)))$ where $\mathcal{F}(\mathcal{P}_h(V))$ is the number of DCSFs of f, under the ordering h, V is the set of variables of f and $\mathcal{P}_h(V)$ denotes the set of those subsets of V which are valid prefixes of the ordering h. If the formula f corresponds to a CIRCUIT-SAT instance, generated from a circuit C, we further characterize $\mathcal{R}(f)$ in terms of a topological property of C. This characterization is developed in the following section.

4.2 Cut-width and Sub-formula Count

Consider a CIRCUIT-SAT formula $f(C)$ corresponding to circuit C. For the initial part of the analysis assume that C has a single output. The results are extended to multi-output circuits, in Section 4.3. The network C can be seen as an undirected hypergraph with the signals as the hyperedges, and the gates, inputs and outputs as the nodes. For the purpose of this exposition a Boolean network and its underlying hypergraph are not distinguished. *Cut-width* of a hypergraph is defined as follows.

[1] A formula with empty clauses is trivially unsatisfiable.

[2] Sub-formulas with a different set of clauses may still be functionally equivalent. However, we do not recognize this equivalence in this treatment.

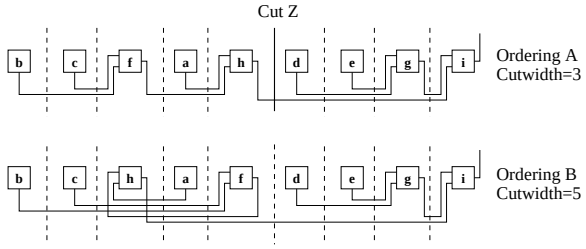


Figure 6: Example cut-widths for the circuit of Figure 4(a)

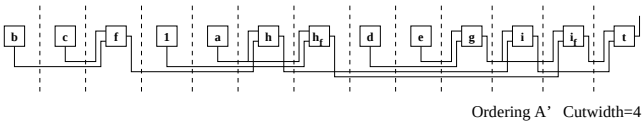


Figure 7: Example cut-width, ATPG circuit of Figure 4(b)

Definition 4.1 *Given a hypergraph $G(V, E)$ and a one-to-one function h, ordering the vertices of G. $h : V \rightarrow \{1, 2, \ldots, |V|\}$. The* cut-width *of G, under the ordering h, is denoted as $W(G, h)$ and is given by the expression*

$$W(G,h) = \max_{i \in \{1,2,\ldots,|V|\}} |\{e \in E : \exists u, v \in V \text{ such that } \{u,v\} \subseteq e \text{ and } h(u) \leq i < h(v)\}|$$

(**Note:** Each hyperedge e of G is denoted by the set of vertices spanned by it.) The *minimum cut-width* of G over all possible orderings h is denoted by $W_{min}(G)$. Henceforth, cut-width of a circuit without mention of a particular variable ordering will refer to the minimum cut-width $W_{min}(G)$. Figure 6 illustrates the notion of cut-width on the example circuit from Figure 4(a), using two different variable orderings, A and B. Ordering A, which was used for the backtracking tree example of Figure 5, also happens to be a minimum cut-width (W_{min}) ordering for this circuit.

The number of nodes at a certain level in the backtracking tree for $f(C)$ can be bounded in terms of the size of an appropriate *cut* of the circuit C. A partition $(\delta_{V_C}, \overline{\delta_{V_C}})$ of the variables V_C defines a unique *cut* in C. An assignment of truth values to the variables δ_{V_C} in the formula $f(C)$ yields a sub-formula $f_{sub}(C)$ of $f(C)$.

Lemma 4.1 *Given a Boolean network C, its corresponding CIRCUIT-SAT formula $f(C)$ and a* cut *$(\delta_{V_C}, \overline{\delta_{V_C}})$ of V_C, the number of DCSFs that can be obtained by the set of all possible truth assignments to the variables δ_{V_C} is denoted by $\mathcal{F}(\delta_{V_C})$ and can be bounded as:*

$$\mathcal{F}(\delta_{V_C}) \leq 2^{2^{k_{fo}}|(\delta_{V_C}, \overline{\delta_{V_C}})|} \tag{4.2}$$

where $|(\delta_{V_C}, \overline{\delta_{V_C}})|$ denotes the size of the cut, i.e. the number of distinct nets crossing the cut.

The usefulness of this result (see [7] for the proof) stems from the fact that the formula set size is exponential not in the size of the variable set but in the size of the cut, which is potentially much smaller. For example consider the cut $(\delta_V, \overline{\delta_V})$ on the circuit of Figure 4(a), with $\delta_V = \{b, c, f, a, h\}$; this corresponds to the level in the backtracking tree corresponding to the *Cut Z* label in Figure 6. A naive bound suggests that there are 2^5 possible sub-formulas generated after the assignment of the variables in δ_V (there are 2^5 distinct assignments to the variables). However, the assigned variables have

only one means of affecting the sub-formulas on the unassigned variables; this is through the single cut net between h and i. Thus regardless of the assignment to the variables of δ_V, Lemma 4.1 indicates that there can be at most 2^2 distinct sub-formulas for any assignment to the δ_V variables. Based on this result and the above definition of cut-width we derive the following bound for the running time of Algorithm 1 on $f(C)$.

Theorem 4.1 *Given a Boolean network C and ordering h on V_C, Algorithm 1 can solve the CIRCUIT-SAT instance $f(C)$ in time $O(n \cdot (2^{2k_{fo}W(C,h)}))$, where $n = |V_C|$.*

From the above result (see [7] for the proof) it is evident that if a circuit has a cut-width which is *logarithmic* in the size of the circuit, CIRCUIT-SAT can be performed on it in polynomial time. We discuss further implications of this result in Section 5 where we provide theoretical as well as empirical results analyzing cut-width properties for practical classes of circuits.

As explained in Section 2, under the SAT formulation of the ATPG problem, testing for a certain fault ψ on a circuit C amounts to performing CIRCUIT-SAT on a certain circuit, namely C_ψ^{ATPG}. The following result (see [7] for proof) shows that, for any fault ψ in circuit C, the cut-width of C is linearly related to the cut-width of C_ψ^{ATPG}. This means that we can reason about the asymptotic behavior of Algorithm 1, on ATPG-SAT instances generated from circuit C, by analyzing the cut-width properties of circuit C (or sub-circuits thereof) rather than having to deal with the circuit C_ψ^{ATPG}, which could be significantly more involved.

Lemma 4.2 *Given a Boolean network C, for any ordering h of the variables V_C and any fault ψ on C , $\exists$ an ordering h_ψ of the variables of C_ψ^{ATPG} such that*

$$W(C_\psi^{ATPG}, h_\psi) \leq 2 \cdot W(C, h) + 2 \tag{4.3}$$

Figure 7 illustrates this result on our example ATPG circuit from Figure 4(b). As shown in Figure 6 the circuit of Figure 4(a) has a cut-width of 3 under ordering A (Figure 5). The ordering A' can be derived from this to yield a cut-width of 4 for the ATPG circuit of Figure 4(b).

4.3 Extension to Multi-output Circuits

The discussion so far has been restricted to single-output circuits. Consider a multi-output circuit C, with p primary outputs $o_1, o_2, \ldots o_p$. For the purpose of a CIRCUIT-SAT test, C can be seen as a set of p single-output circuits $\{C_1, C_2, \ldots C_p\}$, one each for the *transitive fanin cone* of each primary output. CIRCUIT-SAT on C can be performed by performing CIRCUIT-SAT on each of the single-output circuits $C_1, C_2, \ldots C_p$, one at a time. Then, $\mathrm{CIRCUIT-SAT}(C) = \mathrm{CIRCUIT-SAT}(C_1) \vee \ldots \vee \mathrm{CIRCUIT-SAT}(C_p)$.

In this scenario, the results of Sections 4.1 and 4.2 can be applied to multi-output circuits as follows. Given a multi-output circuit $C = \{C_1, C_2, \ldots C_p\}$ and a set $H = \{h_1, h_2, \ldots h_p\}$ of node orderings for the single-output circuits $C_1, C_2, \ldots C_p$, the notion of cut-width as given by Definition 4.1 can be extended as:

$$W(C, H) = \max_{i \in \{1,2,\ldots p\}} W(C_i, h_i) \tag{4.4}$$

The minimum cut-width $W_{min}(C)$ generalizes on similar lines, except now the minimum is over all possible *sets of orderings* H. Hence the running time of $\mathrm{CIRCUIT-SAT}(C)$, (based on Algorithm 1) can be bounded as:

$$\mathcal{R}(f(C)) = O(p \cdot n_{max} \cdot 2^{2k_{fo}W(C,H)}), \quad \text{where } n_{max} = \max_{i \in \{1,2,\ldots p\}} |V_{C_i}|. \tag{4.5}$$

Similarly, Lemma 4.2 can be restated as:

Lemma 4.3 *Given a multi-output Boolean network C, for any set of orderings $H = \{h_1, h_2, \ldots h_p\}$ of the variables $V_{C_1}, V_{C_2}, \ldots V_{C_p}$ and any fault ψ on C, $\exists$ an ordering H_ψ of the variables of C_ψ^{ATPG} such that*

$$W(C_\psi^{ATPG}, H_\psi) \leq 2 \cdot W(C, H) + 2 \tag{4.6}$$

5 Cut-width Properties of Circuits

5.1 Log-bounded-width Circuits

In the following we define a class of circuits known as *log-bounded-width circuits* and show that by employing Algorithm 1 ATPG can be efficiently performed on these circuits. We also prove that k-bounded circuits (see Section 3.2) lie within the class of log-bounded-width circuits (see [7] for proofs).

Definition 5.1 *A given multi-output circuit C is* log-bounded-width *if for each single stuck-at fault ψ on C, there exists a set of orderings H of the variables $V_{C_\psi^{sub}}$, such that*

$$W(C_\psi^{sub}, H) = O(log(|C_\psi^{sub}|)) \tag{5.1}$$

Lemma 5.1 *Given a log-bounded-width circuit C and any single stuck-at fault ψ on C, test generation for ψ can be accomplished in time polynomial in the size of the circuit C.*

Lemma 5.2 *Given a k-ary tree $\mathcal{T}$, there exists an ordering h, of the variables $V_\mathcal{T}$ such that $W(\mathcal{T}, h) \leq (k-1)\log(n)$.*

Theorem 5.1 *Any k-bounded circuit, for a given constant k is log-bounded-width.*

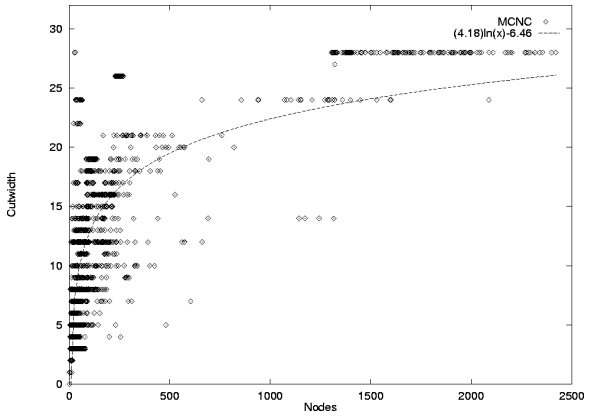
As shown above, tree circuits are of log-bounded-width. Intuitively, reconvergence tends to increase circuit cut-width. But, as long as the circuits are sufficiently "tree-like" the log-bounded-width property could be expected to apply. The *locality of reconvergence* required by k-boundedness is just one instance of this. In principle log-bounded-width simply requires a *minimality of reconvergence* and is therefore a more general property than k-boundedness.

5.2 Practical VLSI Circuits

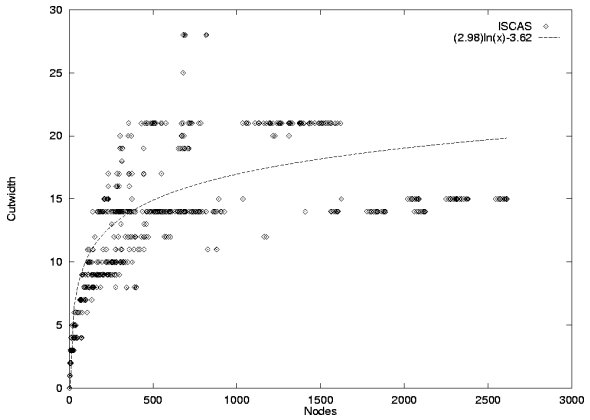
From the discussion on circuit cut-width it is apparent that cut-width is intrinsically linked to the circuit topology. However, practical designs are not usually specified with topology in mind, and so we cannot *a priori* identify log-bounded-width circuits. Thus, we have performed an empirical study of cut-width for a set of circuits. We consider the circuits from the **MCNC91** and **ISCAS85** multi-level combinational benchmark suites and estimate their cut-widths. By showing that the cut-widths for these circuits grow slowly with the circuit size, we provide evidence that these circuits do in fact have log-bounded-width and hence are easily testable.

5.2.1 Experimental Setup

The key element of our experimental setup is our measurement of the *cut-width* of a single-output circuit C. By definition, the minimum cut-width is the value of the *max-cut* obtained under a *min-cut linear arrangement* (MLA) [11] of C. Since the MLA problem is known to be NP-complete, we use a well-known algorithm [13] to approximate the MLA, and hence estimate the cut-width for a given circuit. This algorithm generates a placement based on recursive mincut bipartitioning, until the partitions are sufficiently small



(a) MCNC91 logic benchmarks (11315 datapoints)



(b) ISCAS85 benchmarks (7389 datapoints)

Figure 8: Cut-width Results For Benchmark Circuits

and then performs an exact MLA for each of these partitions. We used the **HMETIS** package [16] from the University of Minnesota to perform the bipartitioning.

Note the ease of the overall ATPG problem for a circuit is related to the difficulty of each individual ATPG-SAT instance for that circuit. Thus, we generated one data-point for each potential fault ψ in circuit C. This data-point measures the approximate cut-width of the circuit C_{ψ}^{sub} versus the size of this circuit. The size of the circuit C_{ψ}^{sub} is an approximate measure of the size (number of variables) in the SAT instance $\mathrm{ATPG} - \mathrm{SAT}(C, \psi)$ and the cut-width of this circuit is indicative of the complexity of solving this instance (as per Equation 4.5 and Lemma 4.3).

Note that we do not necessarily advocate using MLA as a method for solving ATPG. Though this method leads to a feasible implementation of Algorithm 1, we only use the MLA here to demonstrate that we expect the benchmark circuits to be easily testable due to their small cut-widths.

5.2.2 Study of Benchmark Suites

The structures of the **MCNC91** and **ISCAS85** benchmark circuits vary widely; some circuits have nodes with over a dozen inputs each, other circuits implement complex logic at each node, while others are decomposed into 2-input AND gates and inverters. Actual implementations of circuits would not exhibit this wide variation of complexity; fanin and node complexity is necessarily limited due to speed and size requirements on the gates. Moreover, for the purposes of performing ATPG it is often desirable to map circuits to simple AND and OR gates (with inverters), as this makes the corresponding SAT formulas much easier to derive. Thus, we mapped the benchmark circuits to three (or fewer) input AND/OR gates, allowing inversions. The `tech_decomp` procedure from the logic optimization package **SIS** [22] was used to perform this mapping.

As described above, for every potential fault ψ in each circuit, we determined the difficulty of solving the related ATPG-SAT instance by finding an estimate for the cut-width of the subcircuit C_{ψ}^{sub}. Figure 8(a) shows the results for the circuits identified as "logic" circuits from the **MCNC91** benchmark suite. We excluded circuit t481, which we considered degenerate, having over 3800 nodes after gate mapping yet with only a single output. Figure 8(b) corresponds to the **ISCAS85** combinational benchmark circuits. We omitted the circuits C3540 and C6288 in this analysis, due to limitations in our min-cut linear arrangement procedure. Our method ran successfully for all the remaining benchmarks (48 from **MCNC91** and 9 from **ISCAS85**).

Note that the growth of cut-width versus circuit size is small; on the graphs we plot log curves fitted to the data points to illustrate this. These curves proved to give the best least-squares fit to the data [20], among linear ($y = ax + b$), logarithmic ($y = a\log(x) + b$) and power ($y = ax^b$) curves. These plots suggest that the cut-width is roughly a logarithmic function of circuit size for these circuits, and so we can expect these benchmarks to be easily testable. This agrees with the empirical results from **TEGUS** (Figure 1).

5.2.3 Using Generated Circuits

To strengthen the evidence obtained from the above experiment, we repeated the technique on artificially generated circuits [14], parameterized to topologically resemble circuits from the **MCNC91** and **ISCAS85** suites. This allowed us to examine the growth of cut-width for extremely large circuits. The complete results from this experiment are presented in [7]; in summary, the same logarithmic increase in cutwidth versus circuit size was seen for the generated circuits as was observed for the actual benchmark circuits.

6 BDDs and CIRCUIT-SAT

The concept of *circuit-width* has been used by researchers [1, 19] to obtain upper bounds on the size of BDDs representing the circuit function. Binary decision diagrams (BDDs) and CNF Boolean formulas are both representations of Boolean functions. Solving CIRCUIT-SAT on a Boolean circuit C could be done by doing a "0" check on the BDD for C. Alternatively, one can construct a CNF Boolean formula $f(C)$ and solve satisfiability on the formula using a backtracking algorithm. In essence, a BDD and a backtracking tree represent the same entity, *i.e.* the Boolean space of the function. However, our results have no direct relationship to the BDD bounds.

Berman [1] gave a bound on the BDD size, for any *topological ordering* of the circuit elements. This result was extended by McMillan [19] for arbitrary orderings. McMillan's result can be summarized as follows. Given a single-output circuit C, with n inputs, if the elements of C can be linearly ordered such that over all cross-sections of the linear arrangement, w_f (forward width) bounds the number of wires running in the forward direction and w_r (reverse width) bounds the number of wires in the reverse direction, then the size of the BDD representing the output of C can be upper bounded by $n2^{w_f 2^{w_r}}$. This result differs from the result presented in this paper on two counts.

- Our definition of circuit cut-width is *independent of the direction of signal-flow* (our characterization of width is on an undirected hypergraph) and thus substantially different from w_f and w_r in an operational sense.

- The above result is exponential in the *forward width* and double-exponential in the *reverse width*, while our result has only a single exponential. We exploit this property in defining the class of log-bounded-width circuits.

The key to these differences lies in the fact that BDDs represent the intrinsic nature of a Boolean function, independent of the specific hardware implementation, while CIRCUIT-SAT formulas (as per the construction of Section 2) are in one to one correspondence with the circuit topology. The proof techniques used in the two results reflect this difference. Therefore, the two results, although similar in spirit, characterize different entities altogether.

7 Conclusions and Future Work

We have presented one of the first attempts at reconciling the theoretical, worst case complexity of ATPG with the relative ease with which practical instances of it are solved. For the purpose of analysis we have employed a SAT based formulation of ATPG [18], with a caching based variant of simple backtracking (see Section 4) used to model the SAT solver.

Under this model of the algorithm the complexity of ATPG on a given circuit has been characterized in terms of a topological property of the circuit, namely the *circuit cut-width*. Theoretical arguments and experimental results confirm that this property can be used to predict polynomial runtimes of ATPG, for a wide range of practical VLSI circuits.

Specifically, this analysis has been used to define a class of circuits called log-bounded-width circuits which we have shown to be efficiently testable. Additionally, this class of circuits has been shown to subsume the class of k-bounded circuits. Our experiments on a wide range of benchmark and generated circuits show that they exhibit the log-bounded-width property. On an intuitive level the log-bounded-width property essentially captures the "treeness" of the circuit. As long as a circuit has limited reconvergence (not necessarily local reconvergence), the log-bounded-width property can be expected to apply.

More generally, algorithm analysis in CAD has seen two distinct eras. The first is the era in which little distinction was made between the fundamental complexity of a problem that was being solved and the best known running time of a particular piece of software to solve it. As algorithmic maturity improved and as computational complexity concepts such as NP-hardness and NP-completeness became more widely known a second era of algorithmic analysis began. In this era developers of algorithms felt obligated to perform some formal analysis of their algorithms and show worst case computational complexity results. But soon it was discovered that some algorithms attacking some NP-hard problems, such as ATPG, were quite efficient, while those with provable polynomial bounds had unacceptably high runtimes. As a result, worst-case complexity analysis alone has been seen to be inadequate to enable algorithm designers to anticipate and understand the running time of their algorithms in practice. To remedy this situation we need to gain greater insight into the particular properties of the objects, such as sequential circuit netlists, to which our algorithms are applied, if we are to accurately understand the true computational complexity of the CAD problems that we must solve. Like [8], this work aims to be one step in that direction.

Acknowledgments

Thanks to George Karypis and the rest of the **HMETIS** team, and Mike Hutton and the `circ/gen` team, for providing key software components. We are grateful to Christos Papadimitriou, Olivier Coudert and Yuji Kukimoto for useful feedback and discussion.

References

[1] C. L. Berman. Circuit Width, Register Allocation and Ordered Binary Decision Diagrams. *IEEE Trans. CAD*, 10(8):1059–1066, Aug 1991.

[2] E. Boros, Y. Crama, and P. L. Hammer. Polynomial-time Inference of All Valid Implications for Horn and Related Formulae. *Ann. Math Art. Intell.*, 1:21–32, 1990.

[3] D. Brand. Verification of Large Synthesized Designs. In *IEEE ICCAD*, pages 534–537, 1993.

[4] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: A Multiple-Level Logic Optimization System. *IEEE Trans. on CAD/ICAS*, CAD-6(6):1062–1082, Nov 1987.

[5] F. Brglez and H. Fujiwara. A Neural Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *Intl. Symp. on Circuits and Systems*, Jun 1985.

[6] K.-T. Cheng and L. A. Entrena. Multi-level Logic Optimization by Redundancy Addition and Removal. In *European Conference on Design Automation*, pages 373–377, Jun 1993.

[7] P. Chong, M. R. Prasad, and K. Keutzer. Why is ATPG Easy ? Technical Report UCB/ERL M99/9, ERL, University of California, Berkeley, Feb 1999.

[8] O. Coudert. Exact Covering of Real-Life Graphs is Easy. In *Proceedings of the* 34^{th} *DAC*, pages 121–126, Jun 1997.

[9] S. Devadas, H.-K. T. Ma, and A. Sangiovanni-Vincentelli. Logic Verification, Testing and Their Relationship to Logic Synthesis. In *Testing and Diagnosis of VLSI and ULSI*, pages 181–246. Kluwer Academic Publishers, 1988.

[10] H. Fujiwara. Computational Complexity of Controllability/Observability Problems for Combinaitonal Circuits. In *Intl. Symp. on Fault-Tolerant Computing*, pages 64–69, Jun 1988.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[12] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah. Algorithms for the Satisfiability (SAT) Problem: A Survey. *DIMACS Series in Discrete Mathematics and Computer Science*, 35:19–151, 1997.

[13] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1997.

[14] M. Hutton, J. Grossman, J. Rose, and D. Corneil. Characterization and Parameterized Random Generation of Digital Circuits. In *33rd Design Automation Conference*, pages 94–99, 1996.

[15] O. H. Ibarra and S. K. Sahni. Polynomially Complete Fault Detection Problems. *IEEE Trans. Computers*, C-24(3):242–249, Mar 1975.

[16] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. In *34th Design Automation Conference*, pages 526–529, 1997.

[17] A. Kuehlmann, A. Srinivasan, and D. P. LaPotin. Verity - A Formal Verification Program for Custom CMOS Circuits. *IBM Journal of Research and Development*, 39:149–165, 1995.

[18] T. Larrabee. Efficient Generation of Test Patterns Using Boolean Difference. In *Intl. Test Conference*, pages 795–801, 1989.

[19] K. L. McMillan. *Symbolic model checking: An approach to the state explosion problem*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1992.

[20] S. L. Meyer. *Data Analysis For Scientists and Engineers*. Wiley and Sons, 1975.

[21] P. W. Purdom and C. A. Brown. Polynomial-Average-Time Satisfiability Problems. *Information Sciences*, 41:23–42, 1987.

[22] E. M. Sentovich et al. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, ERL, College of Engineering, University of California, Berkeley, May 1998.

[23] J. P. M. Silva and K. A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *ICCAD*, pages 220–227, 1996.

[24] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Combinational Test Generation Using Satisfiability. *IEEE Trans. on CAD/ICAS*, 15(9):1167–1176, Sep 1996.

[25] T. W. Williams and K. Parker. Testing Logic Networks and Designing for Testability. *Computer*, pages 9–21, Oct 1979.

[26] S. Yang. Logic Synthesis and Optimization Benchmarks User Guide, Version 3.0. Technical report, Microelectronics Center of North Carolina, 1991.