WHY IS ATPG EASY?

by

Philip Chong, Mukul R. Prasad and Kurt Keutzer

Memorandum No. UCB/ERL M99/9

25 February 1999

WHY IS ATPG EASY?

by

Philip Chong, Mukul R. Prasad and Kurt Keutzer

Memorandum No. UCB/ERL M99/9

25 February 1999

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Why is ATPG easy?

Philip Chong Mukul R. Prasad Kurt Keutzer
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

25th February 1999

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 [IS75]. This work is one of the first attempts to reconcile these seemingly disparate results. We introduce the concept of cut-width of a circuit and characterize the complexity of ATPG in terms of this property. We provide theoretical results and empirical evidence to argue that an interestingly large class of practical circuits have cut-width characteristics which ensure 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 proved to be effective tools of logic optimization [DMSV88, CE93] and have recently found application in verification techniques as well [Bra93, KSL95]. It has been known for more than two decades that the ATPG problem is NP-complete [IS75]. 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 [WP79] 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** [Yan91] and **ISCAS85** [BF85] benchmark suites, using **TEGUS** [SBSV96], an ATPG tool based on a Boolean satisfiability (SAT) formulation. 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 [Lar89] and later developed by Stephan et al. [SBSV96]. The reason for this choice is twofold. First, SAT is a well researched problem (see [GPFW97] for an excellent survey), and offers a clean and general framework for analyzing a wide range of search

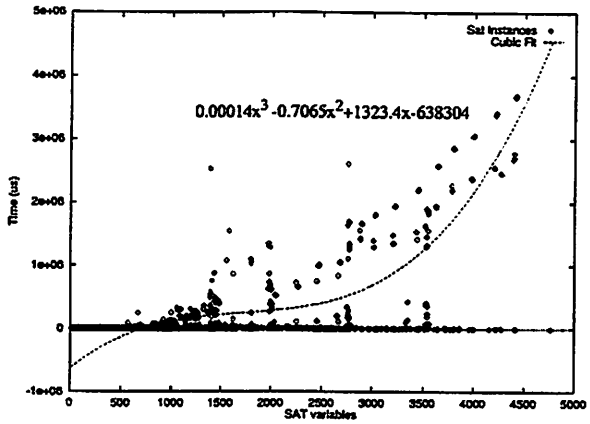


Figure 1: Results of TEGUS on ATPG-SAT instances

techniques and heuristics. Second, SAT is a solution platform for a large number 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

Definition 2.1 Single Stuck-at-fault: *Given a Boolean network C [BRSVW87], 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 $\mathcal{B} \in \{0,1\}$).*

Definition 2.2 Faulted Circuit: *Given a circuit C and a single-stuck at fault $\psi(X, \mathcal{B})$, denote by C_ψ the circuit C with the fault ψ operative i.e.* fault-net *X asserted to value $\mathcal{B}$.*

Definition 2.3 The ATPG Problem: *Given a Boolean network C and a* single stuck-at fault *ψ, the* ATPG *problem $ATPG(C, \psi(X, \mathcal{B}))$ has the answer* YES *if and only if there exists an assignment of Boolean values to the primary inputs of C (and also 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 also C_ψ) have complementary logic values. Such a Boolean assignment, if one exists is said to be a test for the fault ψ. Otherwise the fault is said to be untestable.*

Definition 2.4 CIRCUIT-SAT: *Given a Boolean circuit C, the answer to the problem $CIRCUIT-SAT(C)$ is 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.*

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 $C_1, C_2, \ldots C_m$. Each clause C_i is a set of k_i literals $l_1, \ldots l_{k_i}$. A literal is an instance of a variable or its complement. f is interpreted as the conjunction (logical AND) of the clauses $C_1, C_2, \ldots C_m$, 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 $f = \{C_1, C_2\}$, where $C_1 = \{x_1, \overline{x_2}\}$ and $C_2 = \{x_2, x_3\}$.

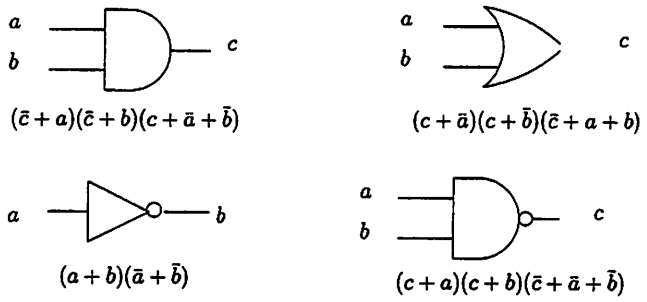


Figure 2: SAT formulas for simple gates

Definition 2.5 SAT: *The Boolean satisfiability problem $SAT(f)$ 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.

SAT and $CIRCUIT-SAT$ are well known instances of *NP-Complete* problems [GJ79]. 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.

- C_ψ^{fanout}: The sub-circuit of C_ψ corresponding to the *transitive-fanout* of X in C_ψ. C_ψ^{fanout} has as its primary inputs, gates on the *fault-boundary* of C (Figure 3).
- C_ψ^{sub}: The sub-circuit of C containing all gates, inputs and outputs in the transitive fanin of the transitive fanout of the fault-point X.
- C_ψ^{ATPG}: The circuit corresponding to the pairwise *XOR* of the outputs of C_ψ^{sub} and C_ψ^{fanout}. C_ψ^{fanout} derives its inputs from appropriate signal points in C_ψ^{sub}.

The set of all satisfying assignments for the *CIRCUIT-SAT* instance C_ψ^{ATPG} gives precisely the set of all input vectors that test the fault ψ (see [Lar89] for details). Thus the ATPG problem $ATPG(C, \psi(X, C))$ can be formulated as an instance of Boolean satisfiability, namely $CIRCUIT-SAT(C_\psi^{ATPG})$. Henceforth, we will refer to this special instance of the satisfiability problem as *ATPG-SAT*.

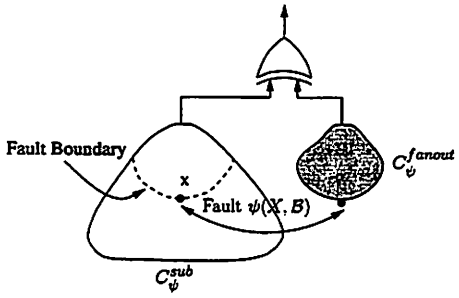


Figure 3: Circuit C_ψ^{ATPG} used for ATPG-SAT

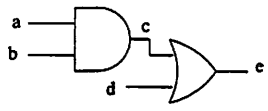


Figure 4: $CIRCUIT - SAT$ instances are not q-Horn

Definition 2.6 ATPG-SAT *refers to the SAT instance corresponding to an ATPG problem. Specifically,* $ATPG - SAT(C, \psi)$ *refers to the SAT formula for testing the single stuck-at fault* ψ *on circuit* C.

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 [SBSV96] enforces this latter condition for exactly this reason.

3 Applying Existing Techniques

There are a number of ways of exploring the easiness of ATPG. One approach is to try and see what classes of ATPG problems can be solved in polynomial time, i.e. whether there exists an algorithm that can solve those ATPG instances in polynomial time. Alternatively, one could start with the SAT formulation of ATPG and attempt to show through either a worst case or average case analysis that some set of ATPG-SAT instances are easy to solve.

Here we discuss three such approaches based on an application 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 attempt to circumvent this shortcoming by developing a characterization of the complexity of ATPG-SAT in terms of 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. [BCH90] 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. We argue that it is highly unlikely that any ATPG-SAT instances of practical significance lie in one of the polynomial SAT classes.

Recall that ATPG-SAT are specialized instances of CIRCUIT-SAT. Consider the circuit of Figure 4. Even this simple CIRCUIT-SAT instance does not fall into the class of *q-Horn* formulas (Appendix D gives a detailed proof). Thus it appears that the easiness of ATPG-SAT cannot be explained solely by the intrinsic easiness of the SAT formulas. The answer lies in relating the solution process of SAT to the properties of the circuits from which they were derived. We investigate this further in Section 4.

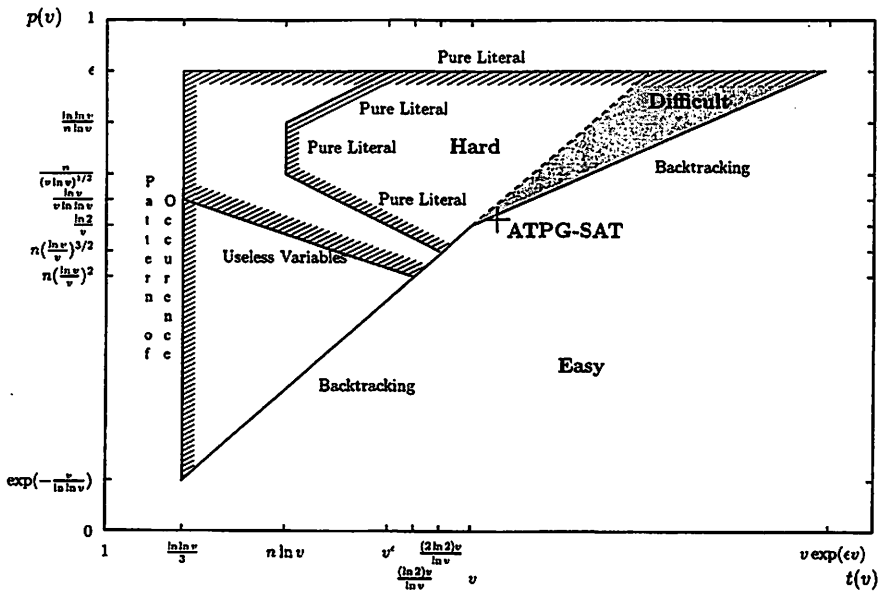


Figure 5: Average Time Analysis (from [PB87])

3.2 k-bounded Circuits

Fujiwara [Fuj88] introduced the notion of *k-bounded* circuits and showed that ATPG can be efficiently performed on this class of circuits[1]. 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

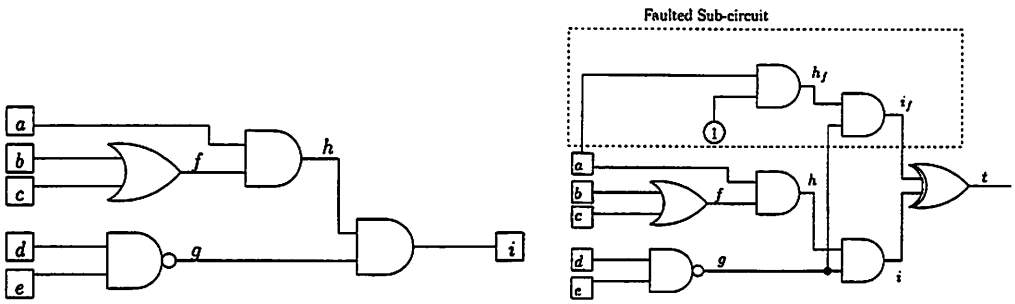
Another approach of assessing the complexity of ATPG-SAT would be to perform an average running time analysis on the the population of ATPG-SAT instances. A number of average-time analyses have already been done for different models of SAT formulas and algorithms; for our analysis we use the one presented in [PB87].

Consider SAT instances generated by the following model. Let v be the number of variables in a SAT instance. Let $p(v)$ be the probability that a given literal appears in a given CNF clause, and let $t(v)$ be the number of CNF clauses which appear in the SAT instance. A given pair of functions, $p(v)$ and $t(v)$ characterize a family of SAT instances.

Figure 5 is taken from [PB87] and depicts the space of SAT problems given the $p(v)$ and $t(v)$ functions. The lines delimit areas of SAT problems which have a known polynomial *average running time* algorithm and are labeled with the name of the associated algorithm. The areas labeled "Hard" and "Difficult" characterize the problems for which there is no known polynomial *average running time* algorithm.

As per the analysis given in Appendix C, under suitable assumptions on the topology of circuits, the class of corresponding CIRCUIT-SAT formulas can be shown to have $p(v) = \frac{7}{6v}$ and $t(v) \geq 1.963v$. ATPG-SAT instances, being specialized instances of CIRCUIT-SAT, are a subset of this class. The point depicting this class is shown marked with **ATPG-SAT** on Figure 5. This point lies just inside the region marked "Easy" for

[1]This algorithm is exponential in k, but for constant k, the algorithm is polynomial in the circuit size.



(a) CIRCUIT-SAT example

(b) ATPG circuit example

Figure 6: Example Circuits

backtracking algorithms and would seem to imply that ATPG-SAT instances have polynomial average running time. However, this conclusion is flawed, since the class of ATPG-SAT instances represent only a fraction of the family of instances represented by that point. It could well be the case that ATPG-SAT instances exhibit an exponential runtime whereas the remaining instances in this class are fairly easy to solve and thereby responsible for the polynomial average runtime of the family as a whole. Thus this analysis is inconclusive at best.

4 Analysis of *ATPG-SAT*

A number of approaches for solving *Boolean satisfiability* have been proposed in the literature [GPFW97]. 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* [GPFW97]. 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 6(a) as our working example. As per the discussion in Section 2 the CIRCUIT-SAT instance corresponding to this circuit is:

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

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 6(b).

4.1 Caching-Based Backtracking for CIRCUIT-SAT

Most popular backtracking based algorithms, especially those proposed in the CAD literature [SBSV96, SS96], have some feature built in to reduce *conflicts* during backtracking. This may be in the form of a pre-processed set of *global implications* [SBSV96] or in the form of generating and storing *conflict-induced clauses* by "learning" from conflicts [SS96]. Our *caching based version* of simple backtracking is a 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 7 shows an example run of this algorithm on Formula 4.1. The variable ordering $A = (b < c < f < a < h < d < e < g < i)$ 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*[2] (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.[3] $\mathcal{R}(f)$ 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,

$$\mathcal{R}(f) = O(\mathcal{F}(\mathcal{P}_h(V))) \tag{4.2}$$

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 $\mathcal{P}_h(V)$

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

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

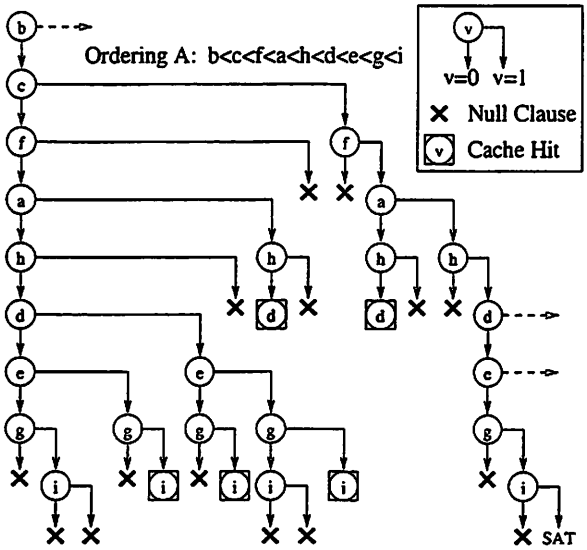


Figure 7: Caching-based backtracking for Formula 4.1

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 hyper-edges, 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.

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:** that each hyperedge e of G is denoted by the set of vertices spanned by that hyperedge.) 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 8 illustrates the notion of cut-width on the example circuit from Figure 6(a), using two different variable orderings, A and B. Ordering A, which was used for the backtracking tree example of Figure 7, 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 disjoint 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*

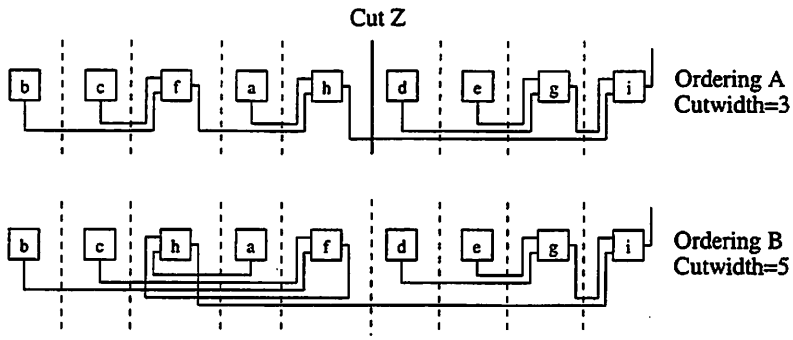


Figure 8: Example cut-widths for the circuit of Figure 6(a)

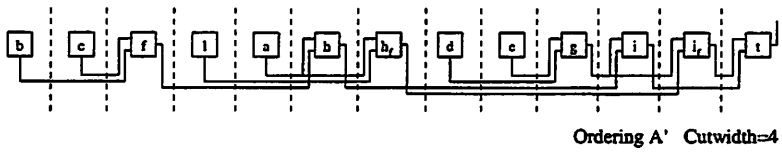


Figure 9: Example cut-width, ATPG circuit of Figure 6(b)

δ_{V_C} is denoted by $\mathcal{F}(\delta_{V_C})$ and can be bounded as:

$$\mathcal{F}(\delta_{V_C}) \leq 2^{2k_{fo}|(\delta_{V_C}, \overline{\delta_{V_C}})|} \tag{4.3}$$

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

Proof: See Appendix A.1. ∎

The usefulness of this result 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 could be potentially much smaller. For example consider the cut $(\delta_V, \overline{\delta_V})$ on the circuit of Figure 6(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 8. 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 the cut-width of a circuit 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 $n = |V_C|$.*

Proof: See Appendix A.2. ∎

From the above result 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 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*

$$W(C_\psi^{ATPG}, h_\psi) \leq 2 \cdot W(C, h) + 2 \tag{4.4}$$

Proof: See Appendix A.3. ∎

Figure 9 illustrates this result on our example ATPG circuit from Figure 6(b). As shown in Figure 8 the circuit of Figure 6(a) has a cut-width of 3 under ordering A (Figure 7). The ordering A' can be derived (see Appendix A.3) from this to yield a cut-width of 4 for the ATPG circuit of Figure 6(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, $CIRCUIT - SAT(C) = CIRCUIT - SAT(C_1) \vee \ldots \vee CIRCUIT - SAT(C_p)$.

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:

$$W(C, H) = \max_{i \in \{1,2,\ldots p\}} W(C_i, h_i) \tag{4.5}$$

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 CIRCUIT-SAT(C), (based on Algorithm 1) can be bounded as:

$$\mathcal{R}(f(C)) = O(p \cdot n_{max} \cdot 2^{2k_{fo}W(C,H)}), \text{ where } n_{max} = \max_{i \in \{1,2,\ldots p\}} |V_{C_i}|. \tag{4.6}$$

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*

$$W(C_\psi^{ATPG}, H_\psi) \leq 2 \cdot W(C, H) + 2 \tag{4.7}$$

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.

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.*

Proof: Applying Lemma 4.3 and Equation 4.6 to the definition of log-bounded-width above (Definition 5.1), we can use Algorithm 1 to solve the instance $ATPG - SAT(C, \psi)$ in time polynomial in $|C_{\psi}^{sub}|$. Since $|C_{\psi}^{sub}| \leq |C|$, then the running time is polynomial in $|C|$ as well. ∎

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

Proof: See Appendix B.1. ∎

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

Proof: See Appendix B.2. ∎

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, (which log-bounded-width has been shown to capture). 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

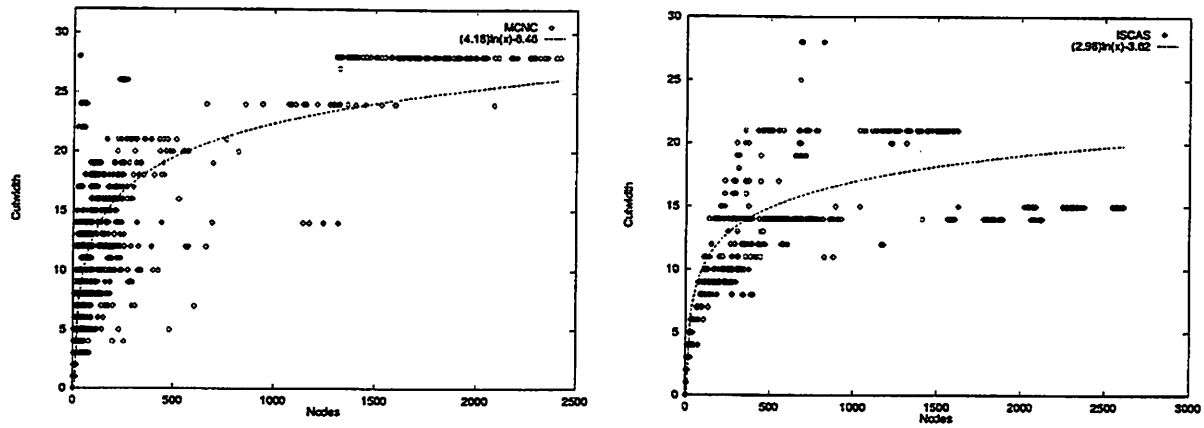
From the discussion on circuit *cut-width* it is apparent that cut-width is intrinsically linked to the topology of the circuit. Thus, when a class of circuits can be described in terms of suitable topological characteristics, it is possible to derive the cut-width properties of that class, and reason about the asymptotic complexity of ATPG-SAT, as was done for *log-bounded-width circuits* and *k-bounded circuits* above. However, practical designs are usually not specified in such a manner. Moreover, extracting common topological characteristics from a set of arbitrary circuit designs is a non-trivial task and beyond the scope of this paper. Thus, we have instead performed an empirical study of cut-width for a set of circuits. The study is organized in two parts. In the first part we study circuits in the **MCNC91** and **ISCAS85** multi-level combinational benchmark suites and estimate their cut-widths. In the second part of this study, we examine the growth of circuit cut-width compared to the size of the circuit. [HGRC96] presents a system which extracts topological properties from a given circuit and generates arbitrarily large circuits which have similar characteristics. Using this scheme we generate a "family" of circuits from a given circuit and then examine the cut-width properties of this family.

5.2.1 Experimental Setup

The key element of our experimental setup is a mechanism to measure the *cut-width* of a single-output circuit C. This can then be used to derive the cut-width of a multi-output circuit. From the definition, the minimum cut-width is simply the value of the *max-cut* obtained under a *min-cut linear arrangement* [GJ79] of C. Since the min-cut linear arrangement (MLA) problem is known to be NP-complete, we use a well-known algorithm [Hoc97] to approximate the MLA, and hence estimate the cut-width for a given circuit. The approximation algorithm generates a placement based on recursive mincut bipartitioning, until the partitions are sufficiently small and then performs an exact MLA for each of these partitions. We used the **HMETIS** package [KAKS97] from the University of Minnesota to perform the bipartitioning.

For a set of circuits under study the ease of ATPG problem can be gauged by analyzing the difficulty of the ATPG-SAT instances generated by them. To this end, we generated one data-point for each potential fault in each circuit. For a given fault ψ in circuit C, the 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 of the SAT instance $ATPG - SAT(C, \psi)$ (in terms of the number of variables) and the cut-width of this circuit is indicative of the complexity of solving this instance (as per Equation 4.6 and Lemma 4.3).

Note that we do not necessarily advocate using MLA as a method for solving ATPG. Certainly, one could compute (or approximate) the MLA for a circuit to obtain a suitable variable ordering for Algorithm 1. However, this is not the motivation for the experiment; we only use the MLA here to obtain the cut-width characteristics of our circuits and show that we expect these circuits to be easily testable.



(a) MCNC91 logic benchmarks (11315 datapoints)

(b) ISCAS85 benchmarks (7389 datapoints)

Figure 10: Cut-width results for benchmark circuits

5.2.2 Study of Existing Benchmark Suites

In studying the cut-width properties of the **MCNC91** and **ISCAS85** benchmark circuits, it became clear that the individual circuits have different structural properties. Some of these circuits have nodes with fanin of a dozen or more inputs, while others are composed solely of two-input AND gates and inverters. Similarly, some of the benchmark circuits have nodes which implement complex functions, while others use only simple AND/OR gates.

These differences probably would not exist in actual implementations of circuits; 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 the corresponding SAT formulas become much easier to derive. Thus, in order to bring more uniformity to the circuits and to more closely emulate the actual ATPG process, we mapped the benchmark circuits to three or fewer input AND/OR gate networks, allowing inversions. The `tech_decomp` procedure from the logic optimization package **SIS** [S+98] 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 sub-circuit C_{ψ}^{sub}. Figure 10(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 10(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 tool.[4] We expect C6288 to have a large cut-width, as it has been traditionally known as a difficult example to test in the CAD community. In any event, 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 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).

[4] We used HMETIS in a mode which fixed some vertices to specific partitions. These circuits generated too many fixed vertices for HMETIS to handle.

5.2.3 Study of Generated Circuits

Using the existing benchmark suites limits the size of the circuits which we analyze. Ideally, we would like to have a large range of circuit sizes so that we can examine the growth of the cut-width with larger circuits. To this end, we adopted the approach of synthesizing circuits using the programs `circ` and `gen`, described in [HGRC96]. These programs respectively extract important characteristics from existing circuits and generate random circuits with these same characteristics. Note that these programs do not generate any useful circuits in that the function of each node is undefined. However, this is of no concern to us, as we are only interested in the structure of the circuit, in particular the cut-width corresponding to each possible fault point.

Our goal is to generate circuits with structures similar to the original MCNC benchmark circuits but with varying sizes. To this end, we use `circ` to find the characteristics for a selection of the benchmark circuits, and then scale these parameters before using gen. In particular, we change only the number of nodes in the circuit, the number of primary inputs, the number of primary outputs, and the number of edges (nets) in the circuit. We do not change the depth of logic, as this parameter is bounded for practical circuits to meet delay constraints. We also do not change the distribution of delays, fanouts or edge lengths in the circuit; [HGRC96] identifies these parameters as important in characterizing the structure of a circuit, and we wish to obtain circuits structurally similar to the original benchmarks.

For each benchmark circuit used here, we used `circ` and `gen` to generate a "family" of circuits ranging from 1,000 nodes to 6,000 nodes. For each generated circuit, we take each possible fault ψ, find the induced sub-circuit C_{ψ}^{sub}, and calculate the size of this sub-circuit and its minimum cut-width; this is exactly the same procedure as used with the original benchmark circuits.

Figures 11(a) through 12(b) show the cut-width versus circuit size for four different families of circuits generated as described above. We used a least-squares method [Mey75] to fit three different curves to the data: linear ($y = ax + b$), logarithmic ($y = a\log(x) + b$) and power ($y = ax^b$) curves, where y is the cut-width and x is the number of nodes. Of the three curves, the log curve gives the smallest square error for all four circuit families; the best-fit curves are shown in the figures.

For the generated circuits, there appears to be more spread of the data away from the fitted curves than for the existing benchmarks. We suggest that this is due to the randomness of the circuits built by gen; the hand-designed benchmark circuits have apparent regularities (e.g. some benchmark circuits have identical cone sizes across all outputs) which cannot be captured by gen, and these regularities help reduce the variance of our data as compared to random circuits.

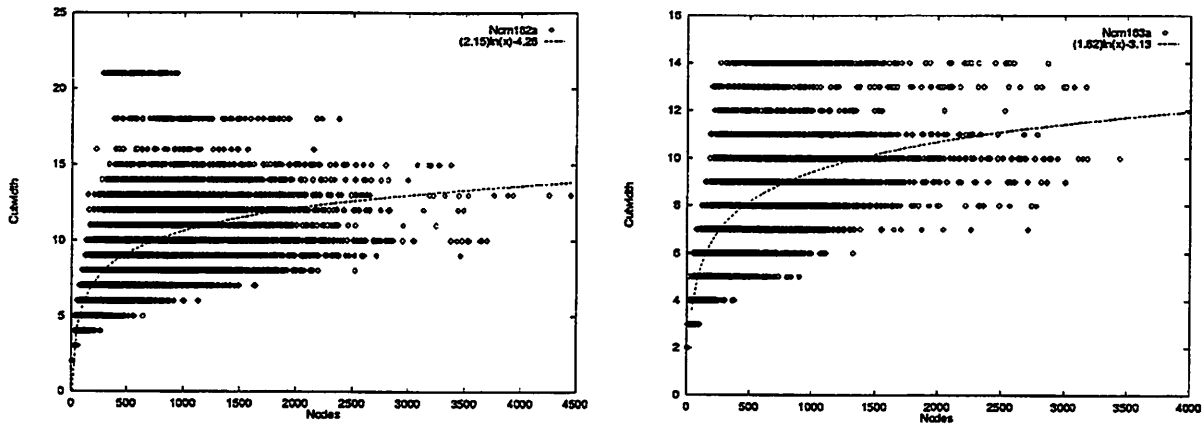
In any case, the growth of the cut-width is sub-linear with the size of the circuit, which indicates that the complexity of ATPG on typical circuits grows sub-exponentially with the size of the problem.

6 BDDs and CIRCUIT-SAT

The concept of *circuit-width* has been used by researchers [Ber91, McM92] to obtain upper bounds on the size of BDDs representing the circuit function. At first glance our treatment of circuit cut-width would seem to bear a striking similarity to these results. However, our results have no direct relationship to the BDD bounds. We discuss this aspect in some detail below and conclude that neither result subsumes the other, each useful in its own domain.

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 building a BDD for the circuit and doing a "0" check on the BDD. 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.

Berman [Ber91] gave a bound on the BDD size, for any *topological ordering* of the circuit elements. This result was extended by McMillan [McM92] 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



(a) cm162a family (37192 datapoints)

(b) cm163a family (37776 datapoints)

Figure 11: Cut-width results for generated circuits

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.

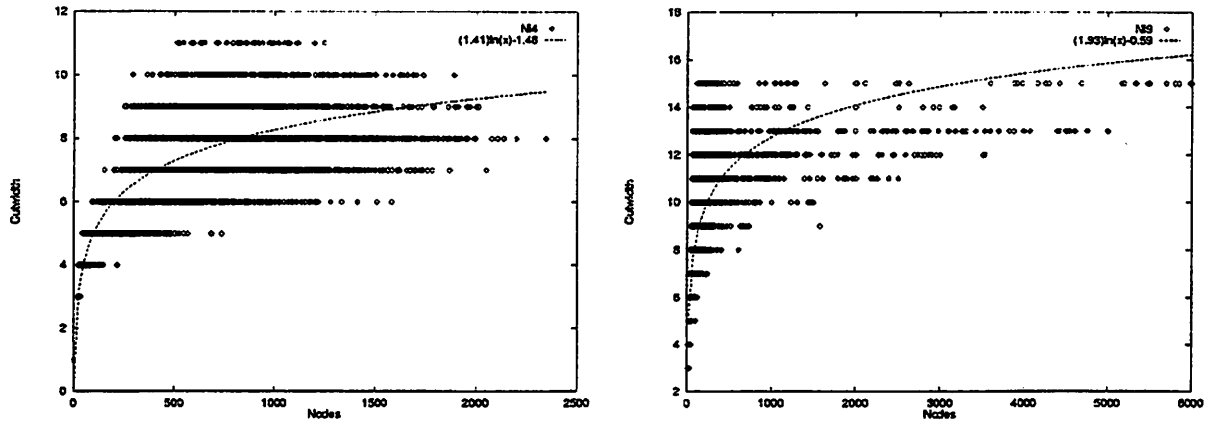
The explanation for these discrepancies lies in the following differences between BDDs and CIRCUIT-SAT formulas. BDDs represent the intrinsic nature of a Boolean function, independent of the specific hardware implementation, while CIRCUIT-SAT formulas (as per the construction of Section 2) are in one to one correspondence with the circuit topology. The result of [McM92] bounds the BDD size by bounding the number of possible multi-output functions that a certain sub-circuit of the original circuit could compute. Our proof technique however, treats the SAT formula as a string encoding the circuit topology and tries to bound the number of distinct sub-strings that can be generated from a partial truth assignment to the CIRCUIT-SAT variables. 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 [Lar89], 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 *undirected 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



(a) i4 family (38297 datapoints)

(b) i9 family (37215 datapoints)

Figure 12: Cut-width results for generated circuits

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.

The caching based variant of backtracking and the results on min-cut linear arrangement have been used as a proof methodology in this work. However, they can be extended into a practical tool for performing ATPG with suitable modifications. As a first step, the min-cut arrangement based variable ordering could be used to replace the naive ordering heuristics used by current ATPG-SAT solvers.

Practical SAT algorithms [Lar89, SBSV96, SS96] employ a host of other heuristics on top of backtracking, in their respective solution engines. Quantifying the benefits of these additional heuristics could offer further insight into the easiness of ATPG. Additionally, a number of circuit based CAD problems *viz.* false path eliminating static timing analysis, delay fault testing and logic verification use satisfiability as the core solution engine. Since our analysis characterizes SAT in terms of circuit properties, in a more general setting the results could potentially be applicable to these domains as well.

Appendix

A Proofs for Section 4

A.1 Proof of Lemma 4.1

Consider the set of $2^{|\delta_{V_C}|}$ possible different Boolean assignments to the variables δ_{V_C}. Only a fraction of these produce consistent sub-formulas. It is these assignments that we consider here. These assignments partition the clauses of $f(C)$ into three *disjoint* categories.

- Clauses all of whose variables are part of δ_{V_C}. Every consistent sub-formula of $f(C)$ has these clauses satisfied.
- Clauses all of whose variables are part of $\overline{\delta_{V_C}}$. These clauses are unaffected by any assignment to the variables δ_{V_C} and thus appear unaltered in *any* consistent sub-formula.
- Clauses part of whose variables are in $\overline{\delta_{V_C}}$ and part in δ_{V_C}. We call these clauses *injured clauses.*

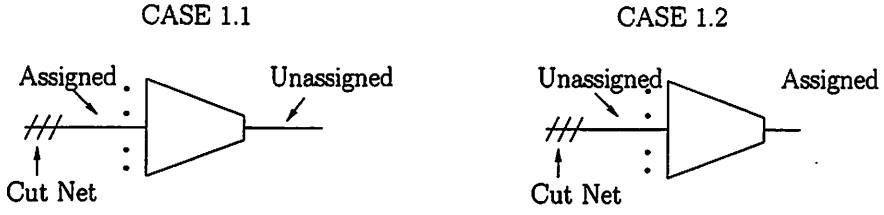


Figure 13: Case 1 for generating injured clauses

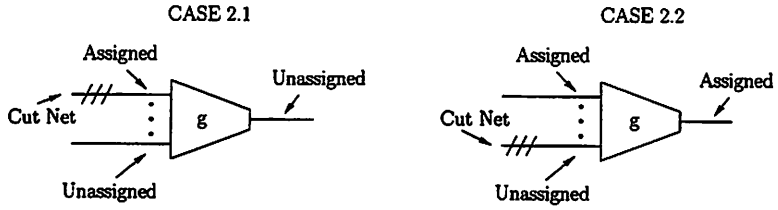


Figure 14: Case 2 for generating injured clauses

From the above categorization it is clear that different consistent sub-formulas of the set $\mathcal{F}(\delta_{V_C})$ differ only in the configuration of the *injured clauses*. Furthermore, under *any* assignment to variables δ_{V_C} an injured clause can have only *two* different configurations. Consider a typical injured clause $C = (l_1 + l_2 + \ldots + l_i + l_{i+1} + \ldots + l_k)$ such that the variables corresponding to literals $l_1 + l_2 + \ldots + l_i$ are part of δ_{V_C} and the variables corresponding to literals $l_{i+1} + \ldots + l_k$ are part of $\overline{\delta_{V_C}}$. Under any assignment to the variables δ_{V_C}, C takes one of the two configurations, $(l_{i+1} + \ldots + l_k)$ or 1 (i.e. it has been satisfied). Thus we can bound

$$\mathcal{F}(\delta_{V_C}) \leq 2^{(\#\text{ injured clauses})} \tag{A.1}$$

An upper bound can be obtained for the number of injured clauses based on the following arguments. Every injured clause must contain *at least* one assigned variable and at least one unassigned variable. Moreover, a pair of variables occur in a common clause only under one of the following two cases:

- **Case 1:** They form an input-output pair for a gate (see Figure 13). For this pair to produce an injured clause either the input variable is assigned and the output unassigned or vice-versa. In both these cases, the input net falls in the cut $(\delta_{V_C}, \overline{\delta_{V_C}})$.
- **Case 2:** They form a pair of "sibling" inputs for a common gate g (see Figure 14). As before, they can be responsible for an injured clause if and only if one of them is assigned and the other unassigned. Additionally, the output of g can be either assigned or unassigned. In either case, from the clause construction of Figure 2, it is clear that every injured clause that these "siblings" participate in already contains a pair of variables that have been counted in Case 1 (namely the output of g and the input that differs from the output in assignment status). Thus, this case is subsumed by Case 1.

From the above case analysis it is evident that *every* injured clause can be associated with a cut-net and also that Case 1 can account for all injured clauses. Since the network is fanout-bounded by k_{fo}, each cut net can fan out to k_{fo} gates and therefore produce at most k_{fo} instances of Case 1. Moreover, since the network is composed of primitive gates only, a given pair of variables can occur in at most two common clauses (see Figure 2). Thus *each cut net* can account for at most $2k_{fo}$ injured clauses. Hence,

$$\text{Number of injured clauses} = 2k_{fo}|(\delta_{V_C}, \overline{\delta_{V_C}})| \tag{A.2}$$

Applying this result to Equation A.1 the bound on $\mathcal{F}(\delta_{V_C})$ follows.

■

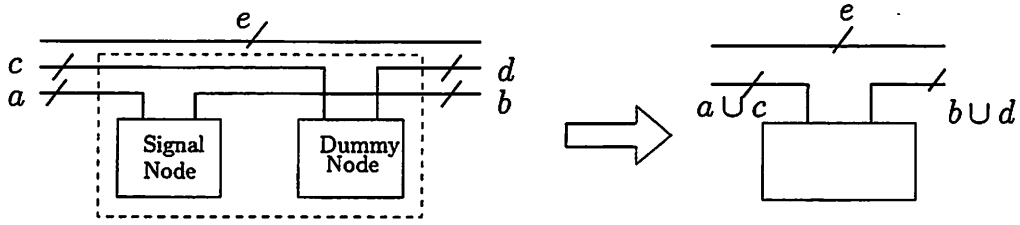


Figure 15: Proof for ATPG circuit width vs. original circuit width

A.2 Proof of Theorem 4.1

To prove the result, we derive a bound on $\mathcal{F}(\mathcal{P}_h(V_C))$ and then apply Equation 4.2. Recall that $P_h(V_C) = \{\delta_{V_C} | \delta_{V_C} \subseteq V_C$, and δ_{V_C} is a prefix of the ordering $h\}$. Therefore $|\mathcal{P}_h(V_C)| = |V_C| = n$.

$$\begin{aligned}
\mathcal{F}(\mathcal{P}_h(V_C)) &\leq \sum_{\delta_{V_C} \in \mathcal{P}_h(V_C)} \mathcal{F}(\delta_{V_C}) \\
&\leq n \cdot \max_{\delta_{V_C} \in \mathcal{P}_h(V_C)} \mathcal{F}(\delta_{V_C}) \\
&\leq n \cdot \max_{\delta_{V_C} \in \mathcal{P}_h(V_C)} 2^{2k_{fo}|(\delta_{V_C}, \overline{\delta_{V_C}})|} \text{ (from Lemma 4.1)} \\
&= n \cdot 2^{2k_{fo}W(C,h)} \text{ (from Definition 4.1)}
\end{aligned}$$

A.3 Proof of Lemma 4.2

Consider the composition of the circuit C_ψ^{ATPG}. It is composed of the two sub-circuits C_ψ^{fanout} and C_ψ^{sub} and a single 2-input XOR gate t (see Figure 3). Note that both C_ψ^{fanout} and C_ψ^{sub} are sub-circuits of C. One may see that given any variable ordering h for V_C, this implies a corresponding ordering h_{sub} for any sub-circuit C_{sub} of C such that $W(C_{sub}, h_{sub}) \leq W(C, h)$.

Given ordering h for V_C, a suitable ordering h_ψ can be constructed as follows. Extract the implied orderings for sub-circuits C_ψ^{fanout} and C_ψ^{sub} from h. Now merge these two together by putting each variable v_f of the faulted sub-circuit C_ψ^{fanout} just after its corresponding "unfaulted variable" v (derived from C_ψ^{sub}). Now construct h_ψ by adding t to the beginning of this merged ordering. To derive the width of C_ψ^{ATPG} under this ordering, consider the following.

The gate t is a two input gate and can therefore contribute at most 2 to $W(C_\psi^{ATPG}, h_\psi)$ (this accounts for the additive 2 in the expression). Now, for the moment assume that the primary inputs of C_ψ^{fanout} are not fed from fanout points in C_ψ^{sub} but from separate dummy nodes (the dummy nodes are inserted after the corresponding signal nodes in the ordering h_ψ). In this scenario it is easy to see that the width of the resulting circuit is at most $2 \cdot W(C, h) + 2$. Merging the dummy nodes with the corresponding signal nodes does not increase the cut-width of the resulting circuit (see Figure 15). Hence the required result follows.

■

B Proofs for Section 5

B.1 Proof of Lemma 5.2

Consider a k-ary tree $\mathcal{T}$ over n vertices with root r. For a vertex ordering, take the variables by using depth-first search starting from the root; at each node visit the children in increasing order of the size of the sub-trees rooted at each child. Under this ordering $\mathcal{T}$ has a max-cut of at most $(k-1)\log(n)$ edges. This can be proved by induction on n. For the base case $n = 1$, the cut is zero.

For larger n, there are two cases. Let s_i, $1 \leq i \leq k$ be the subtrees rooted at the immediate children of r, and let c_i be the size of the max-cut for s_i. For the first case, let all $|s_i| \leq n/2$. Then the max-cut under the given DFS ordering is at most $(k-1)+c$, where $c = \max_i c_i$. By the induction hypothesis, $c \leq (k-1)\log(n/2)$, so the max-cut of $\mathcal{T}$ is at most $(k-1)\log(n)$.

For the second case, for some t, $|s_t| > n/2$. Then this subtree is visited last by the DFS ordering, and so the max-cut of $\mathcal{T}$ by this ordering is at most $\max((k-1)+c, c_t)$, where $c = \max_{i \neq t} c_i$. Since $|s_i| < n/2, i \neq t$, the induction hypothesis gives $(k-1)+c \leq (k-1)\log(n)$, and since $|s_t| < n$, $c_t \leq (k-1)\log(n)$ as well.

Thus the max-cut of $\mathcal{T}$ is at most $(k-1)\log(n)$.

■

B.2 Proof of Theorem 5.1

First consider the graph G consisting of the blocks of a k-bounded circuit; by the non-reconvergence property of k-bounded circuits, the cone for each output of G is a k-ary tree. For each output tree, use the ordering scheme proposed in the proof of Lemma 5.2 to order the blocks of G. Now, within each block order the vertices of the block arbitrarily. Each block can thus increase the max-cut by a factor of at most 2^k. Hence, given Lemma 5.2 for k-ary trees we can conclude an upper bound of $2^k \cdot (k-1)\log(n)$ for the max-cut of a k-bounded circuit.

■

C Average-Time Analysis

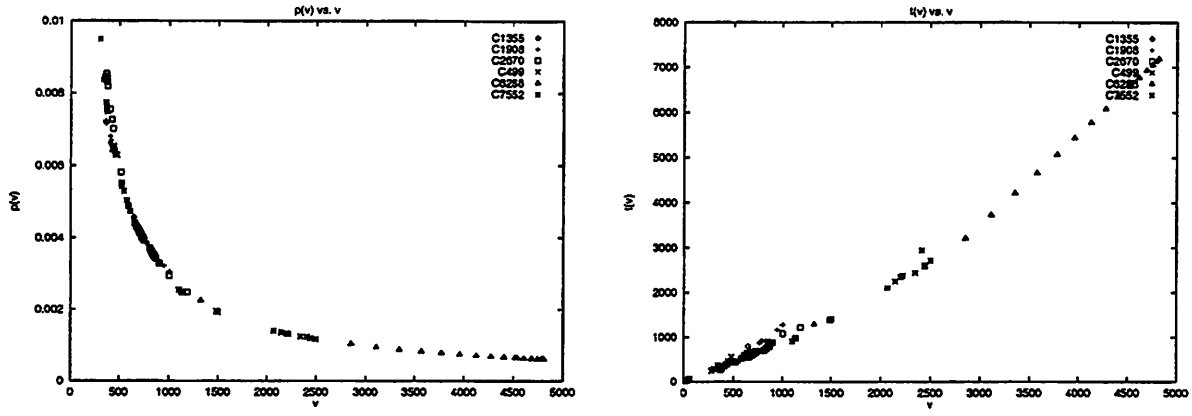
Let v be the number of variables in an SAT instance. Let $p(v)$ be the probability that a given variable appears in a given CNF clause, and let $t(v)$ be the number of CNF clauses that appear in the SAT instance. Consider the *CIRCUIT-SAT* problem applied to a circuit which contains only 2-input AND gates, allowing the inputs and outputs of the gates to have inversions. Note that this is a typical decomposition technique used in the presence of more complex gates used to simplify the construction of the SAT formula. If we construct the SAT formulas for each gate, then a single gate will give rise to exactly three CNF clauses. For instance, if we have a gate for $x = y \cdot z$, then the CNF formula we obtain is $(y + \bar{x})(z + \bar{x})(\bar{y} + \bar{z} + x)$. Suppose we have n gates, and k primary inputs to the circuit; then we have $v = n + k$.

Now consider the size of the clauses generated by this circuit. Two clauses from each gate will have exactly two literals, while the remaining clause will have three literals. Thus the average clause length in the overall SAT formula is $\frac{7}{3}$ literals. Since there are $2v$ possible literals, any given literal has probability $\frac{7}{6v}$ of appearing in any given clause, so $p(v) = \frac{7}{6v}$.

The analysis in [PB87] shows that, if $\lim_{v\to\infty} vp(v) = b \geq \ln 2$, then simple backtracking will give a polynomial average running time when $t(v) \geq (\ln 2 + \epsilon)v\frac{d}{b}$, for some positive ϵ and where d is the solution of $\ln(1+d) + d\ln(1+\frac{1}{d}) = 2b$. Now we have $\lim_{v\to\infty} vp(v) = \frac{7}{6} = b$, and solving for d gives $d = 3.305$. In the limit as $\epsilon \to 0$, we require $t(v) \geq \ln(2)v\frac{d}{b} = 1.963v$ for a polynomial average running time. Note that $t(v) = 3n = 3(v-k)$, so for a polynomial average running time we require $3(v-k) \geq 1.963v$, or $k \leq 0.346v$.

We expect, for practical circuits, that the primary inputs be only a small fraction of the total number of nodes. Since the circuits are mapped to 2-input AND gates and inverters, any interesting function will be mapped to a large number of gates while leaving the number of primary inputs unaffected. Thus we assume $k \leq 0.346v$, as required.

Above, we assumed a formulation based on a *CIRCUIT-SAT* problem; note that an instance of ATPG can be formulated as a *CIRCUIT-SAT* problem by simply duplicating part of the circuit.



(a) $p(v)$ vs. v

(b) $t(v)$ vs. v

Figure 16: Benchmark SAT instances in TEGUS

Figures 16(a) and 16(b) show the values of $p(v)$ and $t(v)$ as measured on SAT instances generated from a number of the ISCAS85 benchmark circuits using TEGUS. Curve fitting indicates $p(v)$ is indeed inversely proportional to v, while $t(v)$ is linear with v.

D CIRCUIT-SAT and the q-Horn Property

We show that the CIRCUIT-SAT formula for Figure 4 is not q-Horn. This SAT formula, f, is given by:

$$f = e \wedge f_1 \wedge f_2$$

where

$$f_1 = (e + \overline{c})(e + \overline{d})(\overline{e} + c + d)$$
$$f_2 = (\overline{c} + a)(\overline{c} + b)(c + \overline{a} + \overline{b})$$

We prove that f is not *q-Horn* by proving that its *complexity index*, $Z(f)$ is greater than 1. As defined in [BCHS94] the *complexity index* of a CNF ϕ is the optimal value $Z(\phi)$ of the linear programming problem, denoted by $LP(\phi)$

$$Z(\phi) = \min Z$$
$$\text{such that}$$
$$\sum_{i \in P_k} \alpha_i + \sum_{i \in N_k} (1 - \alpha_i) \leq Z \quad (k = 1 \ldots m) \text{ and}$$
$$0 \leq \alpha_i \leq 1 \quad (i = 1 \ldots n)$$

where P_k is the set of positive literals in clause k and N_k is the set of negative literals in clause k.

It is shown in [BCH90] that all *q-Horn* instances have a complexity index of at most 1. We prove by contradiction that $Z(f) > 1$. Suppose that $Z(f) \leq 1$
Consider $LP(f_1)$

$$1+e-c \leq Z$$
$$1+e-d \leq Z$$
$$1-e+c+d \leq Z$$
$$0 \leq c \leq 1, \quad 0 \leq d \leq 1, \quad 0 \leq d \leq 1$$

Substituting $Z = 1$ in the above and solving gives

$$c = d = e = 0 \tag{D.1}$$

Now, consider $LP(f_2)$

$$1+a-c \leq Z$$
$$1+b-c \leq Z$$
$$2+c-a-b \leq Z$$
$$0 \leq a \leq 1, \quad 0 \leq b \leq 1$$

Substituting $Z = 1$ and the solution of $LP(f_1)$ from Equation D.1 in $LP(f_2)$ gives no solution.

$$\Rightarrow Z(f_1 \wedge f_2) > 1$$
$$\Rightarrow Z(f) > 1$$

■

Acknowledgments

This work originated as a class project in algorithms at U.C. Berkeley under the guidance of Christos Papadimitriou, to whom we are grateful for motivation and guidance. Olivier Coudert provided helpful advice during the early stages of this work. Thanks to George Karypis and the rest of the **HMETIS** team, and Mike Hutton and the `circ/gen` team, for providing key software components and to Yuji Kukimoto for useful feedback on a preliminary version of this work. This work was supported in part by California MICRO and NSERC Canada.

References

[BCH90] 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.

[BCHS94] E. Boros, Y. Crama, P. L. Hammer, and M. Saks. A Complexity Index for Satisfiability Problems. *SIAM Journal of Computing*, 23(1):45–49, February 1994.

[Ber91] C. Leonard Berman. Circuit Width, Register Allocation and Ordered Binary Decision Diagrams. *IEEE Transactions on Computer-Aided Design*, 10(8):1059–1066, August 1991.

[BF85] F. Brglez and H. Fujiwara. A Neural Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *International Symposium on Circuits and Systems*, June 1985.

[Bra93] D. Brand. Verification of Large Synthesized Designs. In *IEEE International Conference on Computer Aided Design*, pages 534–537, 1993.

[BRSVW87] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: A Multiple-Level Logic Optimization System. *IEEE Transactions on CAD/ICAS*, CAD-6(6):1062–1082, November 1987.

[CE93] 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, June 1993.

[DMSV88] 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.

[Fuj88] Hideo Fujiwara. Computational Complexity of Controllability/Observability Problems for Combinaitonal Circuits. In *International Symposium on Fault-Tolerant Computing*, pages 64–69, June 1988.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[GPFW97] Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the Satisfiability (SAT) Problem: A Survey. *DIMACS Series in Discrete Mathematics and Computer Science*, 35:19–151, 1997.

[HGRC96] Michael Hutton, J. P. Grossman, Jonathan Rose, and Derek Corneil. Characterization and Paramterized Random Generation of Digital Circuits. In *33rd Design Automation Conference*, pages 94–99, 1996.

[Hoc97] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1997.

[IS75] O. H. Ibarra and S. K. Sahni. Polynomially Complete Fault Detection Problems. *IEEE Transactions on Computers*, C-24(3):242–249, March 1975.

[KAKS97] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. In *34th Design Automation Conference*, pages 526–529, 1997.

[KSL95] 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.

[Lar89] Tracy Larrabee. Efficient Generation of Test Patterns Using Boolean Difference. In *International Test Conference*, pages 795–801, 1989.

[McM92] K. L. McMillan. *Symbolic model checking: An approach to the state explosion problem.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1992.

[Mey75] Stuart L. Meyer. *Data Analysis For Scientists and Engineers.* Wiley and Sons, 1975.

[PB87] Paul W. Purdom and Cynthia A. Brown. Polynomial-Average-Time Satisfiability Problems. *Information Sciences*, 41:23–42, 1987.

[S+98] 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.

[SBSV96] Paul Stephan, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Combinational Test Generation Using Satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(9):1167–1176, September 1996.

[SS96] J. P. Marques Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *International Conference on Computer Aided Design*, pages 220–227, 1996.

[WP79] Thomas W. Williams and Kenneth Parker. Testing Logic Networks and Designing for Testability. *Computer*, pages 9–21, October 1979.

[Yan91] Saeyang Yang. Logic Synthesis and Optimization Benchmarks User Guide, Version 3.0. Technical report, Microelectronics Center of North Carolina, 1991.