Satisfiability Models and Algorithms for Circuit Delay Computation

LUÍS GUERRA E SILVA, JOÃO MARQUES-SILVA, and L. MIGUEL SILVEIRA
IST/Technical University of Lisbon, INESC ID/Cadence European Labs, Portugal
and
KAREM A. SAKALLAH
University of Michigan

The existence of false paths represents a significant and computationally complex problem in the estimation of the true delay of combinational and sequential circuits. In this article we conduct a comprehensive study of modeling circuit delay computation, accounting for false paths, as a sequence of instances of Boolean satisfiability. Several path sensitization models and delay models are studied. In addition we evaluate some of the most competitive Boolean satisfiability algorithms seeking to identify which are best suited for solving circuit delay computation problems. Finally, realistic delay modeling (taking into account extracted interconnect delays and fanout data) is considered in order to experimentally evaluate the complexity of solving real-world instances.

Categories and Subject Descriptors: D.6.3 [**Logic Design**]: Design Aids—*Verification*

General Terms: Algorithms, Verification

Additional Key Words and Phrases: Boolean satisfiability, circuit delay computation, delay modeling, false path, timing analysis

1. INTRODUCTION

Recent years have seen an ever-increasing need for more accurate delay estimation methodologies in digital circuits, in particular due to the decisive role that delay estimation plays in determining limiting operating clock frequencies. A key problem associated with circuit delay estimation is the existence of false paths, which cause straightforward and efficient topological path analysis procedures to yield potentially conservative delay estimates. Unfortunately, for reasons hard to assess and explain, current high-level synthesis systems [Bergamaschi 1991] are prone to generate circuits with many false paths. Consequently, there is a growing need for correctly identifying such paths to guide performance optimization in a computationally efficient manner.

Authors' addresses: L. Guerra e Silva, J. Marques-Silva, L. M. Silveira, INESC ID, Rua Alves Redol 9, 1000 Lisboa, Portugal; email: lgs@algos.inesc.pt; K. A. Sakallah, University of Michigan, Ann Arbor, MI.

In contrast with topological delay estimation, solving the false path problem is computationally hard, being an NP-complete problem [McGeer 1989]. Research work on false paths has been extensive and, among others, several promising modeling and algorithmic approaches have been proposed [Ashar et al. 1993; Chen and Du 1991; Devadas et al. 1993; McGeer and Brayton 1991; Marques-Silva and Sakallah 1994; Yalcin and Hayes 1995]. Despite this research effort, we believe that a comprehensive and unified computational study of different models and algorithms for solving the false path problem is still missing. In this article we propose to partially solve this problem by studying a set of path sensitization criteria, under the assumption of floating mode circuit operation. This work is undertaken within a unified framework for solving the false path problem, which is based on Boolean satisfiability (SAT). Furthermore, we explore more realistic delay modeling within the proposed framework, thus evaluating how SAT-based circuit delay computation is dependent upon the delay model considered.

The computational study described here is necessarily incomplete, since several relevant models and algorithms are not covered. Nevertheless, this study proposes an experimental procedure that can be generalized for those other models. Furthermore, we note that false paths can exist in both combinational and sequential circuits, even though in this article we exclusively consider combinational false paths.

The article is organized as follows. We start by introducing a few definitions related to combinational circuits and the Boolean satisfiability problem. In Section 3 we describe how to formulate the circuit delay computation problem as an instance of Boolean satisfiability. This section closely follows the work of McGeer et al. [1991], but a significantly simpler approach is used to derive the SAT models for the viability sensitization criterion [McGeer and Brayton 1991]. In addition, the SAT modeling approach of McGeer et al. [1991] is shown to be easily extended to other path sensitization criteria under the floating mode of operation, namely, static sensitization [Benkoski et al. 1987] and the exact sensitization criterion [Chen and Du 1991]. Afterwards, in Section 4, we describe the circuit delay computation algorithm and propose improvements in the delay stepping strategies, which result in significant performance gains over existing procedures. In Section 5, we discuss the use of more realistic delay models in the formulation of the circuit delay computation problem in order to evaluate the computational challenge posed by real-world instances. In Section 6, the experimental procedure is described and experimental results are shown and analyzed. Finally, the conclusions are presented in Section 7.

2. NOTATION AND DEFINITIONS

2.1 Combinational Circuits

In the following we represent a *combinational circuit* as a directed acyclic graph $C = (V, E)$, referred to as the *circuit graph*, composed of simple gates and primary inputs, represented by nodes (vertices) in the graph, and *connections*, represented by directed edges between them. V and E denote, respectively, the set of nodes and the set of edges. The *primary inputs* are nodes with no incoming edges. All the nodes with no outgoing edges are *primary outputs* (note that there may be primary output nodes with outgoing edges). The sets of primary inputs and outputs of C are, respectively, $PI(C)$ and $PO(C)$. A *complete path* is a sequence of nodes[1] connecting a primary input to a primary output. A *partial path* connects any two nodes within a complete path. A complete path is referred to simply as a *path*.

For each circuit node $v \in V$ in the graph, all the nodes connected to v by incoming edges are *fanin* nodes of v and all the nodes connected to v by outgoing edges are *fanout* nodes of v. $I(v)$ denotes the set of fanin nodes of v and $O(v)$ denotes the set of fanout nodes of v. Furthermore, each node v has an associated Boolean value that can be either 0, 1, or X (also referred to as *true*, *false*, and *unassigned*). The Boolean value of each node is represented by a Boolean variable named after the node (e.g., the Boolean value of node $v \in V$ is represented by the Boolean variable $v \in \mathcal{B}$, with $\mathcal{B} = \{0, 1, X\}$). If v is the output node of a gate, then $c(v)$ denotes the controlling logic value of the node and $nc(v)$ denotes the noncontrolling logic value of the node.

In order to model delays that digital signals suffer when traversing a combinational circuit, we associate with each edge a delay (in graph nomenclature we call it a weight). This can be either an integer or a real number, depending on the accuracy of the chosen delay model. The delay of edge $\langle v, u \rangle$, between nodes v and u, is denoted $d(v, u)$.

2.2 CNF Formulas and Boolean Satisfiability

A CNF formula φ on n Boolean variables $x_1, x_2, \ldots, x_n$, is the conjunction (AND) of m clauses $\omega_1, \omega_2, \ldots, \omega_m$, each of which is the disjunction (OR) of one or more literals, where a literal is the occurrence of a variable on its complemented or uncomplemented form. A CNF formula φ denotes a unique n-variable Boolean function $f(x_1, x_2, \ldots, x_n)$ and each of its clauses corresponds to an implicate of f.

A clause is said to be *satisfied* if at least one of its literals assumes value 1, *unsatisfied* if all of its literals assume value 0, *unit* if all but one literal assume value 0, and *unresolved* otherwise. Literals with no assigned Boolean value are said to be *free literals*. A CNF formula is said to be *satisfied* if all its clauses are satisfied, and is *unsatisfied* if at least one clause is unsatisfied.

The Boolean satisfiability problem (SAT) is concerned with finding an assignment to the arguments of $f(x_1, x_2, \ldots, x_n)$ that make the function equal to 1, or proving that the function is equal to the constant 0. The SAT problem is known to be NP-complete, therefore it is commonly accepted that any algorithmic solution for solving SAT requires worst-case exponential time in the size of each problem instance description.

3. PROBLEM FORMULATION

As first observed in Hrapčenko [1978], a combinational circuit may contain paths that are never exercised, independently of the input vector. These are

[1]Assuming that every two nodes are connected by at most one edge. Otherwise, a path is an alternating sequence of nodes and edges, starting and ending in a node.

designated by *false paths*. Clearly, false paths cannot be responsible for the delay of the circuit, since no signal transition can propagate along them. Therefore, given a combinational circuit C, the circuit delay computation (CDC) problem consists of computing the delay of the longest true (i.e., non-false) path of C. If we define a function $\Phi_C(\Delta)$ that assumes the value 1 if and only if C contains a true path with delay no smaller than Δ, and 0 otherwise, then the CDC problem can be formulated as

$$\begin{array}{c} \max\ \Delta \\ \text{s.t.}\ \ \Phi_C(\Delta) = 1. \end{array} \tag{1}$$

A true path is exercised by at least one input vector. Therefore, computing the value of $\Phi_C(\Delta)$ corresponds to determining if there is an input vector $\mathbf{x}$ that exercises at least one path of C, with delay no smaller than Δ. For a given delay Δ, two types of constraints govern the computation of $\mathbf{x}$: functional and temporal. Functional constraints describe the logical function of the circuit, and are independent of the value of Δ considered. Temporal constraints capture the temporal behavior of the circuit, by relating component delays and stable times of the nodes, and therefore are highly dependent on Δ. Assuming that these constraints can be described as clauses of two CNF formulas $\varphi_C(\mathbf{x})$ (functional constraints) and $\varphi_\Delta(\mathbf{x})$ (temporal constraints) we can compute $\mathbf{x}$ by solving the satisfiability problem for $(\varphi_\Delta \cdot \varphi_C)(\mathbf{x})$. Consequently, the following relation can be obtained.

$$\Phi_C(\Delta) = \begin{cases} 1 & \text{if} \quad (\varphi_\Delta \cdot \varphi_C)(\mathbf{x}) \text{ is satisfiable} \\ 0 & \text{if} \quad (\varphi_\Delta \cdot \varphi_C)(\mathbf{x}) \text{ is unsatisfiable.} \end{cases} \tag{2}$$

In the remainder of this section, we derive CNF formulas for $\varphi_C(\mathbf{x})$ and $\varphi_\Delta(\mathbf{x})$.

3.1 Consistency Function

The functional constraints are Boolean constraints that describe the logical behavior (function) of the circuit. The mapping of a circuit into a set of Boolean constraints is made using gate consistency functions. The *consistency function* of a circuit gate models consistent assignments on the Boolean values of its input and output nodes. The definition follows.

Definition 1 (*Gate Consistency Function*). Given a gate with n input nodes $x_1, x_2, \ldots, x_n$, one output node y, and logical function $f(x_1, x_2, \ldots, x_n)$, its *consistency function* $\xi : \mathcal{B}^{n+1} \mapsto \mathcal{B}$ is defined by

$$\xi(x_1, x_2, \ldots, x_n, y) = \overline{f(x_1, x_2, \ldots, x_n) \oplus y}. \tag{3}$$

As can be observed from the previous definition, the consistency function assumes the value 1 when the Boolean values of the input and output nodes of a gate are consistent with its logical function, and 0 otherwise. Since the consistency function is a Boolean expression it can be represented by a CNF formula. Table I shows the CNF formulas of the consistency functions for simple gates, as proposed in Larrabee [1992].

For the consistency function of a combinational circuit to be 1, the consistency function of every gate must be 1. Therefore, the consistency function of a

Table I. Consistency Functions for Simple Gates

Type	Logical Function	Consistency Function (CNF)
AND	$y = x_1 \cdot x_2 \cdot \ldots \cdot x_n$	$\prod_{i=1}^{n}(x_i + \bar{y}) \cdot \left(\sum_{i=1}^{n}\overline{x_i} + y\right)$
NAND	$y = \overline{x_1 \cdot x_2 \cdot \ldots \cdot x_n}$	$\prod_{i=1}^{n}(x_i + y) \cdot \left(\sum_{i=1}^{n}\overline{x_i} + \bar{y}\right)$
OR	$y = x_1 + x_2 + \cdots + x_n$	$\prod_{i=1}^{n}(\overline{x_i} + y) \cdot \left(\sum_{i=1}^{n}x_i + \bar{y}\right)$
NOR	$y = \overline{x_1 + x_2 + \cdots + x_n}$	$\prod_{i=1}^{n}(\overline{x_i} + \bar{y}) \cdot \left(\sum_{i=1}^{n}x_i + y\right)$
INV	$y = \bar{x}$	$(y + x) \cdot (\bar{y} + \bar{x})$
BUFFER	$y = x$	$(\bar{y} + x) \cdot (y + \bar{x})$

combinational circuit is given by the conjunction of the consistency functions of all the gates that belong to that circuit. Furthermore, if the consistency function of each gate is represented by a CNF formula, then the consistency function of the whole circuit will also be a CNF formula, because the conjunction of CNF formulas is itself a CNF formula.

It becomes clear that the CNF formula $\varphi_C(\mathbf{x})$ that represents the functional constraints of circuit C is given by the CNF representation of the consistency function of C. When solving an instance of SAT on circuit C, $\varphi_C(\mathbf{x})$ must be included in the formula to satisfy, in order to guarantee that any satisfiable solution found will have consistent Boolean values assigned to the circuit nodes.

3.2 Characteristic Function

Assuming the floating mode of operation [Chen and Du 1991], any circuit node is considered to undergo a single transition from an initial unknown value to a final known *stable value* [Chen and Du 1991]. The time instant at which this transition occurs is designated by *stable time* of the node [Chen and Du 1991]. Clearly, under the floating mode of operation the only relevant temporal property of a node is its stable time. This property can be captured as a set of Boolean constraints by using an appropriate *characteristic function* [McGeer et al. 1991], which is defined as follows.

Definition 2 (*Characteristic Function*). Given a circuit node y and an input vector $\mathbf{x}$, we define the *characteristic function* of y at time t as the Boolean function $\chi^{y,t}(\mathbf{x})$ such that $\chi^{y,t}(\mathbf{x}) = 1$ if and only if circuit node y stabilizes no earlier than t, when input vector $\mathbf{x}$ is applied to the primary inputs. Otherwise, we have $\chi^{y,t}(\mathbf{x}) = 0$.

Clearly, Definition 2 leads to the observations presented in Lemma 1.

LEMMA 1. *Given a circuit node* y, *an input vector* $\mathbf{x}$, *and two time instants* τ_1 *and* τ_2 *such that* $\tau_1 < \tau_2$, *the following conditions must hold.*

1. $[\chi^{y,\tau_2}(\mathbf{x}) = 1] \Rightarrow [\chi^{y,\tau_1}(\mathbf{x}) = 1]$
2. $[\chi^{y,\tau_1}(\mathbf{x}) = 0] \Rightarrow [\chi^{y,\tau_2}(\mathbf{x}) = 0]$.

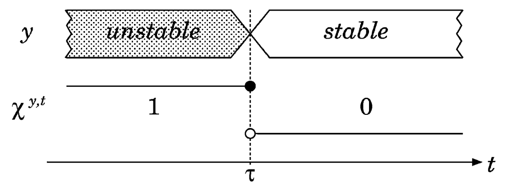

Fig. 1. Capturing the temporal behavior of a node using a characteristic function.

Proof. If a given node stabilizes no earlier than τ_2, we can also say that it stabilizes no earlier than $\tau_1 = \tau_2 - \epsilon$, $\epsilon > 0$. Therefore if $\chi^{y,\tau_2}(\mathbf{x})$ is 1, then all the characteristic functions $\chi^{y,\tau_1}(\mathbf{x})$, corresponding to times earlier than τ_2, are also 1. Similarly, if a given node stabilizes earlier than τ_1, we can also say that it stabilizes earlier than $\tau_2 = \tau_1 + \epsilon$, $\epsilon > 0$. Therefore, if $\chi^{y,\tau_1}(\mathbf{x})$ is 0, then all the characteristic functions $\chi^{y,\tau_2}(\mathbf{x})$, corresponding to times later than τ_1, are also 0. □

Consequently, it can be concluded that if node y stabilizes at time τ, then its characteristic function will be 1 for times smaller than or equal to τ and 0 for times greater than τ, as can be observed graphically in Figure 1. Thus the temporal behavior of every circuit node can now be described as a set of Boolean constraints, expressed in terms of node characteristic functions.

The purpose of CDC is to compute the true delay of a circuit. Hence, the following theorem is of vital importance for SAT-based CDC.

Theorem 1. *Given a combinational circuit C with delay* Δ, *and considering the set of primary outputs* $PO(C)$, *then for at least one input vector* $\mathbf{x}$ *the following equality must hold.*

$$\sum_{y \in PO(C)} \chi^{y,\Delta}(\mathbf{x}) = 1. \tag{4}$$

Proof. If Δ is the delay of a combinational circuit C, then there is an input vector $\mathbf{x}$ for which at least one of the primary outputs of C stabilizes no earlier than Δ. As a consequence, for at least one primary output $y \in PO(C)$, we must have $\chi^{y,\Delta}(\mathbf{x}) = 1$. Therefore, the disjunction of the characteristic functions $\chi^{y,\Delta}(\mathbf{x})$ of all the primary outputs must be 1. □

The previous theorem states that if Δ is the delay of a combinational circuit then there is an input vector $\mathbf{x}$ for which the characteristic function of at least one primary output is 1, at time Δ. This is a necessary but not sufficient condition, because any delay below the delay of the circuit will also satisfy it. The delay of the circuit is the *largest* delay that satisfies this condition.

The temporal constraints simply require that, for a given delay Δ, there must be an input vector $\mathbf{x}$ such that Condition (4) is satisfied. In order to satisfy Condition (4), we must obtain the Boolean expression of $\chi^{y,\Delta}(\mathbf{x})$, for each primary output y, implicitly or explicitly as a function of the input vector $\mathbf{x}$.

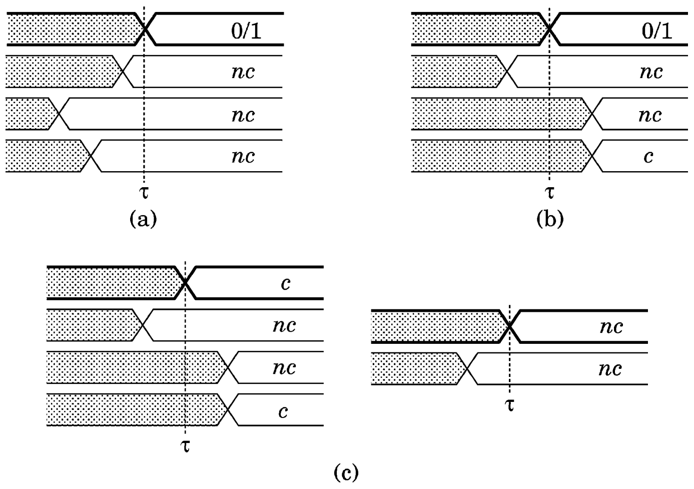

Fig. 2. A characterization of path sensitization criteria, under the floating mode of operation: (a) static; (b) viability; (c) exact.

3.3 Satisfiability Models for Path Sensitization

Next we show how to use characteristic functions to capture path sensitization conditions.

3.3.1 *Path Sensitization Conditions.* The conditions under which signals propagate from the primary inputs to the primary outputs in a combinational circuit are generally referred to as path sensitization conditions. Path sensitization conditions depend on the model of operation assumed for the circuit, in particular the different forms of stimuli on the primary inputs, and the waveform model assumed at each node in the circuit. Even though detailed and precise models can be considered, as we have mentioned before, we restrict ourselves to floating mode operation, under which all nodes are assumed to undergo a single transition, from an initial unknown value to a final known *stable* value. Most criteria defined under floating mode operation are conservative (e.g., viability [McGeer and Brayton 1991] and the exact criterion under floating mode operation [Chen and Du 1991]), thus overestimating the circuit delay in some situations. Nevertheless, as shown in McGeer and Brayton [1991], viability and exact floating mode sensitization are robust, thus providing upper bounds on the circuit delay under the bounded gate delay model (i.e., assuming that each gate delay is within some interval $[0, d_{\max}]$).

A characterization of different sensitization criteria for simple gates under floating mode of operation and assuming single path sensitization is illustrated in Figure 2. This diagram identifies logical and temporal constraints on the side inputs of each node y (thick line) in a path. τ denotes the propagation delay of a signal transition reaching y along a given path. The side input values can either be controlling (c) or noncontrolling (nc). Shaded areas indicate that a given circuit node value is unknown and may experience changes in time (unstable).

For static sensitization, the side inputs are required to assume noncontrolling values for propagation of a signal transition to occur. For viability, the side inputs are required to be either noncontrolling or to stabilize later than the node on the path. Finally, for the exact sensitization criterion under floating mode operation, it is assumed that the initial value of each primary input is unknown and changes to a known logic value at the specified arrival time. In the exact floating-mode sensitization criterion, a node y in the fanout of a node z stabilizes as a direct consequence of node z stabilizing, if z is either the earliest controlling value to stabilize or all fanin nodes assume noncontrolling values and z is the latest node to stabilize.

3.3.2 *Static Sensitization.* Under static sensitization (see Figure 2(a)), for a signal transition to reach node y all its side inputs are required to assume noncontrolling values. Therefore, and taking into account that multiple signal transitions can be propagated from the fanin nodes to y, we get the following definition of $\chi_S^{y,t}(\mathbf{x})$,[2]

$$\chi_S^{y,t}(\mathbf{x}) = \sum_{z \in I(y)} \left(\chi_S^{z,t-d(z,y)}(\mathbf{x}) \cdot \prod_{w \in I(y)-\{z\}} [w = nc(y)] \right), \tag{5}$$

which basically requires that, for a given input vector $\mathbf{x}$ at least one fanin node z of y must stabilize no earlier than $t - d(z, y)$ and such that the remaining fanin nodes w assume noncontrolling values. Clearly this condition must hold for any of the fanin nodes.

3.3.3 *Viability.* Given the interpretation of viability for simple gates in Figure 2(b) and considering the generalization for multiple paths with the same delay values, we have the following conditions for a given circuit node y to stabilize at a time no earlier than t for some input vector $\mathbf{x}$.

1. At least one fanin node z of y must stabilize at a time no earlier than $t - d(z, y)$. This condition permits the existence of multiply sensitized partial paths.
2. Furthermore, either a fanin node w of y assumes a noncontrolling value or it stabilizes at a time no earlier than $t - d(w, y)$, thus being passive regarding propagating a signal transition from z to y. Formally, we have

$$\chi_V^{y,t}(\mathbf{x}) = \sum_{z \in I(y)} \chi_V^{z,t-d(z,y)}(\mathbf{x}) \cdot \prod_{w \in I(y)} \left(\chi_V^{w,t-d(w,y)}(\mathbf{x}) + [w = nc(y)] \right), \tag{6}$$

which is basically equivalent to the simplified condition proposed in McGeer et al. [1991].

3.3.4 *Exact Floating Mode Sensitization.* In order to capture the exact path sensitization model under the floating mode of operation [Chen and Du 1991], the following observations are useful.

[2]The subscript S under χ refers to static sensitization. After the definition we discard it, for the sake of simplicity.

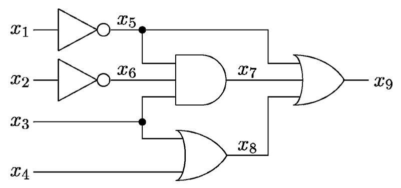

Fig. 3. A simple example circuit.

1. If the fanin node of y in any path being studied assumes a controlling value, then the exact floating mode condition is equivalent to viability.
2. Otherwise, all input nodes must be noncontrolling. In this situation, propagation from any potential fanin node z only requires that a transition reach that node (i.e., $\chi_F^{z,t-d(z,y)}(\mathbf{x}) = 1$) and that all other inputs assume noncontrolling values.

These observations lead to the following definition of $\chi_F^{y,t}(\mathbf{x})$.

$$\chi_F^{y,t}(\mathbf{x}) = \sum_{z\in I(y)} [z = c(y)] \cdot \chi_V^{y,t}(\mathbf{x}) + \prod_{z\in I(y)} [z = nc(y)] \cdot \sum_{w\in I(y)} \chi_F^{w,t-d(w,y)}(\mathbf{x}). \quad (7)$$

Observe that since a fanin node is required to satisfy $\chi_F^{z,t-d(z,y)}(\mathbf{x}) = 1$, then at least one of these nodes will guarantee $\chi_F^{y,t}(\mathbf{x}) = 1$ provided all inputs assume noncontrolling values.

3.4 An Example

Let us consider the simple example circuit shown in Figure 3. Assuming a unit-delay model (i.e., each gate has delay 1), the longest paths in the circuit are $\{x_1, x_5, x_7, x_9\}$ and $\{x_2, x_6, x_7, x_9\}$, both having delay 3. In order to decide whether 3 is the true delay of the circuit we must compute an input vector $\mathbf{x}$ that makes $(\varphi_C \cdot \varphi_{\Delta=3})(\mathbf{x})$ satisfiable, or prove that such a vector does not exist.

The functional constraints $\varphi_C(\mathbf{x})$ can be easily obtained by deriving, for every circuit gate, the corresponding consistency function, as described in Table I. In order to obtain the temporal constraints $\varphi_\Delta(\mathbf{x})$, Theorem 1 and Expression (5), (6), or (7) must be considered. From Theorem 1 we know that if 3 is the delay of the circuit then,[3]

$$\chi^{x_9,3} = 1. \quad (8)$$

Assuming the viability criterion, we can derive $\chi^{x_9,3}$ by applying Expression (6) to node x_9 as follows.

$$\chi^{x_9,3} = (\chi^{x_5,2} + \chi^{x_7,2} + \chi^{x_8,2}) \cdot (\chi^{x_5,2} + \overline{x_5}) \cdot (\chi^{x_7,2} + \overline{x_7}) \cdot (\chi^{x_8,2} + \overline{x_8}).$$

This expression can be simplified by noting that since nodes x_5 and x_8 are connected to the primary inputs by gates of delay 1, they cannot stabilize later than 1, and therefore $\chi^{x_5,2} = 0$ and $\chi^{x_8,2} = 0$. Consequently, we obtain:

$$\chi^{x_9,3} = \chi^{x_7,2} \cdot \overline{x_5} \cdot (\chi^{x_7,2} + \overline{x_7}) \cdot \overline{x_8}. \quad (9)$$

[3]In order to simplify the formulas, during this example we omit the argument ($\mathbf{x}$) of the characteristic functions.

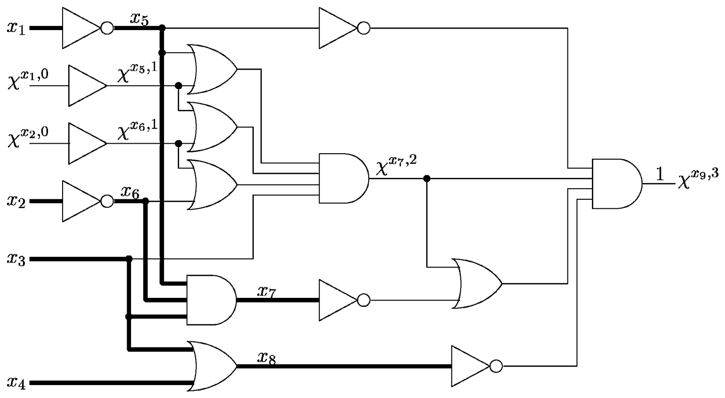

Fig. 4. Mapping of functional and temporal constraints into a circuit.

Similarly, by noting that $\chi^{x_3,1} = 0$, we can derive the characteristic function for node x_7 at time 2, as follows.

$$\chi^{x_7,2} = (\chi^{x_5,1} + \chi^{x_6,1}) \cdot (\chi^{x_5,1} + x_5) \cdot (\chi^{x_6,1} + x_6) \cdot x_3. \tag{10}$$

Since x_5 and x_6 are both outputs of an INV gate with delay 1, whose inputs are x_1 and x_2, respectively, then x_5 and x_6 will stabilize exactly 1 time unit after x_1 and x_2, therefore:

$$\chi^{x_5,1} = \chi^{x_1,0} \tag{11}$$

$$\chi^{x_6,1} = \chi^{x_2,0}. \tag{12}$$

The expressions that we have just derived implicitly represent $\chi^{x_9,3}$ as a function of the primary inputs $\mathbf{x} = (x_1, x_2, x_3, x_4)$. By performing simple substitutions, we could derive an explicit expression for $\chi^{x_9,3}$ in terms of $\mathbf{x}$. However, even for small-sized circuits it is computationally infeasible to obtain CNF representations of the characteristic functions in terms of the primary inputs. Therefore, instead of directly manipulating the characteristic functions, we map them as nodes in a combinational circuit, from which we can easily derive the corresponding CNF formula, given by the circuit consistency function. The circuit that maps $\chi^{x_9,3}$ is presented in Figure 4. The thin line shows the mapping of the characteristic functions, which originate the temporal constraints $\varphi_\Delta(\mathbf{x})$. The thick line shows the original circuit, which originates the functional constraints $\varphi_C(\mathbf{x})$. The only primary output of this circuit is $\chi^{x_9,3}$, which is also the output node of an AND gate that represents Expression (9). Similarly, the AND gate whose output is $\chi^{x_7,2}$ represents Expression (10) and the buffers whose outputs are $\chi^{x_5,1}$ and $\chi^{x_6,1}$ represent Expressions (11) and (12), respectively. By computing the CNF formula of the consistency function of this circuit, we are actually computing the CNF formula of $(\varphi_C \cdot \varphi_\Delta)(\mathbf{x})$.

After obtaining the CNF formula for $(\varphi_C \cdot \varphi_\Delta)(\mathbf{x})$ we use a chosen satisfiability algorithm to compute an input vector $\mathbf{x}$ that makes the formula satisfiable or prove that no such vector exists and that the formula is thus unsatisfiable. However, in this simple example just by inspecting the circuit in Figure 4 we can observe that

$$\begin{aligned} \chi^{x_9,3} = 1 &\Rightarrow \chi^{x_7,2} = 1 \wedge x_8 = 0 \\ \chi^{x_7,2} = 1 &\Rightarrow x_3 = 1 \\ x_8 = 0 &\Rightarrow x_3 = 0 \end{aligned}$$

and thus we conclude that there is no input vector that makes $\chi^{x_9,3} = 1$, which means that the corresponding CNF formula is unsatisfiable.

In fact, it turns out that due to the simplicity of this example it is possible, in this particular case, to obtain an explicit CNF formula for $\chi^{x_9,3}$ by simplifying the previously derived expressions, as follows.

$$\begin{aligned} \chi^{x_7,2} &= (\chi^{x_5,1} + \chi^{x_6,1}) \cdot (\chi^{x_5,1} + x_5) \cdot (\chi^{x_6,1} + x_6) \cdot x_3 \\ &= (\chi^{x_1,0} + \chi^{x_2,0}) \cdot (\chi^{x_1,0} + \overline{x_1}) \cdot (\chi^{x_2,0} + \overline{x_2}) \cdot x_3 \\ &= \overline{x_1} \cdot \overline{x_2} \cdot x_3 \end{aligned}$$

$$\begin{aligned} \chi^{x_9,3} &= \chi^{x_7,2} \cdot \overline{x_5} \cdot (\chi^{x_7,2} + \overline{x_7}) \cdot \overline{x_8} \\ &= \chi^{x_7,2} \cdot x_1 \cdot \overline{(x_3 + x_4)} \\ &= \chi^{x_7,2} \cdot x_1 \cdot \overline{x_3} \cdot \overline{x_4} \\ &= \overline{x_1} \cdot \overline{x_2} \cdot x_3 \cdot x_1 \cdot \overline{x_3} \cdot \overline{x_4}\,. \end{aligned}$$

Therefore the CNF formula for the temporal constraints is given by

$$\varphi_\Delta(\mathbf{x}) = \chi^{x_9,3} = \overline{x_1} \cdot \overline{x_2} \cdot x_3 \cdot x_1 \cdot \overline{x_3} \cdot \overline{x_4}\,.$$

Once more, it becomes clear that $\varphi_\Delta(\mathbf{x})$ is unsatisfiable, because both x_1 and x_3 appear in their complemented and uncomplemented form. Since $\varphi_\Delta(\mathbf{x})$ is unsatisfiable, so is $(\varphi_C \cdot \varphi_\Delta)(\mathbf{x})$. We stress, however, that only for very small circuits is it possible to explicitly obtain the desired CNF formula which makes such a procedure inapplicable in general.

4. THE CIRCUIT DELAY COMPUTATION ALGORITHM

Given Expressions (1) and (2) the CDC problem can be formulated as

$$\begin{aligned} &\max\ \Delta \\ &\text{s.t. } (\varphi_C \cdot \varphi_\Delta)(\mathbf{x}) \text{ is satisfiable.} \end{aligned} \tag{13}$$

Simply speaking, the CDC problem consists of computing the largest delay Δ that makes $(\varphi_C \cdot \varphi_\Delta)(\mathbf{x})$ satisfiable. In this section we detail the proposed algorithm for solving this problem.

4.1 Basic Algorithm

An intuitive way of solving problem (13) is to initialize the target delay Δ with its largest value and iteratively decrease it until we find a Δ that makes $(\varphi_C \cdot \varphi_\Delta)(\mathbf{x})$ satisfiable. Clearly, an upper bound for the true delay of the circuit is the delay of its longest (false or nonfalse) path, which corresponds to the longest topological path delay (*LTP*). Therefore, by initializing $\Delta = LTP$ and then iteratively decreasing Δ until $(\varphi_C \cdot \varphi_\Delta)(\mathbf{x})$ becomes satisfiable, we will obtain the true delay of the circuit.

The proposed algorithm for solving the CDC problem is implemented by function **CDC**(), listed in Figure 5. The only input argument of the algorithm is

```
float **CDC**($C$)
{
  $\Delta = \mathbf{LTP}(C)$;
  $\varphi_C = \mathbf{FunctionalCNF}(C)$;
  $\varphi_\Delta = \mathbf{TemporalCNF}(C, \Delta)$;
  while ( $\mathbf{SAT}(\varphi_C \cdot \varphi_\Delta) ==$ UNSATISFIABLE ) {
    $\Delta = \mathbf{IterateDelay}(C, \Delta)$;
    $\varphi_\Delta = \mathbf{TemporalCNF}(C, \Delta)$;
  }
  return $\Delta$;
}
```

Fig. 5. The CDC algorithm.

a combinational circuit C and the single result is its true delay, that we assume to be a real number.

We start by initializing the target delay Δ with the longest topological path delay of C, computed by **LTP**(). In **LTP**(), we perform a forward traversal of the circuit graph, and for each node we compute the delay of the longest path that ends at that node. The longest topological path delay is the delay of the longest path that ends at a primary output. Afterwards, the mapping of circuit C into a CNF formula φ_C is performed by function **FunctionalCNF**(), as described in Section 3.1. The CNF formula φ_Δ of the temporal constraints is generated by function **TemporalCNF**() for each target delay Δ. Inside **TemporalCNF**() we perform a backward traversal of the circuit graph, starting in the primary outputs, and for each node we compute all the necessary characteristic functions. These functions are then mapped into a combinational circuit and the corresponding CNF representation is obtained by extracting its consistency function. Having computed a CNF formula for $\varphi_C \cdot \varphi_\Delta$ we then call a chosen satisfiability algorithm, implemented by function **SAT**() to verify whether $\varphi_C \cdot \varphi_\Delta$ is satisfiable. If it is satisfiable then the value of Δ is returned, which corresponds to the true delay of the circuit. Otherwise, we enter a loop in which new values for the target delay Δ are chosen by function **IterateDelay**(), and for each of which the corresponding temporal constraints φ_Δ are generated by function **TemporalCNF**(). If for a given target delay Δ the CNF formula $\varphi_C \cdot \varphi_\Delta$ is satisfiable, the loop terminates and the value of Δ, which is the true delay of the circuit, is returned.

4.2 Boolean Satisfiability Algorithm

Over the years a large number of approaches have been proposed for solving SAT. It is widely accepted that the two most competitive classes of SAT algorithms are local search and backtrack search. Local search algorithms are not complete, in the sense that they cannot prove unsatisfiability. Most often, this drawback prevents their utilization in a significant number of EDA applications. Consequently, backtrack search algorithms emerge as the most promising solution for solving SAT in EDA, and thus are used to solve the CDC problem.

The overall organization of a generic backtrack search SAT algorithm is implemented in function **SAT**() as shown as Figure 6. The only input argument of the algorithm is a CNF formula φ and the result is either SATISFIABLE or UNSATISFIABLE, according to the satisfiability of the formula. This SAT algorithm captures the organization of several of the most competitive

```
int SAT($\varphi$)
{
  $\lambda = 0$;
  Preprocess($\varphi$);
  while ( Decide($\varphi, \lambda$) == DECISION ) {
    if ( Deduce($\varphi, \lambda$) == CONFLICT ) {
      $\beta$ = Diagnose($\varphi, \lambda$);
      if ( $\beta == 0$ )
        return UNSATISFIABLE;
      else {
        Backtrack($\varphi, \lambda, \beta$);
        $\lambda = \beta$;
      }
    }
    $\lambda = \lambda + 1$;
  }
  return SATISFIABLE;
}
```

Fig. 6. A generic SAT algorithm.

algorithms [Bayardo, Jr. and Schrag 1997; Marques-Silva and Sakallah 1996; Zhang 1997].

The algorithm conducts a search through the space of the possible assignments to the problem instance variables. At each stage of the search, an assignment is selected with the **Decide**() function. A decision level λ is also associated with each selection of an assignment. Implied necessary assignments are identified with the **Deduce**() function. Whenever a clause becomes unsatisfied, the **Deduce**() function returns a conflict indication which is then analyzed using the **Diagnose**() function. The diagnosis of a given conflict returns a backtracking decision level β, which corresponds to the decision level to which the search process can provably backtrack. The **Backtrack**() function clears variable assignments from the current decision level λ up to a decision level β. The search terminates in one of two situations: either **Diagnose**() detects a conflict at the decision level 0, meaning that the formula is unsatisfiable, or **Decide**() detects that all the clauses are satisfied, meaning that the formula is satisfiable.

Currently, and for solving large, structured, and hard instances of SAT, all of the most efficient SAT algorithms implement a number of the following key properties:

1. The analysis of conflicts can be used for implementing nonchronological backtracking search strategies. Hence, assignment selections that are deemed irrelevant can be skipped during the search [Bayardo, Jr. and Schrag 1997; Marques-Silva and Sakallah 1996; Zhang 1997].
2. The analysis of conflicts can also be used for identifying and recording new clauses that denote implicates of the Boolean function associated with the CNF formula. Clause recording plays a key role in recent SAT algorithms, despite in most cases large recorded clauses being eventually deleted [Bayardo, Jr. and Schrag 1997; Marques-Silva and Sakallah 1996; Zhang 1997].
3. Other techniques have been developed. Relevance-based learning [Bayardo, Jr. and Schrag 1997] extends the lifespan of large recorded clauses that will

eventually be deleted. Conflict-induced necessary assignments [Marques-Silva and Sakallah 1996] denote assignments of variables that are necessary for preventing a given conflict from occurring again during the search.

Before running the SAT algorithm, different forms of preprocessing can be applied [Li and Anbulagan 1997; Marques-Silva and Sakallah 1996]. This in general is denoted by a **Preprocess**() function that is executed before invoking the search process.

4.3 Delay Stepping Strategies

A key procedure in the circuit delay computation algorithm is the stepping of target path delays [Guerra e Silva et al. 1998b], implemented in function **IterateDelay**(). In general delay stepping plays a crucial role in the overall efficiency of the algorithm, since it determines the number of iterations to be executed.

The simplest delay stepping strategy is to change the specified delay by the least delay unit at each iteration of the algorithm. Consequently, Δ is continuously decreased by 1 delay fraction (that corresponds to the smallest delay variation possible, given a predefined precision). Although the computation of Δ is immediate, this kind of strategy can result in an enormous number of iterations, especially for circuits with a large number of false paths, where the critical delay is much smaller than the topological delay, or whenever precise delay models are assumed.

Moreover, it is clear that not all path delays are possible on each primary output. Hence, instead of decreasing Δ in a fractional step basis we can analyze the circuit topology and get the next topological delay present at the specified primary output. This is called the *next delay* stepping strategy, because the delay step is determined by the next topological delay present at a given primary output. For circuits with few path delays, this strategy can yield a reduction in the number of iterations, with a small increase in the time necessary to compute Δ. However, for circuits with a large number of paths, where there exists an almost continuous delay distribution, this strategy rapidly becomes as inefficient as the previous one, thus taking significantly more time to compute Δ.

To overcome the limitations of the two previous strategies, we can use *dynamic stepping*. In this strategy the delay step is dynamically adjusted, according to certain criteria, but it is independent of the circuit topology (no next delay computation is performed). The basic principle behind this strategy is to choose appropriate values for the steps in order to enclose the true delay of the circuit in a series of intervals of decreasing size. We start by generating a SAT instance of the CDC problem for $\Delta = LTP$. If this instance is unsatisfiable we iteratively decrease Δ by a delay step s_0 until we reach a satisfiable instance. At this point we know that the true delay of the circuit lies within the interval $[\Delta, \Delta + s_0[$, of size s_0. We can then choose a smaller step s_1 ($s_1 < s_0$), and starting at $\Delta + s_0 - s_1$ continue to iteratively decrease Δ until we reach a satisfiable instance. Again, when that happens we know that the true delay of the circuit lies within the interval $[\Delta, \Delta + s_1[$. By repeatedly using smaller delay steps s_i, we can obtain an estimate of Δ with an arbitrary precision. It should be noted that to implement dynamic stepping in the CDC algorithm shown in Figure 5 the function **IterateDelay**() must receive as an argument the result of the **SAT**() function. Furthermore, the function **IterateDelay**() will replace the function **SAT**() in the control of the loop.

The choice of the correct values for the delay steps is a trade-off between faster progression, which is achieved using large steps, and smaller delay intervals, which can be obtained using small steps. In addition it should be noted that for delay values much smaller than the true delay of the circuit there are many paths to be considered, and therefore the corresponding SAT instances can become very large and potentially hard to solve. This makes the use of large delay steps (or some strategies such as binary search) potentially dangerous, because one could easily end up in a path delay zone far below the true delay of the circuit, which could have disastrous results in terms of CPU time.

A particularly interesting implementation of dynamic stepping can be obtained by matching the computation of a specific decimal place of the true delay with the delay step used. Let us suppose that we want to compute the digit k_n $(0 \leq k_n \leq 9)$ of the nth decimal place $(n \geq 1)$ of the true delay of a circuit, given an estimate Δ_{n-1} with $(n-1)$ decimal places. The value of k_n can be obtained by bounding the previous estimate Δ_{n-1} to an interval $[\Delta_{n-1} + 10^{-n} \times k_n, \Delta_{n-1} + 10^{-n} \times (k_n +, 1)[$. This can be easily accomplished by iteratively testing all the delays starting at $\Delta_{n-1} + 10^{-(n-1)} - 10^{-n}$ and ending at $\Delta_{n-1} + 10^{-n}$, with delay step $s = 10^{-n}$. Clearly this procedure can be used to compute estimates of the true delay with an arbitrary number of decimal places. The computation of the value of each decimal place (starting from the first) takes at most nine iterations. To compute the integer part of the true delay we start by testing for $\Delta = LTP$ and if the generated SAT instance is unsatisfiable we round Δ to the nearest integer smaller than *LTP*. If the instance generated for this delay is still unsatisfiable we iteratively decrease Δ using delay steps of 1 until we find a satisfiability instance. When that happens we have computed the integer value of the true delay of the circuit and we can start using the previously described procedure to identify the following decimal places.

Let us consider the example, shown in Figure 7, of a circuit with *LTP* of 20.0 and true delay of 17.3. Using fractional stepping we spend 28 iterations to find the true delay, which corresponds to the number of all possible delays with precision of 1 decimal place that exist between 20.0 and 17.3 (including both). The next delay stepping strategy yields a slight reduction in the number of iterations when compared to fractional stepping because not every delay is present at each primary output, thus we only spend 22 iterations. Using dynamic stepping we start with a delay step of 1 to compute the integer value of the true delay. When that value is computed (17) we reduce the delay step to 0.1 and we decreasingly test all the delays starting in 17.9 (since 18.0 has already been tested) until 17.3, which corresponds to an estimate of the true delay of the circuit with 1 decimal digit. This estimate is obtained in only 11 iterations.

From the previous example we can clearly conclude that when using a fractional delay strategy the number of iterations increases geometrically with the

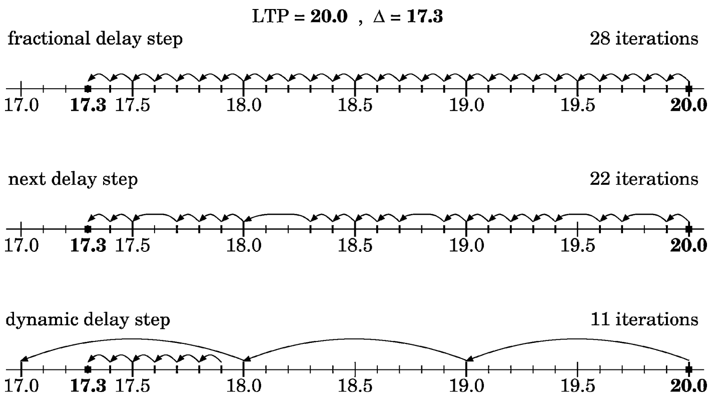

Fig. 7. Illustration of delay stepping strategies.

number of decimal places. With the next delay strategy, the rate of growth is almost the same. However, using this particular implementation of dynamic stepping, the number of iterations necessary to compute each decimal place (starting at the first) is at most 9.

5. REALISTIC DELAY MODELING

Existing research work on CDC has focused mainly on path sensitization models and algorithms, and on gate and interconnect delay models. Nevertheless, work in these two main areas has evolved separately, and so most path sensitization models and algorithms assume very rudimentary gate and interconnect delay models. Even though such models provide an attractive framework for comparing the correctness and performance of the various approaches to path sensitization, due to their poor accuracy they cannot be used in practical circuit design. For robust and accurate CDC we need in general to consider more realistic delay models which should take into account at least these constraints:

—different delay values for different types of gates;
—different delay values for different gate inputs;
—variation of delay with the number of fanouts/fanins (accounting for input and output capacitances); and
—interconnect delay estimation (for circuits for which layout information is available).

Gate delays and their variation with the number of fanouts/fanins can be easily modeled using information available from the IC library databook. Interconnect delay, however, is hard to estimate for the benchmark circuits available, which are only described at the gate level. To obtain this information the benchmark circuits were mapped using the standard-cell library ECPD07 (ES2/Atmel),[4] and the parasitic capacitances of the interconnect were extracted.

[4]Mapping to this library requires that each gate with more than four inputs has to be expanded into a sequence of gates each with no more than four inputs.

For each gate input, the propagation delay t_p is given by

$$t_p = t_{p_i} + dt_p \cdot C_l, \tag{14}$$

where t_{p_i} is the intrinsic propagation delay, dt_p is the differential (load-dependent) propagation delay, and C_l is the load capacitance at the gate output. Furthermore, this load capacitance is given by

$$C_l = C_i + \sum_j C_{g_j}, \tag{15}$$

where C_i is the lumped interconnect capacitance and $\sum_j C_{g_j}$ is the sum of the input capacitances of all the fanouts. The interconnect resistance for this technology is very small, resulting in a negligible interconnect delay that has been discarded. However, the interconnect capacitance is significant and was used to more realistically model the load-dependent propagation delay of each gate. Even though both rise and fall delays are available for each gate input, that is not supported by the problem formulation proposed in Section 3 and therefore the most conservative values are considered. We further note that all delays were computed with two-digit precision.

Clearly this gate delay model will lead to a significantly larger number of path delays, which will increase the number of iterations of the circuit delay computation algorithm [Guerra e Silva et al. 1998a].

6. EXPERIMENTAL RESULTS

The CDC algorithm consists of iteratively generating and solving instances of SAT for several circuit delays, starting at the largest topological path delay. All the models and algorithms described in the previous sections have been implemented and used for generating many instances of SAT, each of which denotes the sensitization conditions for a certain specified delay, in a given circuit. These instances have been solved using some of the most competitive SAT algorithms currently available. The results obtained are presented and discussed throughout this section. All the experiments were conducted on a Sun Ultra 1 machine, with one 167 MHz UltraSPARC processor and 384 Mb of physical memory. The CPU times are presented in seconds.

6.1 Sensitization Criteria and SAT Algorithms

The results of solving CDC, assuming a unit delay model, for static sensitization, viability, and the exact sensitization criterion under floating mode of operation, are shown in Tables II, III, and IV, respectively. For each table, the first column presents the name of the benchmark circuit and the second column exhibits the longest topological delay of each circuit and its true delay, as computed by the CDC algorithm. The third column presents the number of CDC iterations (i.e., the number of SAT instances generated and solved) and the fourth column exhibits the CPU time spent generating all the SAT instances. The remaining columns exhibit the CPU time used by the various SAT algorithms for solving the instances derived from the CDC problem. The algorithms used are: GRASP [Marques-Silva and Sakallah 1996], RELSAT

Table II. Run Times for CDC Using Static Sensitization

Circuit	LTP/Δ	It	CNF	Grasp	Relsat	Posit	Sato	Satz	Dp	Hhugo	Tegus
c432	17/17	1	0.01	0.02	0.15	0.00	0.03	0.06	0.15	0.00	0.04
c499	11/11	1	0.01	0.03	0.27	0.01	0.09	0.08	0.61	1.00	0.04
c880	24/24	1	0.01	0.05	0.17	0.01	0.04	0.09	0.59	1.00	0.05
c1355	24/24	1	0.02	0.15	0.44	0.03	0.07	0.35	>1000	1.00	0.12
c1908	40/37	11	0.20	0.27	3.61	0.00	0.57	1.57	29.61	13.00	19.07
c2670	32/30	6	0.16	5.74	30.51	711.99	0.86	>1000	>1000	21.00	>1000
c3540	47/46	4	0.06	0.65	4.57	0.07	0.46	1.03	712.52	8.00	263.54
c5315	49/47	7	0.12	2.41	6.04	0.28	1.01	4.07	>1000	27.00	>1000
c6288	124/123	3	0.14	71.00	21.72	0.51	152.77	4.27	>1000	431.00	>1000
c7552	43/42	6	0.05	2.63	8.27	0.01	1.33	4.92	>1000	21.00	6.13
cla.16	34/33	2	0.01	0.04	0.08	0.00	0.03	0.01	0.77	2.00	0.44
cbp.12.2	40/23	73	1.07	0.97	5.29	0.11	1.99	2.48	143.27	50.00	10.31
cbp.16.4	44/27	83	0.90	1.12	5.67	0.09	2.19	2.66	119.60	43.00	36.92
csa.16.4	41/22	91	0.49	0.31	2.72	0.01	1.42	2.03	12.38	38.00	189.47
csa.32.4	81/30	571	9.08	12.30	50.15	0.73	14.62	30.24	>1000	393.00	>1000

Table III. Run Times for CDC Using Viability

Circuit	LTP/Δ	It	CNF	Grasp	Relsat	Posit	Sato	Satz	Dp	Hhugo	Tegus
c432	17/17	1	0.01	0.02	0.10	0.00	0.03	0.07	0.14	0.00	0.03
c499	11/11	1	0.01	0.03	0.08	0.01	0.09	0.06	0.62	1.00	0.04
c880	24/24	1	0.01	0.05	0.16	0.01	0.03	0.11	0.59	0.00	0.05
c1355	24/24	1	0.02	0.26	0.41	0.07	0.08	0.23	>1000	1.00	0.13
c1908	40/37	11	0.19	0.31	3.44	0.00	0.62	1.40	54.92	12.00	15.38
c2670	32/30	6	0.18	5.07	28.67	623.99	2.08	>1000	>1000	258.00	>1000
c3540	47/46	4	0.06	0.67	4.03	0.06	0.37	1.10	908.99	9.00	266.40
c5315	49/47	7	0.13	3.25	5.78	0.26	1.10	4.11	>1000	538.00	>1000
c6288	124/123	3	0.16	47.33	28.57	>1000	9.27	1.71	>1000	>1000	148.69
c7552	43/42	6	0.05	2.66	9.77	0.01	1.33	4.87	>1000	21.00	3.77
cla.16	34/34	1	0.00	0.02	0.04	0.00	0.02	0.01	0.33	1.00	0.17
cbp.12.2	40/23	73	1.20	1.23	4.81	0.31	2.08	2.82	>1000	45.00	11.08
cbp.16.4	44/27	83	1.01	1.34	5.64	0.17	2.37	3.06	357.81	45.00	35.57
csa.16.4	41/22	91	0.54	0.33	2.73	0.02	1.46	2.22	15.48	56.00	378.87
csa.32.4	81/30	571	9.92	14.75	48.28	1.80	15.12	32.54	>1000	427.00	>1000

Table IV. Run Times for CDC Using the Exact Criterion

Circuit	LTP/Δ	It	CNF	Grasp	Relsat	Posit	Sato	Satz	Dp	Hhugo	Tegus
c432	17/17	1	0.01	0.05	0.18	0.01	0.03	0.10	0.35	1.00	0.05
c499	11/11	1	0.01	0.03	0.09	0.01	0.10	0.15	0.68	1.00	0.04
c880	24/24	1	0.01	0.06	0.23	0.01	0.03	0.15	1.10	1.00	0.07
c1355	24/24	1	0.04	0.36	0.76	0.00	0.13	0.49	>1000	9.00	0.47
c1908	40/37	11	0.33	1.00	8.86	0.14	0.96	2.27	111.07	15.00	23.91
c2670	32/30	6	0.32	12.20	38.05	>1000	3.76	>1000	>1000	65.00	>1000
c3540	47/46	4	0.13	1.05	9.32	0.13	0.47	1.57	>1000	13.00	557.68
c5315	49/47	7	0.27	7.78	13.70	3.99	1.30	8.68	>1000	173.00	>1000
c6288	124/123	3	0.35	42.61	47.92	>1000	108.67	21.40	>1000	27.00	>1000
c7552	43/42	6	0.11	5.22	17.30	0.04	1.52	5.46	>1000	25.00	33.55
cla.16	34/34	1	0.01	0.07	0.08	0.00	0.04	0.03	4.46	1.00	0.11
cbp.12.2	40/23	73	2.39	10.97	23.55	1.97	7.72	6.99	>1000	83.00	252.40
cbp.16.4	44/27	83	2.02	10.69	23.77	1.96	8.90	6.17	>1000	80.00	>1000
csa.16.4	41/22	91	1.07	3.19	10.43	0.56	2.37	11.78	>1000	61.00	>1000
csa.32.4	81/30	571	21.18	179.43	314.61	34.80	51.99	>1000	>1000	938.00	>1000

[Bayardo, Jr. and Schrag 1997], Posit [Freeman 1995], Sato [Zhang 1997], Satz [Li and Anbulagan 1997], Dp [Barth 1995], Heerhugo [Groote and Warners 1999], and Tegus [Stephan et al. 1992].[5] It should be noted that instead of generating an instance of SAT for each target delay (as is described in the proposed algorithm), we have decided to generate a SAT instance for each output/delay pair. Using this approach we are able to split a hard instance in many easier instances, which has proven to be a much more efficient technique in terms of CPU time, even though more iterations are usually required.

As can be seen, the SAT instances are very easy to generate, even for large circuits. It is interesting to observe that the vast majority of the instances of SAT are extremely easy to solve with most SAT algorithms. The exception to this rule is Dp, the basic Davis–Putnam procedure [Davis and Putnam 1960], which is unable to solve a significantly large number of benchmarks. On the other hand, only a few SAT algorithms are able to solve all the instances in a reasonable amount of time. In general, Grasp, RelSat, and Sato seem to be the most efficient and robust algorithms for solving this class of instances of SAT.

Comparing the sensitization criteria we can observe that static sensitization spends the smallest CPU time, followed by viability and the exact criterion. However, static sensitization is known to underestimate the circuit delay, as can be observed for *cla.16*, which makes it unusable. Clearly, viability seems to exhibit the best trade-off between correctness and performance.

6.2 Realistic Delays

In Table V we present the CPU times for checking satisfiability using Grasp and Relsat[6] obtained assuming both a unit delay model and a realistic delay model on the technology mapped circuits. For this experiment the path sensitization criterion used was viability. As can be observed, the CPU times increase significantly as more realistic delays are considered, not only because the number of iterations of the algorithm also increases, but also because the SAT instances are harder to solve due to a larger number of characteristic functions generated by an almost continuous delay distribution. However, this added complexity signifies that we are now able to obtain much more accurate estimates for the circuit delay.

[5]Grasp refers to the February 2000 version, run by command "`sat-grasp +V0 +rt8 +g20 <file>`". Relsat refers to version 1.1.2 modified to print the elapsed CPU time instead of the elapsed real time, run by command "`relsat 3 <file>`". Posit refers to version 1.0, run by command "`posit -f <file>`". Sato refers to version 3.0, run by command "`sato -f <file>`". Satz refers to the September 1996 version, run by command "`satz <file>`". Dp refers to version 0.2, run by command "`dp -h2 -t2 -0 -f<file>`". Hhugo (Heerhugo) refers to version 0.3, run by command "`heerhugo`" with input file `Ain`, in propositional format. Tegus refers to the satisfiability checker incorporated into version 1.4 of Sis, to which an interface able to read DIMACS files was added. Furthermore, a number of search strategies were added with an increasing number of backtrack limits in order to prevent Tegus from aborting instances before exceeding the CPU time limit.

[6]Even though Sato exhibits the smallest of the CPU times, it is known to compute the wrong answer for some instances, and therefore we chose not to include it in the following tests.

Table V. Run Times for Unit Versus Realistic Delay Models, Using Viability

Circuit	Unit Delay					Realistic Delay				
	LTP/Δ	It	CNF	GRASP	RELSAT	LTP/Δ	It	CNF	GRASP	RELSAT
c432	17/17	1	0.01	0.02	0.10	21.81/21.46	36	0.32	0.31	2.12
c499	11/11	1	0.01	0.03	0.08	16.96/16.96	1	0.00	0.02	0.11
c880	24/24	1	0.01	0.05	0.16	18.06/18.06	1	0.00	0.03	0.15
c1355	24/24	1	0.02	0.26	0.41	23.19/22.76	85	2.51	3.35	10.59
c1908	40/37	11	0.19	0.31	3.44	26.00/23.36	881	1174.89	44.74	75484.41
c2670	32/30	6	0.18	5.07	28.67	35.16/33.33	336	34.49	146.75	1873.77
c3540	47/46	4	0.06	0.67	4.03	35.33/33.63	171	304.41	260.38	15395.12
c5315	49/47	7	0.13	3.25	5.78	62.51/61.25	174	5.96	37.59	94.98
c6288	124/123	3	0.16	47.33	28.57	92.62/91.69	134	173.94	989.61	14018.60
c7552	43/42	6	0.05	2.66	9.77	37.60/35.25	435	48.26	120.63	891.55
cla.16	34/34	1	0.00	0.02	0.04	28.05/28.03	3	0.02	0.07	0.08
cbp.12.2	40/23	73	1.20	1.23	4.81	27.44/17.41	3856	142.62	209.58	539.44
cbp.16.4	44/27	83	1.01	1.34	5.64	31.72/20.64	5299	62.33	93.27	326.61
csa.16.4	41/22	91	0.54	0.33	2.73	35.51/19.90	7348	48.88	30.36	220.53
csa.32.4	81/30	571	9.92	14.75	48.28	72.00/28.42	45695	28.42	5158.47	13245.08

Table VI. Results for Various Delay Stepping Stategies Using Viability and Realistic Delays

Circuit	LTP/Δ	Fractional			Next			Dynamic		
		It	CNF	GRASP	It	CNF	GRASP	It	CNF	GRASP
c432	21.81/21.46	36	0.32	0.31	23	0.30	0.23	11	0.37	0.26
c499	16.96/16.96	1	0.00	0.02	1	0.00	0.02	1	0.00	0.02
c880	18.06/18.06	1	0.00	0.03	1	0.00	0.03	1	0.00	0.03
c1355	23.19/22.76	85	2.51	3.35	77	2.56	3.33	16	3.00	4.20
c1908	26.00/23.36	881	1174.89	44.74	631	1164.66	42.87	30	36.87	1.43
c2670	35.16/33.33	336	34.49	146.75	210	31.25	124.38	28	9.22	28.94
c3540	35.33/33.63	171	304.41	260.38	158	299.29	262.01	15	94.38	178.14
c5315	62.51/61.25	174	5.96	37.59	59	3.64	22.40	20	1.55	10.38
c6288	92.62/91.69	134	173.94	989.61	134	172.33	1019.02	15	51.77	553.93
c7552	37.60/35.25	435	48.26	120.63	178	43.24	88.97	37	11.49	32.49
cla.16	28.05/28.03	3	0.02	0.07	3	0.02	0.07	5	0.04	0.13
cbp.12.2	27.44/17.41	3856	142.62	209.58	1088	75.96	104.90	65	3.23	7.98
cbp.16.4	31.72/20.64	5299	62.33	93.27	554	16.02	17.47	80	1.05	1.86
csa.16.4	35.51/19.90	7348	48.88	30.36	402	4.87	3.24	105	0.76	0.59
csa.32.4	72.00/28.42	45695	28.42	5158.47	12690	2044.94	3046.32	511	32.70	58.84

6.3 Delay Stepping Strategies

The delay computation results for the three delay stepping strategies analyzed, using realistic delays and viability criteria, are shown in Table VI. Observing the table one can easily conclude that the selection of the delay stepping strategy significantly affects the overall efficiency of the algorithm. For fractional delay stepping, a huge number of iterations is required to solve certain instances, with the corresponding large CPU times. Using next delay stepping, a reduction in the number of iterations is achieved. Finally, for dynamic stepping a dramatic reduction in the number of iterations is observed, with the corresponding small CPU times.

7. CONCLUSIONS

In this article we propose a unified propositional satisfiability modeling and algorithmic framework for studying circuit delay computation. Different path sensitization criteria were considered and reasonably efficient results were obtained. Regarding the SAT algorithms used, one class provides by far the most efficient and robust results. The algorithms in this class (GRASP, RELSAT, and SATO) use a number of search pruning techniques, which are shown to be particularly effective for solving circuit delay computation problems. Moreover, more realistic delay models, which take into account extracted interconnect delays and fanout data, were incorporated into the proposed modeling and algorithmic framework. Preliminary results suggest that the approach is still feasible, although necessarily more inefficient. Three delay stepping strategies were considered: fractional delay stepping, next delay stepping and dynamic stepping. The dynamic stepping provides by far the most efficient and robust results.

REFERENCES

ASHAR, P., MALIK, S., AND ROTHWEILER, S. 1993. Functional timing analysis using ATPG. In *Proceedings of the European Design Automation Conference*.

BARTH, P. 1995. A Davis–Putnam based enumeration algorithm for linear pseudo-Boolean optimizations. Tech. Rep. MPI-I-95-2-003 (Jan.), Max-Planck-Institut für Informatik.

BAYARDO, JR., R. AND SCHRAG, R. 1997. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the National Conference on Artificial Intelligence* (*AAAI'97*).

BENKOSKI, J., MEERSCH, E. V., CLAESEN, L., AND DE MAN, H. 1987. Efficient algorithms for solving the false path problem in timing verification. In *Proceedings of the International Conference on Computer-Aided Design* (Nov.), 44–47.

BERGAMASCHI, R. 1991. The effects of false paths in high-level synthesis. In *Proceedings of the International Conference on Computer Aided-Design* (Nov.).

CHEN, H.-C. AND DU, D. H. C. 1991. Path sensitization in critical path problem. In *Proceedings of the International Conference on Computer Aided-Design* (Nov.), 208–211.

DAVIS, M. AND PUTNAM, H. 1960. A computing procedure for quantification theory. *J. ACM 7*, 201–215.

DEVADAS, S., KEUTZER, K., AND MALIK, S. 1993. Computation of floating-mode delay in combinational circuits: Practice and implementation. *IEEE Trans. CAD 12*, 12 (Dec.), 1924–1936.

FREEMAN, J. W. 1995. Improvements to propositional satisfiability search algorithms. PhD Thesis, Department of Computer and Information Science, University of Pennsylvania.

GROOTE, J. AND WARNERS, J. 1999. The propositional formula checker HeerHugo. Tech. Rep. SEN-R9905, Centre for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands.

GUERRA E SILVA, L., MARQUES-SILVA, J., SILVEIRA, L. M., AND SAKALLAH, K. A. 1998a. Realistic delay modeling in satisfiability-based timing analysis. In *Proceedings of the IEEE International Symposium on Circuits and Systems* (Monterey, CA, May–June).

GUERRA E SILVA, L., MARQUES-SILVA, J., SILVEIRA, L. M., AND SAKALLAH, K. A. 1998b. Timing analysis using propositional satisfiability. In *Proceedings of the IEEE International Conference on Electronics, Circuits and Systems* (Lisboa, Portugal, Sept.).

HRAPČENKO, V. 1978. Depth and delay in a network. *Soviet Math. Dokl. 19*, 4.

LARRABEE, T. 1992. Test pattern generation using Boolean satisfiability. In *IEEE Trans. Comput. Aided Des. 11* (Jan.), 4–15.

LI, C. M. AND ANBULAGAN. 1997. Look-ahead versus look-back for satisfiability problems. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*.

MARQUES-SILVA, J. AND SAKALLAH, K. A. 1994. Efficient and robust test-generation based timing analysis. In *Proceedings of the International Symposium on Circuits and Systems*, 303–306.

MARQUES-SILVA, J. AND SAKALLAH, K. A. 1996. GRASP: A new search algorithm for satisfiability. In *Proceedings of the International Conference on Computer-Aided Design* (Nov.), 220–227.

MCGEER, P. C. 1989. On the interaction of functional and timing behavior of combinational logic circuits. PhD Thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA.

McGeer, P. C. and Brayton, R. K. 1991. *Integrating Functional and Temporal Domains in Logic Design*. Kluwer Academic, Norwell, Mass.

McGeer, P. C., Saldanha, A., Stephan, P. R., Brayton, R. K., and Sangiovanni-Vincentelli, A. L. 1991. Timing analysis and delay-fault test generation using path-recursive functions. In *Proceedings of the International Conference on Computer Aided-Design*, 180–183.

Stephan, P. R., Brayton, R. K., and Sangiovanni-Vincentelli, A. L. 1992. Combinational test generation using satisfiability. Tech. Rep. UCB/ERL M92/112 (Oct.), Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA.

Yalcin, H. and Hayes, J. P. 1995. Hierarchical timing analysis using conditional delays. In *Proceedings of the International Conference on Computer Aided-Design* (Nov.).

Zhang, H. 1997. SATO: An efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction* (July), 272–275.

Received March 2000; accepted July 2001