An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization

Sachin S. Sapatnekar, Vasant B. Rao, Pravin M. Vaidya and Sung-Mo Kang

I. INTRODUCTION

Circuit delays in MOS integrated circuits often need to be reduced to obtain faster response times, with a minimal area penalty. A typical MOS digital integrated circuit consists of multiple stages of combinational logic blocks that lie between latches, clocked by system clock signals. Delay reduction must ensure that the worst-case delay of the combinational blocks is such that valid signals reach a latch before any transition in the signal clocking the latch, with allowances for set-up time requirements. In other words, the worst-case delay of each combinational stage must be restricted to be below a certain specification. The requirements for hold times are different in nature, and are not addressed in this paper.

Given the MOS circuit topology, the delay can be controlled by varying the sizes of transistors in the circuit. Here, the size of a transistor is measured in terms of its channel width, since the channel lengths in a digital circuit are generally uniform. Roughly speaking, the sizes of certain transistors can be increased to reduce the circuit delay at the expense of additional chip area.

For a combinational circuit, the transistor sizing problem is formulated as

$$\begin{array}{rc} minimize & Area \\ subject\ \ to & Delay \leq T_{spec} \\ and & \text{Each transistor size} \geq \text{Minsize} \end{array} \qquad (1)$$

Several other formulations have also been suggested, such as minimizing the area-delay product, and minimizing the delay subject to a constraint on the maximum permissible circuit area.

It has widely been recognized that the area, measured as the sum of transistor sizes, and the delay along a path of the circuit can be represented by *posynomial* functions of the sizes of transistors in the circuit. A posynomial is a function g of a positive variable $\mathbf{x} = [x_1, x_2 \cdots x_n] \in \mathbf{R}^{\mathrm{n}}$ that has the form

$$g(\mathbf{x}) = \sum_j \gamma_j \prod_{i=1}^{n} x_i^{\alpha_{ij}} \tag{2}$$

where the exponents $\alpha_{ij} \in \mathbf{R}$ and the coefficients $\gamma_j > 0$. Such a function has the useful property that it can be mapped onto a convex function through an elementary variable transformation, $(x_i) = (e^{z_i})$ [1].

In this paper, the delay of a circuit is defined to be the maximum of the delays of all paths in the circuit. Hence, it can be formulated as the maximum of posynomial functions. This is mapped by the above transformation on to a maximum of convex functions, which is also a convex function. The area function is also a posynomial, and is transformed into a convex function by the same mapping. Therefore, the optimization problem defined in (1) is mapped to a *convex programming* problem, i.e., a problem of minimizing a convex function over a convex constraint set. Due to the unimodal property of convex functions over convex sets, any local minimum of (1) is also a global minimum.

Most approaches model the delay of a CMOS gate as the Elmore time constant [2] of an equivalent RC network representing the circuit under the simplifying assumption that the input signals at the gate nodes of transistors are step functions. Such an assumption ensures that the delay function is posynomial [3], but is not realistic, since actual signals have nonzero rise or fall times. Hedenstierna and Jeppson [4] have developed a delay model for

CMOS inverters that creates an equivalent RC network for the inverter when the signals at the gate nodes of transistors have nonzero rise or fall times. This model is also posynomial, and has been adapted in the transistor sizing tool, MOGLO [5].

The most commonly used measure of the circuit area is given by an affine function of transistor sizes [3,5–12]. While this measure is not very accurate, it has the advantage of being a posynomial function of the sizes of transistors in the circuit.

Various methods have been used for optimization. TILOS [3, 6], performs the task by iteratively identifying a critical delay path, and using a heuristic method to reduce the delay along this path. The iterative process stops when the critical path (i.e., the largest delay path among all paths between a primary input and a primary output) meets the delay constraint. All transistors are initially set to the minimum size, and the sizes of only those transistors that lie on the critical path are increased, in an attempt to meet the delay constraint by increasing the sizes of as few transistors as possible. A subsequent algorithm proposed by Shyu et al. [7] works in two phases. It uses TILOS to generate a rough initial solution in the first phase. In the second phase, it converts the problem to a mathematical optimization problem in a smaller parameter space (corresponding to sizes of transistors on the paths of worst delay), and uses a method of feasible directions to find the optimal solution. The use of the reduced space serves to reduce the complexity of the optimization problem. iDEAS [8], like TILOS, iteratively reduces the delay along the critical path; it differs from TILOS in that it changes the size of more than one transistor in each iteration. The methods used by Cirit [9], Hedlund [10] and Marple [11, 12] formulate non-linear programs, and solve them by the method of Lagrangian multipliers. Another approach, as practised in MOSIZ [13], CATS [14] and iCOACH [15], is to perform the transistor size optimization as a two-step iterative process. The first step is an outer loop in which a timing 'budget', T_i, is assigned to each gate i, using a coarse simplification based on the overall delay specification. In the inner loop, the transistors in gate i are sized optimally so as to satisfy the timing budget, T_i, for that gate. The partitioning of the task into two steps serves to reduce the computational complexity of the algorithm.

There are several problems associated with the above optimization methods. Essentially, they perform a sequence of local optimizations over a reduced parameter space, hoping, but not guaranteeing, that such optimizations would lead to a global optimum. Moreover, apart from using the unimodality property, none of these algorithms takes advantage of the fact that the optimization problem can be posed as a convex programming problem.

With regard to delay modeling, each of the algorithms described in this section, except for [5], assumes waveforms with step transitions at the input and output of each gate. This is not realistic, since actual waveforms have non-zero rise and fall times. In [5], although delay models accommodate the effects of non-zero transition times, the accuracy of the optimization is compromised by choosing uniform widths W_n and W_p for all n-transistors and p-transistors, respectively, in a gate.

In this paper, we tackle the transistor sizing problem as defined in (1), which is the most common form of the problem faced by practising circuit designers. The other formulations mentioned earlier in this section can also be handled using the same approach.

We use a new and more accurate delay estimator that permits waveforms with non-zero rise and fall times, and computes rise and fall delays separately. The details of the delay estimation algorithm are furnished in Section II. An efficient convex programming method [16] is used for global optimization over the parameter space of all transistor sizes in a combinational subcircuit. This algorithm is capable of handling large problem sizes without having to prune any variables; moreover, its complexity is independent of the number of constraints. Hence, the optimization procedure is guaranteed to solve the problem *exactly* by finding the global minimum of the optimization problem, unlike many other problems which make simplifying assumptions for tractability, but cannot guarantee optimality *and* reasonable runtimes. The algorithm starts by bounding the convex domain by an initial polytope. By using special cutting plane techniques, the volume of this polytope is shrunk in each iteration, while ensuring the optimal solution lies within the boundary of the polytope. The iterative procedure stops when the volume of the polytope becomes sufficiently small. A more complete description is given in Section III. Since this is the first practical implementation of the convex programming algorithm [16] on problems of the size that we have handled, a considerable portion of this paper is devoted to practical aspects of the implementation. The extension of the algorithm from combinational circuits to general sequential circuits is outlined in Section IV. Finally, experimental results to illustrate the efficacy of this technique are presented in Section V.

II. THE DELAY ESTIMATION ALGORITHM

In this section, an algorithm for estimating the worst-case delay through the circuit, over all possible input combinations, is described.

Consider a combinational CMOS circuit with a set of primary input nodes and primary output nodes. The circuit is first divided into *channel-connected components* (henceforth referred to simply as *components*); each component corresponds to a set of transistors that are connected by drain and source nodes.

More formally, the definition of a component can be given by the following construction. Create an undirected graph with a vertex for each circuit node and an edge between the drain and source node of each transistor. Next, split the vertices corresponding to the ground node, the supply (V_{DD}) node, and the primary input nodes such that each of these vertices is incident on only one edge after splitting. A component is then a set of transistors corresponding to the edges within a connected component of the graph. This process is illustrated in Fig. 1.

The input nodes of a component consist of all the gate nodes of transistors in the component, and any drain or source node of a transistor in the component that is also a primary input. A component's output nodes include any drain or source node of a transistor in the component that is either a primary output, or a gate node of some transistor in the circuit.

A technique known as PERT (Program Evaluation and Review Technique) [17] is used to compute the maximum overall rise and fall delays between primary inputs and primary outputs of the circuit. A trace-back method is then used to obtain the *critical path*, which consists of the set of gates that lie on the largest delay path from a primary input to a primary output of the combinational network. Two numbers t_h and t_l are assigned to each output node of each component in the circuit, which correspond to the total rise and fall delay from the primary inputs, respectively. In addition, for each component, we compute Δ_h and Δ_l, the Elmore delays of an RC network that corresponds to the worst-case rise and fall scenarios, respectively. Additionally, the output transition waveform is modeled as a function that varies linearly with time. The transition times of the rising and falling waveforms at the output of the component are taken to be $2\Delta_h$ and $2\Delta_l$ respectively.

Fig. 2 shows the input waveform that triggers an output fall transition of an inverting gate in response to an input with an arrival time of $t_{h,max}$ and transition time τ. The definitions of t_l and Δ_l for the output response are illustrated in the figure; t_h and Δ_h are defined in a similar manner.

Finding the Worst-case Elmore Delay

A MOS transistor is modeled as a voltage-controlled switch with an on-resistance R_{on} between drain and source and three grounded capacitances C_d, C_s, and C_g at the drain, source, and gate terminals, respectively. The resistance and capacitances associated with a MOS transistor of channel width x are taken to have the following dependence [7] on x :

$$\begin{aligned} R_{on} &\propto 1/x \\ C_d, C_s, C_g &\propto x \end{aligned}$$

The PERT technique schedules components in order for evaluation. The waveform at an input node of a transistor in a scheduled component could be a steady logic 0, a steady logic 1, a logic 0 to logic 1 transition, or a logic 1 to logic 0 transition, corresponding to a switch that is either ON, OFF, or in transition. The worst-case Elmore delay at an output node of the component must be found over all possible input combinations. Let o denote an output node of the component. The algorithm for finding the worst-case fall delay at o is described below; the worst-case rise delay at o can be found in an analogous manner.

The component is represented by an undirected weighted graph, G, with an edge between the drain and source nodes of each transistor in the component. Edge weights are given by the resistance R_{on} of the corresponding transistor. The V_{DD} node and all of its incident edges are then removed from the graph. Let $t_{h,max}$ denote the maximum value of t_h among all input nodes of the component and suppose this occurs at the gate node of an n-type transistor corresponding to an edge e_{max} in G. It is assumed that the worst-case path is the *largest resistive path* (LRP) (i.e. the path of largest weight) between o and ground that passes through e_{max}. This assumption is valid when the load capacitance at the output node is much greater than the internal capacitance at any node that lies on any path between the output node and the ground node through e_{max}, as is often the case for CMOS circuits. As one pushes a component to its speed limit, internal node capacitances will no longer be small. However, since the capacitances that need to be driven by the component would probably increase, it is hoped that this assumption will hold. For the circuit delay specifications reported in this paper, it is seen in Section V that the approximation is valid.

Since finding the LRP is equivalent to the longest path problem in a graph which is NP-hard [18], we have developed a heuristic to perform this task. This heuristic is exact for series-parallel graphs, such as CMOS complex gates, and can be outlined as follows.

```
$T$ = maximum weighted spanning tree in $G$ containing $e_{max}$ such
      that the path $P$ between $o$ and ground in $T$ contains $e_{max}$
$maxW$ = sum of weights of edges in $P$
$LINK$ = edges in $G - T$
for each edge $e \in LINK$ {
        $T_1 = T \cup e$
        $P_1 =$ max weight $o$-to-ground path in $T_1$ through $e_{max}$
        $W =$ sum of weights of edges in $P_1$
        if ( $W > maxW$ ) {
                $e' =$ any edge in $P - (P \cap P_1)$
                $T = T_1 - e'$, $P = P_1$, $maxW = W$
        }
}
```

The heuristic begins by finding a maximum weighted spanning tree T of G that contains the edge e_{max}, using a variant of Prim's algorithm [18]. Let P' denote the *unique* path in T between o and ground. If P' contains e_{max}, set P to P'; otherwise an edge, $e \notin T$, is added to T such that $T + e$ has a path P between o and ground through e_{max}, and the e is the edge of greatest weight among all edges that satisfy this condition. The introduction of e creates a unique cycle; an edge e', such that $e' \in P'$ and $e' \notin P$, is removed from $T + e$, to give a new initial tree T.

The edges which are not in T constitute the set of *links*. A link is then added to the present tree T to produce a subgraph T_1 that contains a *unique cycle*. Therefore, there can be at most two paths from o to ground in T_1. The path of larger weight is called P_1. If the weight of P_1 is larger than that of P, then the present tree T is updated by removing any edge from T_1 that belonged to P but not to P_1. Also, P is reset to P_1 and the heuristic proceeds to process the next link, and so on, until all links of the original tree have been processed. The path between o and ground in the final tree produced by the heuristic is referred to as the largest resistive path (LRP). In case of series-parallel graphs, the heuristic does indeed generate the path of largest resistance from output to ground; in other cases (such as graphs with bridges), it gives a good approximation.

Now, consider any spanning tree T_w of the graph G. If P_p and P_q are the paths to ground from nodes p and q, respectively, in T_w, let R_{pq} denote the resistance of the path $P_p \cap P_q$. The Elmore delay [2] between o and the ground node in the RC-tree represented by T_w is given by

$$\sum_{j \in T_w} R_{oj} C_j \tag{3}$$

where C_j is the capacitance to ground at node j in T_w. Note that while finding the Elmore delay, the capacitances which lie between the switching transistor and the supply rail are assumed to be at the voltage level of the supply rail at the time of the switching transition, and do not contribute to the Elmore delay.

In order to find a tree that contains the LRP and which maximizes the Elmore delay, certain edges must be added to the LRP in such a way that R_{oj} is maximized for every node j in the graph. The algorithm to construct the worst-case tree T_w from the LRP is as follows. Initially T_w is taken to be the LRP itself. For a node $n_1 \notin T_w$ the algorithm finds a node $n_2 \in T_w$ that is farthest from the ground node and is connected to n_1 by a path that does not intersect T_w. This path is then added to T_w and the procedure is repeated until all nodes of G are included in the tree T_w. The worst-case fall delay at o is then computed using (3).

Example 1: Consider the graph G shown in Fig. 3. Assume that the LRP between the output node o and ground has been found to be **d,e** . Initially, T_w is taken to be the LRP **d,e** . Consider node n_1 which is connected to node o through several paths, one of which is

j,k . This path is added to T_w which now becomes **d,e,j,k** . Note that both nodes n_1 and n_2 are now part of the tree T_w. The nodes n_4 and n_5 are then added to the tree by adding the edges **a** and **b** respectively. Finally, the node n_6 is added to the tree by adding the edge **f** to it. This completes the formation of the worst-case tree which is **d,e,j,k,a,b,f** indicated by the bold edges in Fig. 3. If branch **d** corresponds to the switching transistor, the worst-case Elmore delay is given by

$$(R_d + R_e)(C_o + C_{n_4} + C_{n_5} + C_{n_2} + C_{n_1}) \tag{4}$$

Finally, the value of t_l for output node o is computed by adding $t_{h,max}$, the Elmore delay of the worst-case RC network, Δ_l, and a term [4] related to the transition time of the rising input at the input node corresponding to the worst-case Elmore fall delay.

A more detailed description of how the effect of input transition time is incorporated is provided later in this section. This procedure is repeated for all output nodes of the component.

The value of t_h, the worst-case rise delay at each output node of the component, can be found in a similar manner. The weighted graph representing the component is constructed as before except that the ground node is removed instead of the V_{DD} node. The rest of the procedure to find the worst-case Elmore rise delay is identical to that of the fall delay except that the role of the ground node is replaced by the V_{DD} node, and the roles of t_h and Δ_h are exchanged with those of t_l and Δ_l in the fall delay case.

In other delay estimators that we have come across, the Elmore rise and fall delays are computed directly from the LRP without appending additional edges to extend it to the worst-case RC-tree as described above.

Delay Model for Components under Nonstep Transitions

In [4], it has been shown that a good approximation to the delay, Δ, of a CMOS inverter under excitation from a nonstep input with rise time τ, i.e.,

$$v_{in} = \begin{cases} 0 & t < 0 \\ \frac{t}{\tau} \cdot V_{DD}, & 0 < t < \tau \\ V_{DD}, & t > \tau \end{cases} \tag{5}$$

is given by

$$\Delta = \Delta_{step} + \frac{\tau}{6} \left[1 + \frac{2V_{Tn}}{V_{DD}}\right] \tag{6}$$

where

V_{Tn}	=	Threshold voltage of nMOS transistor
V_{DD}	=	Supply voltage
Δ_{step}	=	Transition delay of the inverter under a step input excitation

Δ is defined as the difference between the time when the output signal crosses the $V_{DD}/2$ level, and the time at which input signal reaches $V_{DD}/2$. We model the falling output signal using a form similar to the input waveform v_{in} in (5). The relationship between the input and output signals in our model, for the falling output transition, is shown in Fig. 2.

A general complex gate such as the AOI gate, when excited by a step excitation, may be represented by an equivalent inverter I whose size is determined by the Elmore delay of the worst-case RC tree described earlier. For an excitation of the type in (5), we may consider the general complex gate as being equivalent to the inverter I being excited by the same excitation. Hence, (6) also holds for complex gates.

The form of the path delay under step excitations is described in [3]. We examine the change required in this form to include the effect of waveforms with nonstep transitions as described in (5), under the assumption that the signal at the output of a component is modeled by a ramp function, as described earlier in this section.

Let $\Delta_{i,step}$ refer to the delay of component i on a path of the circuit, with all input waveforms having step transitions. The delay of the circuit, $Delay_{step}$, is given by

$$Delay_{step} = \Delta_{1,step} + \Delta_{2,step} + \cdots + \Delta_{n,step} \tag{7}$$

When we incorporate the effect of the transition time, and add the simplifying assumption that the magnitude of the threshold voltage is the same for nMOS and pMOS enhancement mode transistors, the delay along the path is given by

$$\begin{aligned} Delay_n &= \Delta_{1,step} + [\Delta_{2,step} + \alpha \cdot Delay_1] \\ &\quad +[\Delta_{3,step} + \alpha \cdot (Delay_2 - Delay_1)] \\ &\quad +[\Delta_{4,step} + \alpha \cdot (Delay_3 - Delay_2)] + \cdots \\ &\quad +[\Delta_{n,step} + \alpha \cdot (Delay_{n-1} - Delay_{n-2})] \\ &= \Delta_{1,step} + \Delta_{2,step} + \cdots + \Delta_{n-1,step} + \Delta_{n,step} + \alpha \cdot Delay_{n-1} \\ &= w_1 \cdot \Delta_{1,step} + w_2 \cdot \Delta_{2,step} + \cdots + w_n \cdot \Delta_{n,step}. \end{aligned} \tag{8}$$

where

$$\begin{aligned} Delay_k &= \text{circuit delay up to k}^{\text{th}} \text{ component from primary inputs} \\ \alpha &= \frac{1}{3} \cdot \left[1 + \frac{2 \mid V_T \mid}{V_{DD}}\right] \\ V_T &= \text{threshold voltage (assumed equal in magnitude for} \\ &\quad \text{nMOS, pMOS for simplicity)} \\ w_k &= \sum_{i=0}^{n-k} \alpha^i. \end{aligned}$$

Thus, the delay is expressed by the weighted sum of the $\Delta_{i,step}$ values. Since each of the $\Delta_{i,step}$ expressions is posynomial [3], and the w_i's are constant, the expression for delay along a path under excitations with nonstep transitions is a posynomial.

In the case where the threshold voltage, V_T is different in magnitude for n and p type transistors, the form of (8) remains the same, but the expression for the w_k's is more involved. In this work, we have assumed that the magnitude of the threshold voltage is the same for n and p type transistors.

As will be shown in Section V, the delay times calculated by our estimator are in good agreement with SPICE results.

Area and Delay Functions

Let n denote the number of transistors in a combinational circuit and let $\mathbf{x} = [\mathrm{x}_1, \mathrm{x}_2, \cdots, \mathrm{x}_\mathrm{n}]$ be an n-dimensional vector of the transistor sizes. The total area of the circuit is taken, for simplicity, to be the sum of the transistor sizes, i.e.,

$$Area(\mathbf{x}) = \sum_{i=1}^{n} x_i. \tag{9}$$

Note that the area function is a posynomial in $\mathbf{x}$.

The equation for the overall delay $Delay(\mathbf{x})$ through the critical path, using our gate-delay model, has been shown to be a posynomial of the form

$$Delay(\mathbf{x}) = \sum_j \gamma_j \prod_{i=1}^{n} x_i^{\alpha_{ij}} = \sum_j \gamma_j x_1^{\alpha_{1j}} x_2^{\alpha_{2j}} \cdots x_n^{\alpha_{nj}} \tag{10}$$

where

$$\gamma_j \geq 0,\ \alpha_{ij} \in \{-1, 0, 1\}\ \forall\, i = 1, 2, \cdots, n.$$

Also, α_{ij} may be -1 only for *critical transistors*, i.e., transistors that lie in the LRP of a component on the critical path. This is because the delay is expressed as a sum of RC products. The only transistors that contribute terms with an exponent of '-1' to these RC products are those that act as resistances, i.e., the critical transistors. Any other transistor may either contribute a term with exponent '1', when it acts as a capacitance, or may make no contribution to the RC product.

III. THE CONVEX PROGRAMMING ALGORITHM

The objective of the algorithm is to solve the following transistor sizing problem

$$\begin{aligned} \textsf{minimize} \quad Area(\mathbf{x}) &= \sum_{i=1}^{n} x_i \\ \textsf{subject to} \quad Delay(\mathbf{x}) &\leq T_{spec} \end{aligned} \tag{11}$$

where the delay $Delay(\mathbf{x})$ is maximum of delays along all paths to a primary output node of the circuit. By making the variable transformation

$$(x_i) = (e^{z_i})$$

the original transistor sizing problem (11) of minimizing a posynomial area function over posynomial constraints becomes

$$\begin{aligned} \textsf{minimize} \quad Area(\mathbf{z}) &= \sum_{i=1}^{n} e^{z_i} \\ \textsf{subject to} \quad D(\mathbf{z}) &\leq T_{spec} \end{aligned} \tag{12}$$

The other formulations mentioned in the introduction, namely, minimizing the delay subject to area constraints, minimizing the area-delay product, or a formulation that involves the area, delay and power dissipation, can also be handled by this algorithm. However, since the above formulation is the most practically useful one, we restrict our discussion to this formulation.

Note that under this transformation, the delay along a path has the form

$$\sum_{j} \gamma_j e^{\sum_{i=1}^{n} \alpha_{ij} z_i}$$

which is a convex function. Since the circuit delay is defined to be the maximum of all path delays, and the maximum of convex functions is also convex, $D(\mathbf{z})$ is a convex function. It can be seen that $Area(\mathbf{z})$ is also a convex function of $\mathbf{z}$. Hence, (12) is a *convex programming* problem of minimizing a convex function over a convex set of constraints.

The algorithm proposed by Vaidya in [16] provides an efficient technique for solving (12). Define the *feasible set*

$$S = \{\mathbf{z} \in \mathbf{R}^n : D(\mathbf{z}) \leq T_{spec}\} \tag{13}$$

and let $\mathbf{z}_{opt}$ be the solution to (12). Initially, a polytope P that contains $\mathbf{z}_{opt}$ is chosen. It is of the form

$$P = \{\mathbf{z} : A\mathbf{z} \geq \mathbf{b}\} \tag{14}$$

where $A \in \mathbf{R}^{m \times n}$ and $\mathbf{b} \in \mathbf{R}^m$. Here, m denotes the number of linear inequality constraints describing the polytope. The initial polytope P, for example, may be selected to be an n-dimensional box describing the set

$$\{\mathbf{z} : \log_e(x_{min}) \leq z_i \leq \log_e(x_{max})\} \tag{15}$$

where x_{min} and x_{max} are the user-specified minimum and maximum allowable transistor sizes, respectively. Thus, this system naturally incorporates upper and lower bound constraints on transistor sizes.

The algorithm proceeds iteratively as follows. First, a *center* $\mathbf{z}_c$ deep in the interior of the current polytope P is found by using a technique which will be described later. Next, an *oracle* is then invoked to determine whether or not the center $\mathbf{z}_c$ lies within the feasible region S. From the definition of S, the oracle is simply a routine that invokes the delay estimator described in Section II, with the transistor sizes $x_i = e^{z_{c,i}}$, to determine whether or not the delay requirement is met. If the point $\mathbf{z}_c$ lies outside S, it is possible to find a *separating hyperplane* passing through $\mathbf{z}_c$ that divides the polytope P into two parts, such that S lies entirely in the part satisfying the constraint

$$\mathbf{c}^T\mathbf{z} \geq \beta \tag{16}$$

where

$$\mathbf{c} = -[\nabla D_{critpath}(\mathbf{z})]^T \tag{17}$$

is the negative of the gradient of the critical path delay (constraint) function, and

$$\beta = \mathbf{c}^T \mathbf{z}_c \tag{18}$$

The separating hyperplane described above corresponds to the tangent plane to the path delay along the critical path. Note that the discontinuity of the derivative of the circuit delay function does not affect matters, since we only deal with the gradient of a path delay, which is a continuous function.

If the point $\mathbf{z}_c$ lies within the feasible region S, then there exists a hyperplane that divides the polytope into two parts such that $\mathbf{z}_{opt}$ is contained in one of them satisfying the constraint (16) with

$$\mathbf{c} = -[\nabla Area(\mathbf{z})]^T \tag{19}$$

being the negative of the gradient of the area (objective) function, and β is once again defined by (18). In either case, the constraint (16) is added to the current polytope to give a new polytope that has roughly half the original volume. The process is repeated until the polytope is sufficiently small.

Since this is the first practical implementation of this convex programming algorithm on problems of the size that we have handled, our work addresses several issues that were inconsequential to previous implementations that worked with a smaller number of variables. Hence, a description of some of the practical issues involved is provided in some detail in this section.

Example 2: Consider the problem

$$\begin{aligned} \text{minimize} \quad & f(x_1, x_2) \\ \text{s.t.} \quad & (x_1, x_2) \in \text{S} \end{aligned}$$

where S is a convex set and f is a convex function. The shaded region in Fig. 4(a) corresponds to S, and the dotted lines show the level curves of the function f. The point $\mathbf{x}^*$ is the solution to this problem. The procedure begins by bounding the expected solution region by a closed polytope, which corresponds to a rectangle in two dimensions. This is shown in Fig. 4(a). The center, $\mathbf{z}_c$ of this rectangle is found. The oracle is invoked to determine whether $\mathbf{z}_c$ lies within the feasible region or not; in this case it can be seen that $\mathbf{z}_c$ lies outside the feasible region. Hence, the gradient of the constraint function is used to pass a hyperplane through $\mathbf{z}_c$, such that the polytope is divided into two parts, one of which contains the solution $\mathbf{x}^*$. This is illustrated in Fig. 4(b), where the hatched region corresponds to the polytope containing the solution. The process is repeated on this new smaller polytope. Its center lies inside the feasible region, and hence the gradient of the objective function is used to generate a hyperplane that further shrinks the size of the polytope, as shown in Fig. 4(c). The result of another iteration is illustrated in Fig. 4(d). The process continues until the polytope has been shrunk sufficiently.

It can be seen that the key parts of this algorithm are

(1) finding the center $\mathbf{z}_c$ of the existing polytope P,

(2) generating gradient functions in (17) and (19) above, and

(3) deciding when to terminate the algorithm.

Procedure for finding the center of the polytope

We would like to find a point inside a polytope that satisfies the property that any separating hyperplane drawn through it divides the original polytope into two parts of approximately equal volume. Finding such a point is difficult [16], and so we settle for finding a point that is reasonably deep within the interior of the polytope, and can be found through relatively inexpensive computation.

Consider a polytope P defined by (14), and let $\mathbf{a_i}^T$ be the i^{th} row of the $m \times n$

matrix A, and b_i be the i^{th} element of the m-dimensional vector b. The center $\mathbf{z}_c$, is taken to be the vector that minimizes the following *log-barrier function*

$$F(\mathbf{z}) = -\sum_{i=1}^{m} \log_e(\mathbf{a_i}^T\mathbf{z} - b_i) \tag{20}$$

Note that near the boundary of the polytope, $F(\mathbf{z})$ tends to infinity and its value decreases as one moves *deeper* into the interior of the polytope. Also, the value of $F(\mathbf{z})$ is undefined outside the boundary of the polytope. Moreover, F is a convex function of $\mathbf{z} \in P$, with a $1 \times n$ gradient vector

$$\nabla F(\mathbf{z}) = -\sum_{i=1}^{m} \frac{\mathbf{a_i}^T}{(\mathbf{a_i}^T\mathbf{z} - b_i)} \tag{21}$$

and an $n \times n$ Hessian matrix

$$H(\mathbf{z}) = \nabla^2 F(\mathbf{z}) = \sum_{i=1}^{m} \frac{\mathbf{a_i}\mathbf{a_i}^T}{(\mathbf{a_i}^T\mathbf{z} - b_i)^2} \tag{22}$$

Since the initial polytope is a box, its center is easy to find. At each subsequent iteration, a constraint of the form $\mathbf{c}^T\mathbf{z} \geq \beta$ is added to the previous polytope whose center is found iteratively using the Newton's method [19] as follows. The initial point $\mathbf{z}_0$ for the Newton's method is found by moving halfway to the closest boundary in the direction $\mathbf{c}$. The initial point $\mathbf{z}_0$ thus obtained is guaranteed to be in the interior of the new polytope.

The Newton's method for finding the center $\mathbf{z}_c$ then generates iterates of the form

$$\mathbf{z}_{k+1} = \mathbf{z}_k + t^*\xi_k \tag{23}$$

for $k = 0, 1, 2, \cdots$, until convergence, where ξ_k is the *Newton direction* at $\mathbf{z}_k$ given by

$$\xi_k = -H^{-1}(\mathbf{z}_k)[\nabla F(\mathbf{z}_k)]^T = -[\nabla^2 F(\mathbf{z}_k)]^{-1}[\nabla F(\mathbf{z}_k)]^T \tag{24}$$

and t^* is the point that minimizes the one-dimensional function

$$\phi(t) = F(\mathbf{z}_k + t\xi_k) \tag{25}$$

and is obtained by performing a one-dimensional line-search.

Note that the process of computing a Newton direction by (24) involves the inversion of an $n \times n$ Hessian matrix which takes O(n^3) time and can prove to be rather expensive. This expense can be cut down by maintaining the inverse of an approximate Hessian $\hat{H}$ via rank-one updates [19] as described below, and by using an approximate Newton direction $\hat{\xi}_k$ instead of ξ_k in the line search. We note that using an approximate Newton direction instead of the exact one essentially does not affect the convergence properties of the center-finding algorithm [16].

Rank-one updates

Let $\mathbf{z}_k$ be the point at the beginning of the $(k+1)^{th}$ iteration of Newton's method for finding the center $\mathbf{z}_c$ of the polytope P described by (14).

Two methods for maintaining the approximate Hessian, using rank-one updates [19] are outlined below.

Method 1

The Hessian at $\mathbf{z}_k$ may be written as

$$H(\mathbf{z}_k) = \nabla^2 F(\mathbf{z}_k) = \sum_{i=1}^{m} \frac{\mathbf{a_i}\mathbf{a_i}^T}{(\mathbf{a_i}^T\mathbf{z}_k - b_i)^2} = A^T \Lambda A \tag{26}$$

where Λ is a diagonal matrix.

Let $\delta_1, \delta_2 > 0$ be small parameters. An approximate Hessian is given by

$$\hat{H} = A^T \hat{\Lambda} A \tag{27}$$

where $\hat{\Lambda} \in \mathbf{R}^{m \times m}$ is a diagonal matrix such that at the start of the $(k+1)^{th}$ iteration, the i^{th} diagonal entry of $\hat{\Lambda}$, λ_{ii} satisfies the condition

$$(\mathbf{a_i}^T \mathbf{z}_{k-1} - b_i)^{-2} \delta_1^{-1} \leq \lambda_{ii} \leq (\mathbf{a_i}^T \mathbf{z}_{k-1} - b_i)^{-2} \delta_2 \ \ \forall \ \ 1 \leq i \leq m. \tag{28}$$

We maintain an approximate inverse Hessian, $\mathcal{H}^{-1}$; the following rank-one correction procedure is used to update $\mathcal{H}^{-1}$ at the beginning of the $(k+1)^{th}$ iteration.

```
For each i = 1 , 2 , $\cdots$ , $m$ {
    if ($\lambda_{ii} < (\mathbf{a_i}^T \mathbf{z}_k - b_i)^{-2} \delta_1^{-1}$) or
       ($\lambda_{ii} > (\mathbf{a_i}^T \mathbf{z}_k - b_i)^{-2} \delta_2$) then {
            $\omega = (\mathbf{a_i}^T \mathbf{z}_k - b_i)^{-2} - \lambda_{ii}$
            $\lambda_{ii} = (\mathbf{a_i}^T \mathbf{z}_k - b_i)^{-2}$
            $\mathbf{e} = \mathcal{H}^{-1} \mathbf{a_i}$
            $\mu = \omega (1 + \omega \mathbf{a_i}^T \mathbf{e})^{-1}$
            $\mathcal{H}^{-1} = \mathcal{H}^{-1} - \mu \mathbf{e}\mathbf{e}^T$
    }
}
```

One of two schemes may be used to calculate the approximate Newton direction : (a) maintaining a more accurate $\mathcal{H}^{-1}$, and setting

$$\hat{\xi}_k = -\mathcal{H}^{-1} (\nabla F(z_k))^T \tag{29}$$

It can easily be verified that each rank-one update to $\mathcal{H}^{-1}$ is of complexity O(n^2). Typically, the number of updates to $\hat{\Lambda}$ per iteration is less than $O(\sqrt{n})$ and this reduces the average cost of an iteration of the center finding algorithm from O(n^3) to O($n^{2.5}$).

(b) maintaining a more approximate $\mathcal{H}^{-1}$, and using it as a preconditioner for a preconditioned conjugate gradient method [20] that solves

$$H \hat{\xi}_k = -(\nabla F(z_k))^T \tag{30}$$

This method trades off the cost of maintaining $\mathcal{H}^{-1}$ accurately against the cost of performing a few iterations of the preconditioned conjugate gradient method.

For Scheme (a) for maintaining an approximate inverse Hessian described above, the parameter δ_1 above is typically chosen to be around 1.5, while δ_2 may be set to about 5, while for Scheme (b), typical values for δ_1 and δ_2 are 3 and 20 respectively.

The reason why δ_2 is set to be larger than δ_1 is as follows. When ω is positive (i.e., when δ_2 determines whether an update is to be made or not), the denominator of μ is relatively large, and hence numerical errors in the calculation of μ are damped out. In the case where ω is negative (i.e. the update decision is dependent on the value of δ_1), the denominator of μ grows smaller as δ_1 increases, and a large δ_1 could lead to an amplification of numerical errors. small. Therefore, the choice of δ_2 may be more liberal than that of δ_1.

In each of these two methods, it suffices to maintain $\mathcal{H}^{-1}$; it is not even necessary to explicitly find $\hat{H}$.

Method 2

The Hessian at $\mathbf{z}_k$ may also be written as

$$H = ,\ + U^T \Omega U \tag{31}$$

Let p be the number of additional planes added to the initial polytope, the box, described by (15). , $\in \mathbf{R}^{n\times n}$ is the Hessian at $\mathbf{z}_k$ due to the planes of this box only, and is a diagonal matrix. The i^{th} diagonal element of , , denoted γ_{ii}, is given by

$$\gamma_{ii} = \left[[\mathbf{z}_{k,i} - x_{min}]^{-2} + [x_{max} - \mathbf{z}_{k,i}]^{-2} \right]^{-1}$$

The rows of $U^T \in \mathbf{R}^{p\times n}$ correspond to the planes added to the initial polytope, i.e., the $(2n+1)^{th}$ to the m^{th} rows of A^T. $\Omega \in \mathbf{R}^{p\times p}$ is a diagonal matrix, whose diagonal entries correspond to the last p diagonal entries of the matrix Λ in (26).

We may now write

$$H^{-1} \;=\; ,^{-1} - ,^{-1} U \left[\Omega^{-1} + U^T ,^{-1} U \right]^{-1} U^T ,^{-1} \tag{32}$$

$$= \;,^{-1} - \,,^{-1} U \mathrm{C}^{-1} U^T \,,^{-1} \tag{33}$$

where $\mathrm{C} = \Omega^{-1} + U^T \,,^{-1} U$

We maintain an approximation $\mathcal{C}^{-1}$ to C^{-1}. An approximate inverse Hessian is then given by

$$\mathcal{H}^{-1} = \,,^{-1} - \,,^{-1} U \mathcal{C}^{-1} U^T \,,^{-1} \tag{34}$$

As in Method 1, it suffices to maintain the approximate inverse of C; it is not necessary to explicitly store C itself. The approximate Hessian or the approximate inverse Hessian are never explicitly maintained; the search direction is found by computing $\xi = -\mathcal{H}^{-1}[\nabla F(\mathbf{z}_k)]^T$, which involves multiplication of the expression (34) for $\mathcal{H}^{-1}$ by a $n \times 1$ vector. The cost of this computation can be seen to be O(np) (if $n \gg p$), i.e., the number of added planes is much less than the problem dimension. This is seen to be the case for large problems, and hence the use of this method would speed up the computation substantially for large problems.

If the number of additional planes, p has not changed since the last calculation of $\mathcal{C}^{-1}$, all that needs to be done to get the new $\mathcal{C}^{-1}$ is a set of rank-one updates. If a new plane has been added, a method outlined in [21] may be used to update $\mathcal{C}^{-1}$. The method involves a rank-one update and a few additional operations to incorporate the effect of the newly-added plane. As before, one of two schemes may be used to calculate the approximate Newton direction:

(a) Maintaining a more accurate $\mathcal{C}^{-1}$, and setting

$$\hat{\xi}_k = -\mathcal{H}^{-1}(\nabla F(\mathbf{z}_k))^T \tag{35}$$

(b) Maintaining a more approximate $\mathcal{C}^{-1}$, and using it to as a preconditioner for a preconditioned conjugate gradient iteration that solves

$$H\hat{\xi}_k = -(\nabla F(z_k))^T \tag{36}$$

It may be noted that the preconditioned conjugate gradient does not need an approximate H or $\mathcal{H}^{-1}$ explicitly, but multiplies $\mathcal{H}^{-1}$ by a $n \times 1$ vector; we have already seen that this operation is computationally cheap when p is small.

It was found experimentally that Scheme (b) of Method 2 gave the best overall results for the problems that we worked on.

One-dimensional Line Search

Once the Newton direction ξ_k of (24) has been found, the value of t^* that minimizes the one-dimensional function $\phi(t)$ defined by (25) is obtained as follows. First, the allowable values of t are bounded by t_{min} and t_{max}, where t_{max} is found by computing the distance from the point $\mathbf{z}_k$ to the nearest boundary of the polytope along the ξ_k direction. The derivative of ϕ in the interval $[0, t_{max}]$ can be shown to be

$$\phi'(t) = \nabla F(\mathbf{z}_k + t\xi_k) \cdot \xi_k = -\sum_{i=1}^{m} \frac{s_i}{s_i t + r_i} \tag{37}$$

where $s_i = \mathbf{a_i}^T \xi_k$ and $r_i = \mathbf{a_i}^T \mathbf{z}_k - b_i$ for each $i = 1, 2, \cdots, m$. Note that

$$\phi'(0) = \nabla F(\mathbf{z}_k) \cdot \xi_k = -\nabla F(\mathbf{z}_k) \cdot H^{-1} \cdot [\nabla F(\mathbf{z}_k)]^T < 0 \tag{38}$$

since the Hessian of F, a convex function is positive definite. Also,

$$\lim_{t \to t_{max}} \phi'(t) > 0 \tag{39}$$

As a result of (38) and (39), and since the function ϕ is convex in the interval $[t_{min}, t_{max}]$, t_{min} can be set to 0, and a simple bisection search can be used to find t^* at which $\phi'(t^*) = 0$ as follows :

```
repeat {
        $t^* = (t_{min} + t_{max})/2$
        if ($\phi'(t^*)$ and $\phi'(t_{min})$ are of opposite sign )
                $t_{max} = t^*$
        else
                $t_{min} = t^*$
        }
until ( $| \phi'(t^*) | < \epsilon$)
```

where ϵ is a small positive number.

Generation of hyperplanes

When the center $\mathbf{z}_c$ of a polytope lies within the feasible region S, the gradient of the area function is required to generate the new hyperplane passing through the center. The area function of (12) has a gradient at the point $\mathbf{z}$ given by

$$\nabla Area(\mathbf{z}) = [e^{z_1}, e^{z_2}, \cdots, e^{z_n}] \tag{40}$$

In the case when the center $\mathbf{z}_c$ lies outside the feasible region S, the gradient of the critical path delay function $D_{critpath}(\mathbf{z}_c)$ defined by (10) is required to generate the new hyperplane that is to be added. For each $k = 1, 2, ..., n$, the k^{th} component of the required gradient vector at a point $\mathbf{z}$ is given by (see equation (10))

$$[\nabla D_{critpath}(\mathbf{z})]_k = \sum_j \gamma_j \alpha_{kj} e^{\sum_{i=1}^n \alpha_{ij} z_i} \tag{41}$$

Note that the transistors in the circuit can contribute to the k^{th} component of the gradient of the delay function in either of two ways :

(a) If the k^{th} transistor is critical (*i.e.*, it lies on the LRP of a component on the critical path of the circuit), or

(b) If the k^{th} transistor is a capacitive load for some critical transistor.

Transistors that satisfy neither of these two requirements have no contribution to the gradient of the delay function.

Termination criterion

The algorithm should be terminated when the volume of the final polytope is sufficiently small. In practice, near the optimum, the polytope becomes flat in the direction normal to the gradient of the area. A practical termination criterion uses this property.

From the current center, $\mathbf{z}_c$, let $\mathbf{z}_1$ and $\mathbf{z}_2$ be the two nearest points on the boundary of the polytope, in the direction of the positive and negative gradient of the area respectively. The difference between the area of the circuit corresponding to the transistor sizes at $\mathbf{z}_1$ and $\mathbf{z}_2$ provides a measure of the flatness of the polytope in the direction of the area gradient. Hence, the termination criterion is taken to be

$$\frac{\mid Area(\mathbf{z}_1) - Area(\mathbf{z}_2) \mid}{Area(\mathbf{z}_c)} < \epsilon \tag{42}$$

where ϵ is a small user-specified number (a reasonable default value is 0.01).

IV. EXTENSION TO SEQUENTIAL CIRCUITS

For sizing sequential circuits, it is first required that latches in the circuit be identified. Next, the combinational subcircuits that lie between these latches are extracted, and the delay constraint for each of these subcircuits is computed. For each subcircuit, the transistor sizing problem is solved by minimizing the area of the subcircuit, while ensuring that its delay requirement is satisfied.

The task of identifying latches proceeds as follows. The circuit is represented by a graph, G, with vertices corresponding to components, and with edges drawn from a component to each component that it fans out to. Feedback loops in the circuit (e.g. cross-coupled NAND gates), which manifest themselves as strongly connected components in this graph, are identified using Tarjan's algorithm [22].

Next, each clock signal is traced from the primary inputs, proceeding from a component to each of the components that it fans out to, until the signal intersects either a feedback loop or a transmission gate. Such a feedback loop or transmission gate is identified as a latch. Thus, this procedure identifies latches which are clocked not only by clock signals at the primary input, but also by qualified clock signals.

All latches are then removed from the circuit. In case of transmission gate latches, this could result in a single component being broken up into two or more components. A new graph $\hat{G}$ is formed, in the same way as G, to represent this new circuit. A breadth-first search of $\hat{G}$ can detect strongly connected components of this new circuit; each such strongly connected component corresponds to a combinational subcircuit that lies between a set of input latches and a set of output latches. From the clock arrival times at these latches, the timing requirements for the combinational subcircuits can be found.

V. EXPERIMENTAL RESULTS

The algorithms described in the previous sections have been implemented in iCONTRAST (**i**llinois **C**onvex **O**ptimization-based **N**ovel **TRA**nsistor **S**izing **T**ool). The program, written in C, now consists of approximately 6000 lines of code.

The input to the program is a SPICE deck that gives a transistor-level netlist of the circuit. In the preprocessing stage, the circuit is first divided into channel-connected components. Next, latches in the circuit are identified. The circuit is divided into combinational subcircuits that lie between latches, and the delay constraints for each such subcircuit are determined. The main body of the procedure carries out a convex optimization on each combinational subcircuit.

It must be mentioned here that for our experimental results, the approximate

Hessian for finding the Newton direction was maintained using Scheme (b) of Method 2 described in Section III.

A set of test circuits described in Table 1 were used to evaluate the performance of iCONTRAST. The entries under *unsized area* and *unsized delay* correspond to the area and delay when all transistors in the circuit are set to the minimum size. In case of the sequential circuit, the delay refers to the maximum stage delay for the circuit. It may be noted that the word 'area' refers to the sum of transistor sizes. The technology parameters used here correspond to a submicron technology. The number of iterations for each circuit were of the order $O(n)$. For these circuits, the initial polytope was taken to be a box with the minimum transistor size being 1.8, and the maximum size being 500.

Table 1 : Circuits used to evaluate iCONTRAST

Circuit	Description	# Transistors	Unsized Area (μm)	Unsized Delay
Inv6	6-inverter Chain	12	21.6	7.0ns
Inv10	10-inverter Chain	20	36.0	12.6ns
Tree	Tree of NAND gates	28	50.4	4.0ns
Add2	2-bit Adder	52	93.6	24.2ns
Add8	8-bit Adder	208	374.4	109.8ns
Add32	32-bit Adder	832	1497.6	452.5ns
Seq	Sequential circuit	244	439.2	26.9ns

Table 2 shows the area of the circuit after it has been sized by iCONTRAST to meet a delay specification, T_{spec}, and the execution time on a Sun SPARCstation I. Since our method solves the underlying convex programming problem exactly, the areas shown here correspond to the globally optimum solution to the transistor sizing problem, with an accuracy that is dictated by the tightness of the user-specified termination criterion. The number of iterations, and the memory requirement for each circuit are also shown. In case of the sequential circuit, Seq, the number of iterations corresponds to the maximum number of iterations required to size any combinational subcircuit.

Consider, for example, the results on the example circuit, Add8. As seen in Table 1, the unsized area and delay for this circuit are 374.4 μm, and 109.8 ns respectively. The area penalty required to achieve a relatively loose delay specification such as the first one, 100 ns, is not very large; the active area of the sized circuit is only 11% larger than the unsized circuit. As the delay specification becomes tighter, the area penalty increases non-linearly; to achieve a delay specification of 40 ns, the active area of the sized circuit is 182 % larger than that of the unsized circuit. A similar trend is visible for each of the other example circuits in Table 2.

The number of iterations and the memory requirement are seen to increase slightly in most cases with the tightness of the delay specification. For the largest circuit, however, the number of iterations is seen to be roughly independent of the delay specification.

None of these results violates the theoretical prediction that the order of magnitude of the number of iterations for a given circuit is dependent only on the size of the initial polytope (which was the same for all circuits) and is independent of the delay specification. The basis for this prediction lies in the fact that the volume of the polytope is roughly halved in each iteration; hence, the volume of the polytope containing the solution is roughly the same after the same number of iterations, regardless of where the solution lies within the initial polytope.

Table 2 : Results on sizing various circuits using iCONTRAST

Circuit	T_{spec}	Sized Area (μm)	Execution Time	# Iterations	Memory requirement
Inv6	5.0ns	29.2	1.2s	26	648 KB
	4.0ns	40.9	1.7s	31	652 KB
	3.0ns	72.2	2.0s	34	656 KB
	2.0ns	244.8	2.9s	39	660 KB
Inv10	10.0ns	45.2	2.5s	31	812 KB
	8.0ns	62.3	2.4s	33	816 KB
	6.0ns	110.0	4.1s	50	848 KB
	5.0ns	177.4	5.2s	59	860 KB
	4.5ns	251.0	6.6s	62	856 KB
Tree	3.5ns	58.8	11.8s	53	796 KB
	3.0ns	74.8	12.3s	64	784 KB
	2.5ns	104.5	14.1s	76	820 KB
	2.0ns	174.7	18.3s	93	836 KB
	1.5ns	407.0	20.6s	108	880 KB
Add2	18.0ns	114.3	33.5s	71	1.6 MB
	15.0ns	132.0	34.0s	74	1.5 MB
	12.0ns	167.3	45.9s	79	1.5 MB
	10.0ns	198.6	60.7s	89	1.6 MB
	8.0ns	247.1	101.6s	115	1.7 MB
	7.0ns	459.6	160.3s	143	1.8 MB
Add8	100.0ns	414.6	18.2m	147	4.4 MB
	80.0ns	491.1	12.3m	179	5.7 MB
	60.0ns	692.9	11.4m	236	5.7 MB
	40.0ns	1430.3	41.7m	271	6.0 MB
Add32	350.0ns	1909.5	420.9m	595	11.1 MB
	250.0ns	2866.5	456.9m	545	11.1 MB
	200.0ns	4329.6	543.5m	538	11.2 MB
Seq	20.0ns	498.5	169.6s	86 †	3.0 MB
	15.0ns	633.9	258.7s	89 †	3.1 MB
	10.0ns	1125.8	429.4s	105 †	3.2 MB

(† largest number of iterations required by a combinational subcircuit)

In a comparison with the optimization algorithm of TILOS [3, 6, 23], using the same delay models for both algorithms, it was found that when the delay specification was loose, the area of the TILOS-sized circuit was close (within a few tenths of a percent) to the optimal one obtained using the iCONTRAST algorithm. However, as the delay specification was made tighter, it was observed that the TILOS solution moved away from the optimal one; in some cases, the area achieved by iCONTRAST was under 1/3 that given by TILOS [23]. In comparison with TILOS, both the CPU time and the memory requirements were found to be larger; however, the improvement in the quality of the solution provided by iCONTRAST could be considerable, since the global optimum is guaranteed by this algorithm.

Fig. 5 shows the variation of transistor sizes in a 7-stage inverter chain. The minimum transistor size allowed here is 1.8 μm. The load that is driven by the chain corresponds to an inverter of $W_p/W_n = 50$ μm/50 μm. This problem has exactly two paths between the primary inputs and the primary output; the delay along both paths, i.e., the rise and the fall delays at the output node, are equal after sizing, as expected. For relatively loose delay specifications, it is seen that only the last stages are made larger, while those towards the input remain relatively unaffected. As T_{spec} (given in ns) is made tighter, it is seen that in addition to affecting the transistors at the output stages, the sizes of the transistors that are closer to the input are also significantly increased. The sizes of transistors in the input stage are restricted by the contribution of the user-specified resistance of the source that drives the first stage. The variation of sizes in the n-transistor stages is illustrated in Fig. 5(a); the variation of p-transistor sizes, shown in Fig. 5(b), follows the same trend as the n-transistor stages.

It should be noted that in this circuit, since the number of n-transistors (p-transistors) in the two paths is not equal, the nature of the variation in transistor sizes is somewhat different from a circuit such as an 8-inverter chain, which has equal numbers of n-transistors (p-transistors) on each path. To illustrate this, note that the path with the larger unsized delay goes through p-transistors 1, 3, 5 and 7. Hence, in the sized circuit, where both path delays are equal, it is seen that p-transistors 3 and 5 contribute to the disruption of the smoothness of the curve by being larger than their interpolated values. Transistors 1 and 7, being at the primary input and primary output respectively, are influenced by other considerations (namely, the input resistance and the output capacitance, respectively), and therefore, such effects are not visible.

However, a caveat is in order here: the above considerations are not the only reason for nonsmoothness of the curve; the curve for an 8-inverter chain is seen to be nonsmooth too. Also, one should curb an instinctive tendency to compare these variations with the smooth exponential variations of Mead and Conway [24], since the two problems are not the same. The Mead-Conway problem principally differs from ours in the following respects:
(a) The objective of their problem is to minimize the number of stages and the circuit delay. In our problem, the circuit topology, and hence, the number of stages is fixed.
(b) The Mead-Conway approach uses a simpler delay model.

Finally, the performance of iCONTRAST's delay estimator on a chain of 8 inverters (Inv8), a complete binary tree of seven 2-input NAND gates (Tree), and a 2-bit adder using complex gates (Add2), in relation with SPICE delay values, is shown in Figs. 6, 7 and 8, respectively. To illustrate the improvement provided by the enhancements of our algorithm, a comparison is provided with the delay obtained by a simple summation of component Elmore delays, without considering input slopes, and without considering the effects of capacitances that do not lie on the LRP. The various SPICE delay values correspond to a different set of transistor sizes in the circuit. For the circuits Inv8 and Tree, where the individual gates are inverters and NAND gates, respectively, the values are in excellent agreement. For a circuit like Add2 that is composed of complex gates, the accuracy can be seen to deteriorate slightly, but remains close to SPICE. It is clear from the data displayed here that the enhancements in our algorithm provide a considerable improvement.

VI. CONCLUSION

In this paper, we have presented a convex programming approach to solving the transistor sizing problem. This approach is guaranteed to find the global minimum solution to the problem. Any of the commonly-specified forms of the transistor sizing problem can be handled by this approach; we have illustrated the algorithm on the most useful form, given in Eq. (1). A major advantage is that the delay constraints do not need to be explicitly stated. Ensuring that the delay of the circuit satisfies the specification, is equivalent to ensuring that the delay along each path of the circuit satisfies the specification; since the number of paths in the circuit could be exponentially large, the number of constraints could be exponential in number. A conventional technique, such as Lagrangian multipliers, would not be able to solve a problem with such a large number of constraints in a reasonable time. The complexity of the algorithm is dependent on the number of variables, the size of the initial polytope, and the termination criterion, and is independent of the number of convex constraints. Moreover, the discontinuities in the circuit delay function do not require special treatment from the algorithm, as in many other transistor sizing algorithms such as [7].

A new delay estimation algorithm, that takes waveform slopes into account and calculates the worst-case delays, is also presented. Experimental comparisons with SPICE show that the enhancements made by this approach over previous approaches afford a large improvement in the quality of the solution.

The algorithm was implemented as a C program on a SUN Sparcstation I, and results on purely combinational circuits with up to 832 transistors, and on a sequential circuit, have been presented here.

ACKNOWLEDGEMENTS

The authors would like to thank the reviewers for their numerous helpful comments, which provided us with some further insight, and helped us to better organize and present the material. Our discussions with Dr. J. P. Fishburn and Dr. A. E. Dunlop of AT&T Bell Laboratories have been very helpful for the development of iCONTRAST, and its comparison with TILOS.

References

[1] J. G. Ecker, "Geometric programming : Methods, computations and applications," *SIAM Review*, vol. 22, pp. 338–362, July 1980.

[2] J. Rubenstein, P. Penfield, and M. A. Horowitz, "Signal delay in RC tree networks," *IEEE Transactions on Computer-Aided Design*, vol. CAD-2, pp. 202–211, July 1983.

[3] J. P. Fishburn and A. E. Dunlop, "TILOS : A posynomial programming approach to transistor sizing," in *Proceedings of the 1985 International Conference on Computer-Aided Design, Santa Clara, CA*, pp. 326–328, Nov. 1985.

[4] N. Hedenstierna and K. O. Jeppson, "CMOS circuit speed and buffer optimization," *IEEE Transactions on Computer-Aided Design*, vol. CAD-6, pp. 270–281, Mar. 1987.

[5] B. Hoppe, G. Neuendorf, D. Schmitt-Landsiedel, and W. Specks, "Optimization of high-speed CMOS logic circuits with analytical models for signal delay, chip area, and dynamic power dissipation," *IEEE Transactions on Computer-Aided Design*, vol. 9, pp. 236–247, Mar. 1990.

[6] A. E. Dunlop, J. P. Fishburn, D. D. Hill, and D. D. Shugard, "Experiments using automatic physical design techniques for optimizing circuit performance," in *Proceedings of the 32nd Midwest Symposium on Circuits and Systems, Urbana, IL*, Aug. 1989.

[7] J. Shyu, J. P. Fishburn, A. E. Dunlop, and A. L. Sangiovanni-Vincentelli, "Optimization-based transistor sizing," *IEEE Journal of Solid-State Circuits*, pp. 400–409, Apr. 1988.

[8] S. S. Sapatnekar and V. B. Rao, "iDEAS : A delay estimator and transistor sizing tool for CMOS circuits," in *Proceedings of the 1990 Custom Integrated Circuits Conference, Boston, MA*, pp. 9.3.1–9.3.4, May 1990.

[9] M. A. Cirit, "Transistor sizing in CMOS circuits," in *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pp. 121–124, June 1987.

[10] K. S. Hedlund, "AESOP : A tool for automated transistor sizing," in *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pp. 114–120, June 1987.

[11] D. P. Marple, "Performance optimization of digital VLSI circuits," Tech. Rep. CSL-TR-86-308, Stanford University, Oct. 1986.

[12] D. Marple, "Transistor size optimization in the Tailor layout system," in *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pp. 43–48, June 1989.

[13] Z. Dai and K. Asada, "MOSIZ : A two-step transistor sizing algorithm based on optimal timing assignment method for multi-stage complex gates," in *Proceedings of the 1989 Custom Integrated Circuits Conference, San Diego, CA*, pp. 17.3.1–17.3.4, May 1989.

[14] L. S. Heusler and W. Fichtner, "Transistor sizing for large combinational digital CMOS circuits," *Integration, the VLSI journal*, vol. 10, pp. 155–168, Jan. 1991.

[15] H. Y. Chen and S. M. Kang, "iCOACH: A circuit optimization aid for CMOS high-performance circuits," *Integration, the VLSI Journal*, vol. 10, pp. 185–212, Jan. 1991.

[16] P. M. Vaidya, "A new algorithm for minimizing convex functions over convex sets," *Proc. IEEE Foundations of Computer Science*, pp. 332–337, Oct. 1989.

[17] T. I. Kirkpatrick and N. R. Clark, "Pert as an aid to logic design," in *IBM Journal of Research and Development*, vol. 10, pp. 135–141, Mar. 1966.

[18] S. Even, *Graph Algorithms.* Computer Science Press, 1979.

[19] D. G. Luenberger, *Linear and Nonlinear Programming.* Addison-Wesley, 1984.

[20] G. H. Golub and F. H. V. Loan, *Matrix Computations.* The Johns Hopkins University Press, 1989.

[21] P. M. Vaidya, "An algorithm for linear programming which requires $O(((m+n)n^2 + (m+n)^{1.5}n)l)$ arithmetic operations," *Mathematical Programming*, vol. 47, pp. 175–201, 1990.

[22] A. V. Aho, J. E. Hopcroft, and J. D. Ullmann, *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, 1974.

[23] J. Fishburn, "Private communication," 1992.

[24] C. Mead and L. Conway, *Introduction to VLSI Systems.* Addison-Wesley, 1980.