Fast Comparisons of Circuit Implementations

Shrirang K. Karandikar and Sachin S. Sapatnekar, *Fellow, IEEE*

***Abstract*—Digital designs can be mapped to different implementations using diverse approaches, with varying cost criteria. Post-processing transforms, such as transistor sizing, can significantly improve circuit performance by optimizing critical paths to meet timing specifications. However, most transistor sizing tools have high execution times, and the possible delay gains due to sizing, and the associated costs are not known prior to sizing. In this paper, we present two metrics for comparing different implementations—the minimum achievable delay and the cost of achieving a target delay—and show how these can be estimated without running a sizing tool. Using these *fast* and *accurate* performance estimators, a designer can determine the tradeoffs between multiple functionally identical implementations, and size only the selected implementation.**

***Index Terms*—Circuit optimization, cost-delay tradeoffs, delay estimation, dynamic programming, gate sizing, logical effort, performance estimation, transistor sizing.**

I. Introduction and Motivation

IMPLEMENTING a design involves synthesis (technology independent optimizations and technology mapping), placement, and routing. In a final timing correction step, transistors in logic gates are appropriately sized to speed up critical paths, thus incurring a cost (which may be area, or power) overhead for gains in circuit speed. Although recent approaches have tried to combine sizing with technology mapping [1], [2], exact wire loads are determined only after placement and routing, and it is difficult to estimate them accurately at the technology mapping stage. Therefore, gate size selection is still performed heuristically, which leaves a large scope for improving the circuit delay at later stages by sizing. The importance of this step can be judged by the amount of research carried out both in academia [3]–[6] and in industry [7], [8]. A major drawback of these optimization tools is their large running times; it can take up to a few hours to calculate the appropriate solution for an industry-sized circuit. In this scenario, it is difficult for a designer to determine if an implementation will be able to meet performance goals after transistor sizing, or which circuit out of multiple different implementations for the same functionality should be chosen for further detailed optimization.

In this paper, we present an approach that estimates the benefits of sizing, but without incurring the overhead of running a sizing tool. This directly addresses the problem stated previously, since a designer can use our approach to compare a large number of implementations. We evaluate implementations based on two metrics; each of which is useful in different contexts. First, we consider the problem of estimating the minimum delay that can be achieved by an implementation, if sizing is applied to it. This metric allows a designer to determine if an implementation can meet a given delay specification. The *delay* of a circuit is the maximum delay of all PI-to-PO paths of the circuit. In order to meet design goals, transistor sizing is applied to the circuit to reduce this delay. The smallest value of delay that can be obtained in this manner is referred to as the *minimum achievable delay*. Most circuits are rarely sized in order to meet this minimum delay value, due to the associated high area overheads. However, those that are on the critical path may be. Additionally, the minimum achievable delay, along with the unsized circuit delay, also helps determine the range of delay values over which an implementation can be used.

In this work, we assume that the input to the sizing tool is a circuit that has been placed and routed, with all device sizes set to the minimum available value. During synthesis, device sizes may have been selected, but in the absence of physical information, these sizes are sub-optimal, and may take the design to a

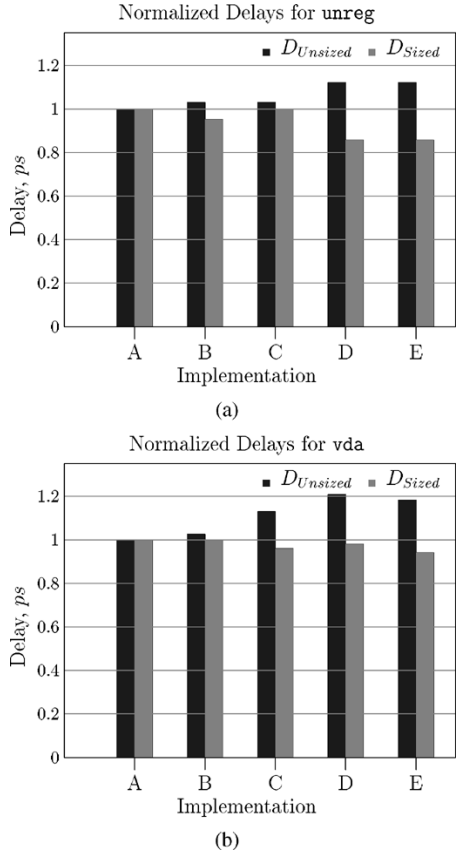


Fig. 1. Comparing unsized and sized delays of implementations.

Manuscript received April 21, 2005; revised August 5, 2005. This work was supported in part by the National Science Foundation under Award CCR-0205227 and Award CCR-0098117, and by the SRC under Grant 2001-TJ-884.

S. K. Karandikar is with the IBM Austin Research Laboratory, Austin, TX 78727 USA (e-mail: akkarand@us.ibm.com).

S. S. Sapatnekar is with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA.

Digital Object Identifier 10.1109/TVLSI.2005.862727

1063-8210/$20.00 © 2005 IEEE

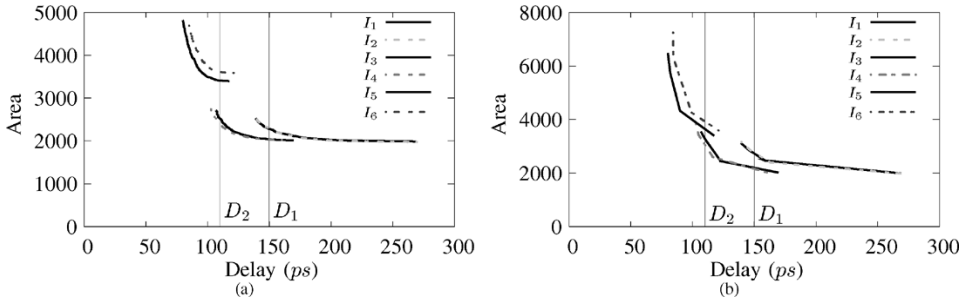


Fig. 2. Area-delay curves of six implementations of benchmark circuit C7552. (a) Actual.(b) Estimated.

state that is far from best. Rather than taking such an arbitrarily sized design, we reset all sizes to the minimum so that all implementations have a similar initial state.

It may seem that the delay of such an unsized circuit can be used as an approximation for minimum achievable delay. However, this is not the case, as can be seen from the situation shown in Fig. 1. This figure shows the normalized delays, before and after sizing for different implementations of two benchmark circuits, unreg and vda. In terms of the unsized delays, implementation A of circuit unreg is the fastest, implementations B and C are a few percent slower, while D and E are about 10% slower. However, once these implementations have been sized, we see that implementations D and E are actually the fastest. A similar situation is seen for circuit vda. Sizing all implementations is impractical, and since the unsized delay cannot be used, our estimator can be a useful tool to determine the best implementation.

The second aspect of making comparisons between different implementations is to determine, for a certain target delay, which implementation will have the least cost overhead after sizing. For convenience, we use the area of the implementation as a measure of the cost by which implementations can be selected. There is a direct correlation of area with other measures of cost, such as power dissipation, sub-threshold leakage and gate leakage, and a similar approach can be used when the cost function is power, or a weighted combination of area and power. This metric is applicable to circuits whose target delay of operation is greater than their minimum achievable delay, and hence need not be sized to the minimum delay value. Rather, the focus for these circuits is to minimize the cost while achieving the target delay. Currently, determining the cost is possible only after sizing has been performed, and as before, evaluating a large number of implementations is infeasible, due to the running time of current sizing tools. Simply using the delay and area of an unsized circuit can be misleading, since different implementations are superior at different delay points. This happens because the shape of the area-delay tradeoff curve can vary by implementation. Consider Fig. 2(a), which shows the area-delay curves of multiple implementations of benchmark circuit C7552, with the area of each implementation shown on the y-axis, and delay on the x-axis. The extreme right point of each curve corresponds to the unsized circuit; this has maximum delay and the smallest area, and successively smaller delay values require larger areas. Note that the curves have a characteristic point (called the 'knee'), at which the rate of change of area with respect to delay changes drastically.

Each curve is bounded by the maximum delay (i.e., the unsized circuit delay) and the minimum achievable delay[1]. However, as can be seen, the *shape* of each curve can vary significantly. For example, in the curves shown in Fig. 2(a), the knee of each curve can either be closer to one of the end points or in the center. This property varies between different circuits, as can be expected, but it also varies between implementations of the same circuit. For implementations I_1 and I_2 of C7552, the knee is closer to the minimum delay point. Hence, we initially observe large improvements in delay for relatively small area cost, for these implementations, but further delay improvement comes at the cost of large increases in area. The situation is reversed for implementations I_5 and I_6, where the knee is closer to the maximum delay point. In this scenario, trying to determine which implementation is the best at some intermediate delay point without having knowledge of the entire area-delay curve is difficult.

Suppose a designer wants to determine the best implementation among those available for some target delay of D_1. Calculating the minimum achievable delay and the unsized circuit delay of all implementations, the designer can determine that all implementations meet this target delay (I_5 and I_6 do so trivially, since their unsized delay is greater than D_1). At a different target delay of D_2, the implementations that have to be considered are I_3, I_4, I_5 and I_6. Implementations I_1 and I_2 need not be considered, since their minimum achievable delay is larger than this value. However, this information is not sufficient, since *which* of these circuits should be selected is still not known. Ideally, s/he would like an ordering of these implementations based on the cost, which in this case, is the area. The required ordering for a delay of D_1 is $\{I_3,\ I_4,\ I_2,\ I_1,\ I_5,\ I_6\}$, and for D_2 it is $\{I_4,\ I_3,\ I_5,\ I_6\}$. Simply ranking implementations based on the unsized delays and areas is not enough, e.g., at one delay point, I_4 has lower area, and at the other I_3 is better. This situation, of different implementations being the best at different delay points, is also seen in implementations of other benchmark circuits. As a drastic example, consider Fig. 3, which shows the area-delay curves of two implementations of benchmark circuit 9symml. Depending on the target delay selected, D_1 or D_2, either implementation I_1 or I_2 will be preferred. Without estimating these area-delay curves, which implementation is more area-efficient can be determined only by generating the area-delay curves.

[1]Technically, the maximum delay of an implementation can be increased by increasing the sizes of loads on the critical path. However, these circuits are clearly suboptimal and are, therefore, not considered.

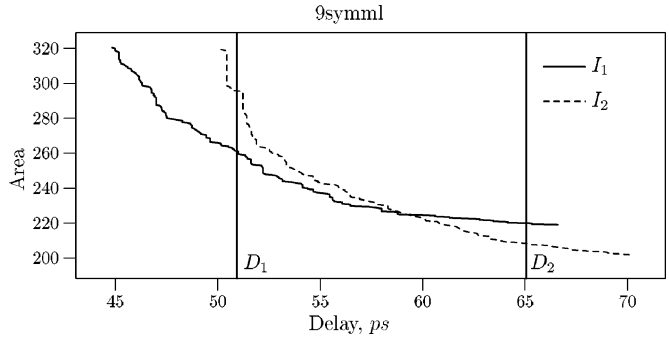


Fig. 3. Two implementations of benchmark circuit 9symml.

To summarize, in this paper we present algorithms that provide an estimate of the minimum achievable delay of a given implementation, and an estimate of the complete area-delay trade-off curve. The algorithms do not run a sizing tool, and are, therefore, *fast*, but at the same time, they enable *accurate* comparisons between different implementations, as we will show in the results section. Our approach is based on the method of logical effort [9], [10], which is well suited for estimating the minimum achievable delay of a *single* path in a circuit, with a heuristic branching factor used to account for multiple fanouts. However, the critical path of a circuit changes dynamically according to the choice of distribution of capacitance over multiple fanouts. An important contribution and differentiator of our algorithm is a means of accurately determining the minimum achievable delay of a circuit by simultaneously considering *all* paths of the circuit. Logical Effort, and its associated drawbacks are described in the following section. We then show how the drawbacks of logical effort can be overcome, in particular, how multiple fanouts are handled in our approach. This is integrated into two algorithms, the first for estimating the minimum achievable delay, and the second for estimating the area-delay curve of an implementation. This work has been published in preliminary form in [11] and [12].

II. Logical Effort

The starting point of our approach is the method of logical effort, which has been widely used in a variety of application domains [1], [13]–[15] as well as in industry standard EDA synthesis tools [16], [17]. Using logical effort, the delay of a gate with is estimated by modeling it as a linear function of the load being driven as

$$D = g \times \frac{c_l}{c_i} + p = g \times h + p = f + p \tag{1}$$

where:

- **Logical Effort** (g) is the complexity of the gate, relative to an inverter. It measures how much worse the gate is at driving a specified load than an inverter. The base case of an inverter is taken to have unit logical effort, and complex gates such as NAND, NOR, and XOR have successively higher values of logical effort.
- **Electrical Effort, or Gain** ($h = c_l/c_i$) describes how the electrical environment of the logic gate affects performance and how the size of the transistors in the gate determines its load-driving capability. c_l is the load being driven and c_i is the input capacitance of the gate under consideration.
- **Effort Delay** ($f = gh$) is the product of the logical and the electrical efforts of the gate.
- **Parasitic Delay** (p) expresses the intrinsic delay of the gate due to its own internal capacitance, and is largely independent of the size of the transistors in the logic gate.

This formulation separates the different components that contribute to the delay of a gate. It also provides the user with a means of sizing the gate—since the logical effort g of a gate is fixed, if a particular effort delay is assigned to a gate, the input capacitance c_i that meets this effort delay can be calculated as

$$c_i = \frac{g \times c_l}{f}. \tag{2}$$

As shown in [10], (1) can be extended to estimate the minimum delay $\hat{D}$ of a *path* of logic as

$$\hat{D} = NF^{1/N} + P \tag{3}$$

where $F = GH$ is the path effort, P is the path parasitic delay, and N is the number of gates on the path under consideration. The path logical effort, G, is the product of the logical efforts of the gates on the path. The path electrical effort H is obtained as the product of the gate electrical efforts, or equivalently, by the ratio of the load being driven by the last gate C_L and the input capacitance of the first gate. The minimum delay of (3) is obtained by distributing the path effort F equally to each gate on the path. Thus, each gate is assigned a gate effort of $f = F^{1/N}$. Starting with the gate at the output, that drives a fixed load of C_L, the size of each gate can be successively determined by using (2).

Equation (3) can be used for determining the minimum delay (and the corresponding gate sizes) of a *simple* path of logic, in which each gate only drives the next gate on the path. However, realistic circuits have gates that drive multiple fanouts. In order to address this situation, [10] introduces the concept of a *branching effort*, $b = C_{\text{total}}/C_{\text{useful}}$, where C_{total} is the total load being driven by the gate, and C_{useful} is the load contributed by the fanout on the path of interest. The gate effort f is now defined to be $f = gbh$. In Fig. 4, gate X drives two gates, Y and Z, which have input capacitances c_{in_Y} and c_{in_Z}, respectively. The total load being driven by gate X is then $C_{\text{total}} = c_{in_Y} + c_{in_Z}$, as shown, and C_{useful} is either c_{in_Y} or c_{in_Z}, depending on whether path P1 or path P2 is being analyzed.

The path effort in (3) is modified to $F = GBH$, where B, the path branching effort, is the product of the gate branching efforts of all gates on the path being analyzed. A similar methodology is followed in order to obtain the minimum delay, and the corresponding gate sizes, i.e., each gate on the path is assigned an effort of $F^{1/N}$, and (2), used to calculate the gate sizes, is modified to include the effect of the branching factor as

$$c_i = \frac{g \times c_l \times b}{f}. \tag{4}$$

In this manner, the branching factor tries to capture the effect of fanouts that are not on the path of interest. This approach, however, has a few serious flaws. First, paths are analyzed individually, and the interactions between the sizes required by each path are not taken into account. More importantly, the branching

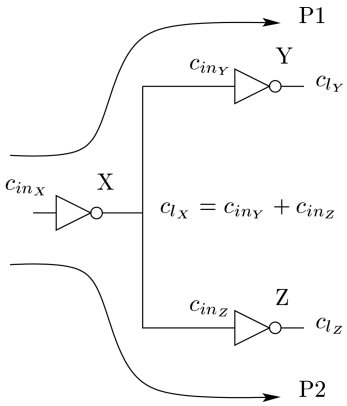


Fig. 4. Multiple fanouts in analyzing circuit delay.

factor is assumed to be fixed, and it is calculated using the initial values of the fanout capacitances (in the example presented in [10], all fanouts are shown to have the same size, both before and after sizing). Hence, when a path is sized using (4), the gate sizes of fanouts not on the path under consideration have to be scaled according to the branching factor initially selected. For example, in Fig. 4, when path P1 is being analyzed, the branching factor of gate X is $b_X = c_{o_X}/c_{in_Y}$. Once the path effort of P1, F_{P1}, has been calculated, and used to size gates X and Y, the size of gate Z will have to be scaled by an appropriate amount, in order to keep the value of b_X constant. If path P2 is not critical, increasing the size of Z is unnecessary, and only increases the load on gate X. If path P2 is analyzed separately, its path effort, F_{P2}, may require completely different sizes of gates X and Z, and gate Y will have to be scaled according to the branching factor $b_X = c_{o_X}/c_{in_Z}$. Thus, in the case of multiple fanouts, the optimal sizes of each fanout (gates Y and Z in Fig. 4), and the corresponding size of the gate driving the fanout (gate X in Fig. 4) cannot be easily determined using the branching factor. Analyzing every path in a circuit separately, and the interactions among all paths is not feasible, because of the exponential number of such paths in a circuit. Thus, while the method of logical effort is well suited to analyze single path delays, it cannot be used directly when critical paths are not well defined, or can change. In the following section, we present an approach that can handle such scenarios.

III. Delay Calculation Integrating Gate Sizing

The advantage of logical effort is that it provides the user with a means of determining the delay of a path of logic while simultaneously determining the gate sizes required for achieving that delay. In this section, we present an approach that is equivalent to logical effort in the degenerate case (single fanouts and no routing capacitances being considered). Even in the degenerate case, our approach has higher accuracy, since only discrete gate sizes that are available in the technology library are used in our calculations. As mentioned in the previous section, logical effort has severe shortcomings when circuits with multiple fanouts are being analyzed. Similar drawbacks exist when routing capacitance is taken into consideration. Our approach overcomes these drawbacks by simultaneously considering gate sizes of *all* paths

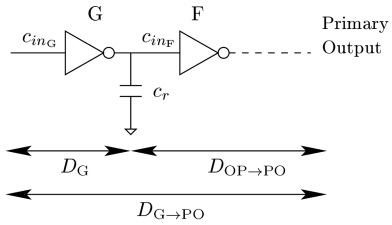


Fig. 5. Components of the delay of a simple path.

in the circuit. A key concept of our approach is calculating and propagating Delay-C_{in} curves for all gates, which capture the effect of changing delay for different gate sizes. We first present this approach for simple paths, including the effects of routing capacitance. This is then extended to handle multiple fanouts.

A. Simple Paths

We define $\mathcal{C}_{in_G}$ to be the set of all possible values of input capacitance, c_{in_G}, of gate G, corresponding to different sizes of G. Consider the situation shown in Fig. 5, where gate[2] G drives fanout gate F. The input capacitance of G and F are c_{in_G} and c_{in_F} respectively. The interconnect between the output of G and the input of F has a routing capacitance of c_r. Thus, the load capacitance that G drives is the sum of the input capacitance of F and the routing capacitance, or

$$c_{l_G} = c_r + c_{in_F}, \quad c_{in_F} \in \mathcal{C}_{in_F} \tag{5}$$

Hence, if F has k different sizes, G will have k corresponding values of load capacitance.

Now consider the delay from the input of a gate G to any primary output. This delay has two components, the delay of G itself, D_G, and the delay from the output of G to a primary output, $D_{OP \to PO}$. Thus,

$$D_{G \to PO} = D_G + D_{OP \to PO}. \tag{6}$$

This equation is incomplete, since it does not take into account the sizes of G or its outputs. These can be incorporated as follows. The term on the left hand side of (6) depends on the size of G that is under consideration, and is therefore correctly represented by $D_{G \to PO}[c_{in_G}]$. We know that the delay of G depends on its load, c_{l_G}, as well as its size, c_{in_G}, and is given by (1). Hence, we represent the gate delay as $D_G[c_{in_G}][c_{l_G}]$. An improved version of (6) is, therefore

$$D_{G \to PO}[c_{in_G}] = D_G[c_{in_G}][c_{l_G}] + D_{OP \to PO}. \tag{7}$$

The last term corresponds to the delay from the input of F to a primary output. Hence, (7) can be rewritten as

$$D_{G \to PO}[c_{in_G}] = D_G[c_{in_G}][c_{l_G}] + D_{F \to PO}[c_{in_F}]. \tag{8}$$

For a simple path of logic, (8) is a recursive definition of delay. For a gate driving a primary output (and hence driving a fixed load), the delay to the primary output is simply the delay

[2]In this discussion, all gates are shown as inverters for illustration purposes only. The method applies directly to more complex gates, with appropriate values for logical effort and parasitic delay.

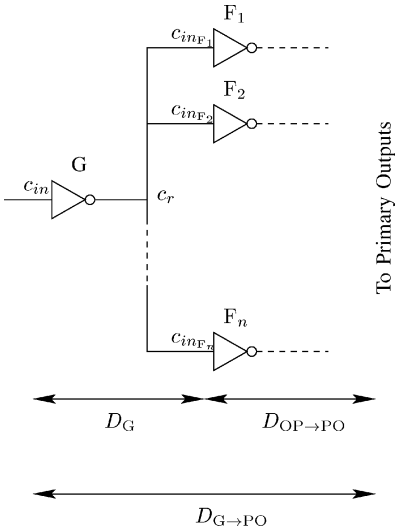


Fig. 6. Components of the delay of a path.

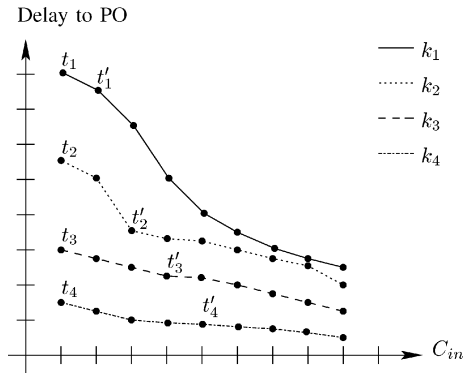


Fig. 7. Combining Delay-C_{in} curves at multiple fanouts.

of the gate itself. The delay for any other gate is defined in terms of the delay from its output to the primary output.

Recall that the value of c_{l_G} also depends on the input capacitance of gate F, c_{in_F}, by (5). As mentioned before, gate F may have k different sizes available. Hence, there are a corresponding number of k different values of $D_{G \to PO}[c_{in_G}]$. We are interested in the minimum delay from the input of gate G to a primary output. However, the effect of different values of c_{in_F} on each component of (8) is opposite: as c_{in_F} increases, so does the delay of gate G, but the delay to primary output of gate F reduces. Thus, in order to obtain the minimum delay from the input of gate G to a primary output for a selected value of input capacitance of G, we need to examine all values of c_{in_F}.

$$D_{G \to PO}[c_{in_G}] = \min_{c_{in_F} \in \mathcal{C}_{in_F}} \{D_G[c_{in_G}][c_{l_G}] + D_{F \to PO}[c_{in_F}]\}. \tag{9}$$

The value of $D_{G \to PO}[c_{in_G}]$ as defined in (9), for different values of c_{in_G} constitutes the Delay-C_{in} curve of gate G. It captures the minimum delay from the input of G to the primary output, for different sizes of G. Note that the sizes of all gates on the path from G to the primary output are implicit in the Delay-C_{in} curve. The formulation of (9) leads directly to a dynamic programming based algorithm, presented in the following section.

B. Multiple Fanouts

In the case that G drives multiple fanouts, the load capacitance c_{l_G}, is calculated as follows. Say G has n fanouts, $F_1, F_2, \ldots F_n$, as shown in Fig. 6. We denote the possible values of the sum of the input capacitances of the fanouts by the set $\mathcal{C}_{L_G}$. Then the load capacitance of G is the input capacitances of these fanouts (an element from $\mathcal{C}_{L_G}$), combined with the routing capacitance, c_r.

$$c_{l_G} = c_r + c_m$$
$$c_m \in \mathcal{C}_{L_G} = \left\{ \sum_{j=1}^{n} c_j : \forall c_j \in \mathcal{C}_{in_{F_j}}, j = 1 \cdots n \right\}. \tag{10}$$

If we assume that each of the fanouts have k sizes, then the number of values that we obtain, for c_{l_G} is k^n. However, we show later that the number of useful values to be considered is actually linear.

The delay calculation of (7) also changes in the case of multiple fanouts. In (7), since there was only one fanout, we could use its delay to a primary output in order to obtain (8). However, in the case of multiple fanouts, we are interested in the maximum delay from the input of gate G to *any* primary output. Thus, the correct value to use for $D_{OP \to PO}$ is the maximum delay to a primary output over all fanouts. Thus,

$$D_{G \to PO}[c_{in_G}] = D_G[c_{in_G}][c_{l_G}] + \max_{j=1 \cdots n} \{D_{F_j \to PO}[c_{in_{F_j}}]\}. \tag{11}$$

As before, we can have multiple values of $D_{G \to PO}[c_{in_G}]$, for different combinations of input capacitances of the fanout gates. Since we are interested in the minimum of these, we obtain

$$D_{G \to PO}[c_{in_G}] = \min_{c_m \in \mathcal{C}_{L_G}} \{D_G[c_{in_G}][c_{l_G}] + \max_{j=1 \cdots n} \{D_{F_j \to PO}[c_{in_{F_j}}]\}\}. \tag{12}$$

Note that a selection of sizes of the fanouts $F_1, F_2, \ldots, F_n$ (which determine the corresponding input capacitances $c_{in_{F_1}}, c_{in_{F_2}}, \ldots, c_{in_{F_n}}$, and therefore, c_m) also fix the value of the load that gate G has to drive, c_{l_G}, by (10).

It may seem that the size of the set $\mathcal{C}_{L_G}$ for a gate G with multiple fanouts is proportional to the product of the number of sizes of the fanout gates. Assume gate G drives four outputs, whose Delay-C_{in} curves are represented by k_1, k_2, k_3 and k_4, shown in Fig. 7. If each fanout has m sizes, each curve has m points, and the size of $\mathcal{C}_{L_G}$ is m^4. However, we can show that most of the values in $\mathcal{C}_{L_G}$ are redundant. For example, consider the tuple $\mathcal{T}$ of the first points t_1, t_2, t_3 and t_4 from each of the curves in Fig. 7. A tuple $\mathcal{T}'$ of the point t_1 from curve k_1 and any other point from k_2, k_3 and k_4 (say t'_2, t'_3 and t'_4), is *inferior* to $\mathcal{T}$ for the following reason. There are two values that are extracted from $\mathcal{T}$ and $\mathcal{T}'$, the maximum delay to a primary output, and the sum of the input capacitances represented by these combinations, which is used as the load in the delay calculation of gate G. The maximum delay is the same in tuples $\mathcal{T}$ and $\mathcal{T}'$, but the load presented by $\mathcal{T}'$ is greater than that of $\mathcal{T}$. Hence, the delay of G, and therefore its delay to a primary output is larger in this case. Since we are interested in minimizing the delay to

a primary output, the solution offered by tuple $\mathcal{T}'$ will never replace that calculated using $\mathcal{T}$.

The above discussion directly leads to a strategy for efficiently selecting useful values of c_l from the Delay-C_{in} curves of outputs. First, these curves are stored in order of nonincreasing delay (and hence increasing sizes). The first value of c_l is the routing capacitance c_r plus the capacitance corresponding to the maximum-delay points from each curve, as in tuple $\mathcal{T}$. The next value is obtained by replacing the point with maximum delay (e.g., t_1 of curve k_1 in $\mathcal{T}$), with the next point from the same curve (t_1'). This effectively ignores the combination of t_1 with remaining points from the other curves. This process is continued till the maximum delay point is the last point on its curve. Thus, in the worst case, the total number of combinations is of the order of the *sum* of number of points on each curve, rather than the product. This worst case occurs when the Delay-C_{in} curves of all outputs are identical. In other situations, the number of combinations is smaller than the sum of the number of points on each curve.

Recall the drawbacks of the traditional branching effort mentioned in the previous section. A fixed branching effort, without considering the interactions between fanout branches can lead to suboptimal circuits, and considering the interactions is impractical, due to the number of combinations involved. Using Delay-C_{in} curves allows us to efficiently assign sizes to multiple fanouts according to the criticality of each branch.

IV. Algorithms for Estimating the Benefits of Sizing

Based on the formulation of Delay-C_{in} curves presented in the previous section, we now present two algorithms that determine the metrics mentioned in Section I. Calculating the Delay-C_{in} curve of a circuit allows us to estimate its minimum achievable delay. Modifications to this approach can be used to determine the cost of sizing a circuit to a target delay, rather than sizing to the minimum achievable delay.

A. Minimum Delay Estimation

Algorithm 1 Minimum Delay Estimation
for each gate G whose outputs have been processed **do**
  // calculate the Delay-C_{in} curves for G
  for all $c_i \in \mathcal{C}_{in_{\mathrm{G}}}$ **do**
    $D_{\mathrm{G}\rightarrow\mathrm{PO}}[c_i] = \infty$
    // G has n fanouts, $\mathrm{F}_1, \mathrm{F}_2, \ldots \mathrm{F}_n$
    for $c_m \in \mathcal{C}_{L_{\mathrm{G}}}$ **do**
      $c_{l_{\mathrm{G}}} = c_r + c_m$
      // determine the delay of gate G
      $D_{\mathrm{G}}[c_i][c_{l_{\mathrm{G}}}] = [g \times (c_l/c_i) + \text{parasitic delay}]_{\mathrm{G}}$
      // the maximum delay from any fanout F
      // to any PO is obtained from the
      // Delay-C_{in} curves of F
      $\text{temp} = D_{\mathrm{G}}[c_i][c_{l_{\mathrm{G}}}] + \max_{j=1\ldots n}(D_{\mathrm{F}_j\rightarrow\mathrm{PO}}[c_j])$
      $D_{\mathrm{G}\rightarrow\mathrm{PO}}[c_i] = \min(\text{temp}, D_{\mathrm{G}\rightarrow\mathrm{PO}}[c_i])$
    end for
  end for
end for
$\text{Minimum Delay} = \max\{\min_{\text{all PI's}}\{\text{delay to PO}\}\}$

Our formulation for calculating the Delay-C_{in} curve of a gate has been presented in the previous section, the most general form being represented by (12). As mentioned before, this is a recursive definition, each value on the curve being defined in terms of the curves of the fanouts. Algorithm 1 exhibits the hallmarks of dynamic programming: the optimal solution for the current gate size is defined in terms of the optimal solutions of its outputs, which are calculated once and used as needed. A single traversal of the circuit, from primary outputs to primary inputs is sufficient to calculate the Delay-C_{in} curves of all gates in the circuit. Processing gates in such a topological manner ensures that when the Delay-C_{in} curve of a gate is being calculated, the Delay-C_{in} curves of all of its fanouts have already been determined.

We assume that primary inputs are inverters of fixed size. The Delay-C_{in} curves at the primary inputs include the delay of these inverters, so that the loading effects of the gates being driven by the primary inputs are taken into account. Once we have calculated the Delay-C_{in} curves at the primary inputs, determining the minimum achievable delay of the circuit is straightforward. For each primary input, we can calculate the minimum delay to any primary output, and the largest such value over all the primary inputs is the minimum achievable delay of the circuit. Algorithm 1 presents the complete algorithm, called Minimum_Delay_Estimation (MDE).

Run Time Analysis: Assume that there are k sizes for each gate in a circuit with N gates, and the maximum fanout on any gate is $|FO|$. The innermost for loop is executed $O(k \times |FO|)$ times, as shown previously, and the cost of determining the maximum delay point is $O(|FO|)$. The second for loop is executed k times, since we assume k sizes for each gate. Finally, since there are N gates in the circuit, the outermost for loop is executed N times. Thus, the running time of Algorithm MDE is $O(N \cdot k \cdot k \cdot |FO| \cdot |FO|)$. However, note that this is a very loose upper bound, since very few gates actually have $|FO|$ fanouts.

Algorithm MDE is optimal for trees. However, most circuits are DAG's, with reconvergent fanouts. The main problem with DAG's is that there are multiple paths from a particular gate to primary outputs, or between two gates. An implicit assumption of our algorithm is that the Delay-C_{in} curves at multiple fanout points are independent, and that we are free to choose the combination of output delays and capacitances that best suit the current gate. However, with reconvergent fanouts, these choices are not independent of each other. Selecting a data point on one output restricts the choices on the other, and determining the relation between different outputs is intractable for general circuits. However, assuming independence is not unreasonable, for an estimator such as ours. If the reconvergent paths are completely unbalanced, i.e., their structure and logic is such that one always has smaller delay than the other, no errors are introduced due to the manner in which their Delay-C_{in} curves are combined. The smallest C_{in} value will consistently be selected for the path with smaller delay. An example of this situation is if the paths correspond to curves k_1 and k_4 in Fig. 7. On the other hand, if the delays of the two paths are roughly of the same order (e.g., if they correspond to curves k_1 and k_2), our approach selects approximately similar values of input capacitances. This may lead to small inaccuracies, since the actual values of input capacitance may be slightly different. However, the error in delay estimation is limited, as shown by the results in Section V.

Our approach can also be used to obtain actual sizes of all gates in the circuit. In Algorithm MDE, we can store the value

of the load of each output that induces the minimum delay (corresponding to values of c_m and $c_{in_{\mathrm{F}_j}}$ in (12)). This information can be used in a forward traversal of the circuit, in order to generate sizes for every gate. A gate with multiple fanins has multiple choices for its size, which can be resolved by selecting the size imposed by the critical input[3]. The effect on the noncritical inputs is that they now have a load different from what was initially assumed. However, the difference in the delays from the primary inputs to the critical and noncritical inputs can be used to compensate for this. In fact, this difference can be usually be used to *reduce* the sizes of the transitive fanin cone of the noncritical inputs, as long as their delay does not become larger than that of the critical input. Gate sizes determined in this manner correspond to a circuit sized for minimum delay. These sizes can be used as an initial feasible solution for an exact sizing tool, instead of using the original unsized circuit. This can lead to a large improvement in running times of the transistor sizing tool, since a circuit sized using our approach is closer to the final solution than the initial, unsized circuit. Note however, that the area of a circuit sized in this manner cannot be used as an estimate for the area of the final circuit, due to the nature of the sizing problem. This issue is discussed in the following subsection.

B. Area-Delay Curve Estimation

As mentioned in Section I, not all circuits need to be sized to operate at the minimum achievable delay. For these circuits, we can trade area for delay, also achieving a reduction in power. In a scenario where a target delay is known, and multiple implementations of a given circuit are available, we would like to determine the cost (in terms of area) for achieving the given delay, which entails estimating the entire area-delay curve of each implementation. As before, determining the exact area-delay curve is expensive. In this subsection, we modify Algorithm 1 in order to quickly estimate the area-delay curve of a given implementation.

Since we are interested in determining the entire area-delay curve of the implementation, a natural approach would be to calculate the area of the transitive fanouts with the delays during the Delay-C_{in} calculation of Algorithm 1. However, there are a few problems with this approach. There are multiple configurations of gate sizes that can achieve the same delay value, and hence multiple solutions for each delay value have to be stored. Unlike in Algorithm MDE, these solutions cannot be pruned. Finally, every combination of points in the enhanced Delay-C_{in} curves of multiple fanouts has to be considered, which further increases the complexity.

We therefore need another approach to estimating the area-delay curve. Recall that the Delay-C_{in} curves calculated in Algorithm MDE implicitly store sizes of gates in the transitive fanout cone required for achieving the minimum delay for each value of C_{in}. Hence, we can size the circuit using different points on the Delay-C_{in} curves of the primary inputs, and calculate the corresponding area. However, these points may not be optimal i.e., the area calculated using the above approach may not be the smallest area for a particular delay. For example, say we have a minimum delay of d_1 for c_{in_1} and d_2 for c_{in_2}, with corresponding circuit areas of a_1 and a_2, and $d_1 > d_2$. It is possible that there is a nonminimum delay $d_2' = d_1$ for an input capacitance of C_{in_2} that had a corresponding circuit area a_2', that is less than a_1. The solution (a_2', d_2') (corresponding to an input capacitance of c_{in_2}) is clearly better than the (a_1, d_1) solution, but since only minimum delay points are considered, the superior solution is ignored.

[3]The critical input of a gate is the input with the largest arrival time. The other inputs are called noncritical inputs.

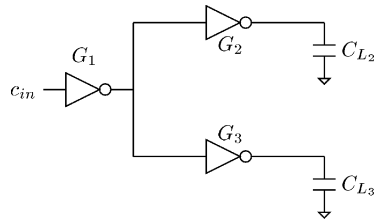


Fig. 8. Example circuit.

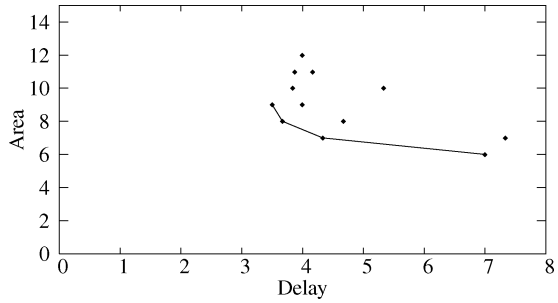


Fig. 9. The area-delay curve for one value of c_{in}.

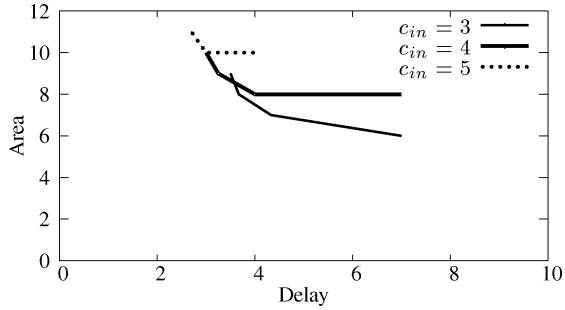


Fig. 10. Area-delay curve of the circuit in Fig. 8.

Consider the circuit shown in Fig. 8, with two branches of the circuit driving different loads. For some input capacitance of c_{in}, we obtain a number of delay values, the minimum of which is stored in the Delay-C_{in} curve, and the other delay values are discarded. However, we can size the circuit using the minimum as well as the discarded delay values (for the same value of c_{in}), and calculate the corresponding areas. These points are shown in Fig. 9, and the best points for an area-delay curve perspective are the ones marked by a line. This procedure can be repeated for other values of c_{in}, and the union of the solutions obtained gives us the area-delay curve desired. This is shown in Fig. 10 for three values of c_{in}. The intersection in the curves corresponding to $c_{in} = 4$ and $c_{in} = 5$ is an example of sub-optimality if only the minimum delay points were to be considered.

Thus, we estimate the area-delay curve of a circuit by sizing it for different values of delay, for every value of C_{in} and measuring the area. In order to keep the run time low, rather than sizing for all delay values, we size the circuit for a limited

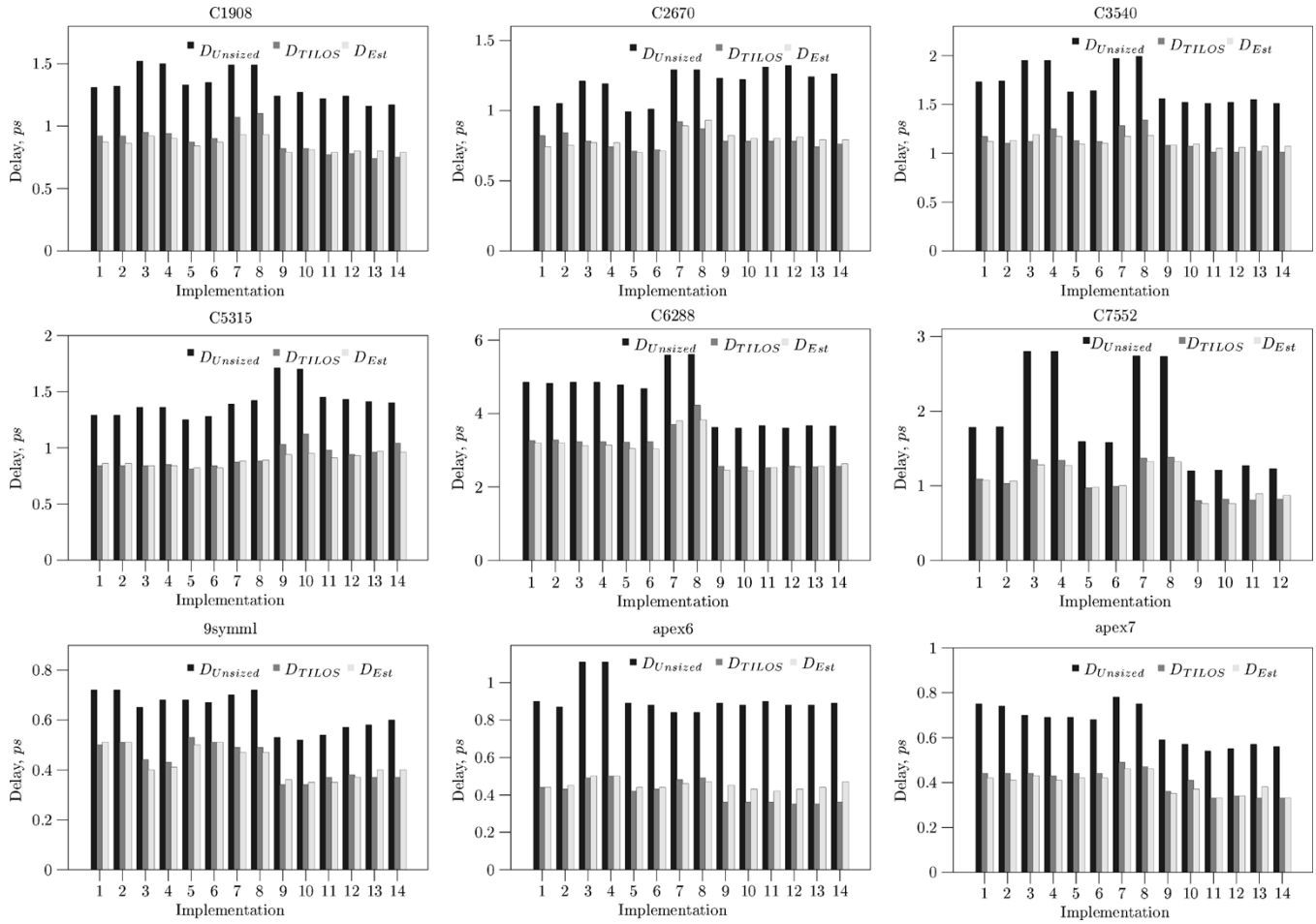


Fig. 11. Results for selected ISCAS and MCNC benchmark circuits.

number of values (in our experiments, we found that selecting 10 sub-optimal delay points was sufficient). This has an impact on the accuracy of our results, but the effect is limited.

Our heuristic, called Algorithm ADC is shown in Algorithm 2. At the primary inputs, we store multiple Delay-C_{in} curves. Each time $D_{\mathrm{G}\rightarrow\mathrm{PO}}[c_i]$ is updated to a new value, we store the replaced value as an entry in a set of secondary curves. The minimum delay values from these secondary curves are then used to size the circuit, and obtain other points on the delay-area curve. Circuits sized in this manner have greater delay than the minimum achievable delay, and after area recovery, they have smaller area as well.

Algorithm 2 Area-Delay Curve Estimation
for each gate G whose outputs have been processed **do**
  if G is not a PI **then**
    Calculate Delay-C_{in} curve of G as in Algorithm 1
  else
    Calculate Delay-C_{in} curve as before, but for each c_i store *all* solutions
  end if
end for

for each set of Delay-C_{in} curves of the PI's **do**
  Min. Delay $= \max\{\min_{\text{all PI's}}\{\text{delay to PO}\}\}$
  // forward traversal
  Size the circuit based on the selected point
  Determine the arrival time at each gate
  // reverse traversal
  Determine the required time at each gate
  // area recovery
  for each gate G in reverse topological order **do**
    $slack = \text{arrival time} - \text{required time}$
    while $slack > 0$ **do**
      Reduce the size of G
      Update the arrival and required times of G and its inputs
    end while
  end for
  Determine area and delay of the sized circuit
end for

The solution obtained using this approach is naturally not exact. However, as discussed above, since the auxiliary data of points on the secondary curve encode sizes of the outputs (and particularly, of sizes of multiple fanouts), these solutions still provide a good representation of the area behavior of the circuit at different delay points. That is, though we cannot use the area-delay curves to make absolute judgments, we can still make comparative judgments between different circuits.

Once the circuit has been sized, we determine the arrival and required times at each gate, and use the slack to reduce the sizes

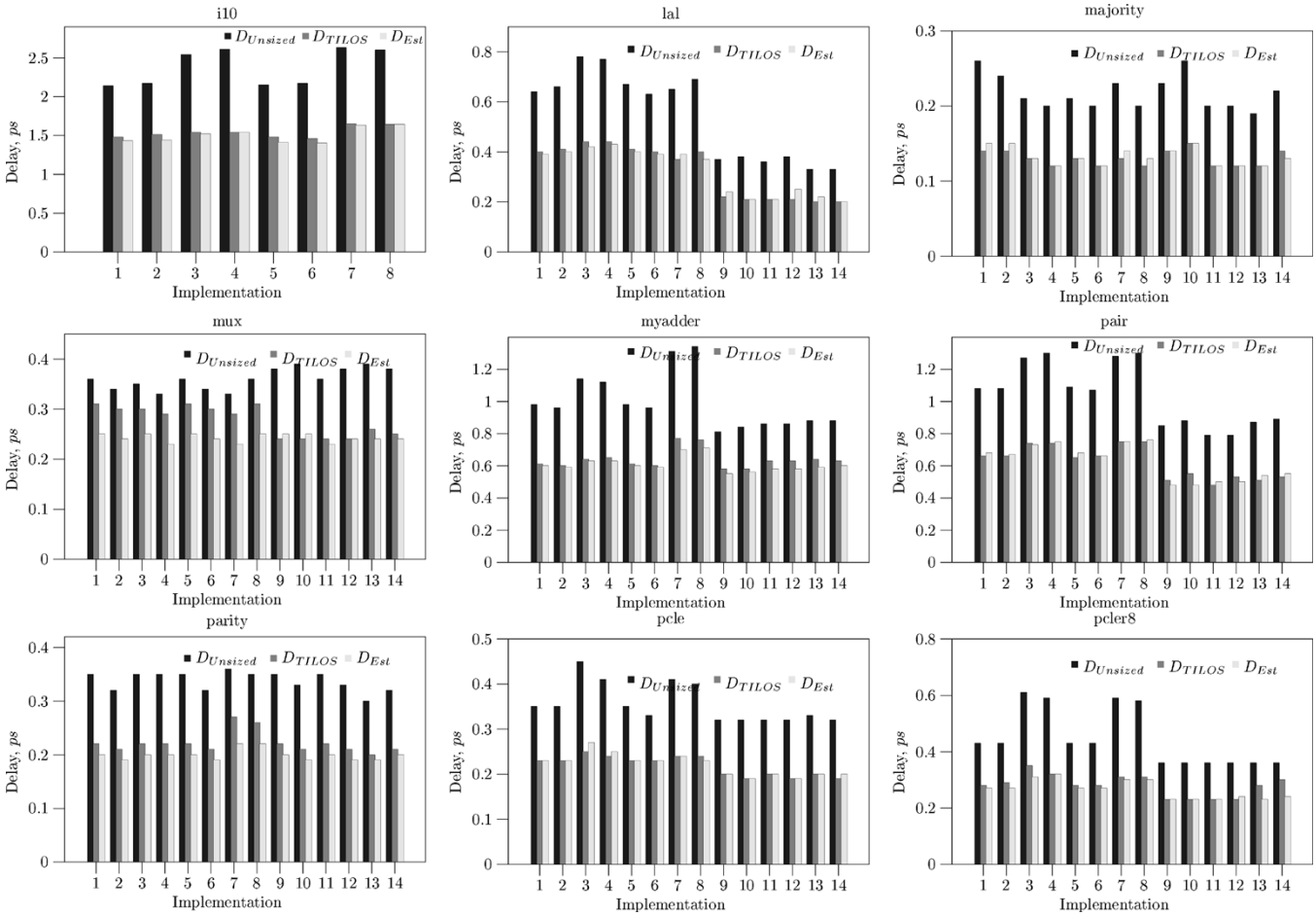


Fig. 12. Results for selected MCNC benchmark circuits.

of the gates. This step can drastically reduce the area of a circuit, since the noncritical parts of the circuit are usually sized to be unnecessarily fast. After the Delay-C_{in} curves have been calculated, the arrival and required times of each gate can be determined in two traversals of the circuit. This calculation is performed for each set of Delay-C_{in} curves available, and hence the running time is dominated by that of Algorithm 1.

V. Results

In the previous sections, we have presented two metrics for determining the benefits obtainable from sizing. In order to validate our approach to calculating these metrics, we proceed as follows. We use a library consisting of multiple sizes of an inverter and two-input NAND, NOR and XOR gates, at the 0.1μ technology node, characterized using the Berkeley Predictive Technology Model[4][18]. In all, we have 10 discrete sizes of each gate type. The calculated gate sizes are rounded off to the nearest size available in this library. We use SIS [19] to map ISCAS and MCNC benchmark circuits, with varying optimization criteria (for area, delay and combinations of area and delay), and obtain 7 implementations. We then add random capacitances at all interconnects of these implementations twice, in order to simulate the effect of different placement and routing solutions. We thus obtain 14 different implementations for each benchmark circuit. The library is also characterized in order to obtain correct values of logical effort and parasitic delays for each gate[5]. Comparisons are made with our implementation of TILOS [3].

[4]Available from http://www-device.eecs.berkeley.edu/~ptm

TABLE I
Percentage Error of MDE wrt TILOS

Circuit	Error%	Circuit	Error%	Circuit	Error%
C1908	5.50	C2670	4.53	C3540	4.79
C5315	3.76	C6288	3.42	C7552	4.45
9symml	4.47	apex6	11.59	apex7	4.52
b1	2.32	b9	5.10	c8	3.50
cc	3.64	cht	11.74	cm138a	8.86
cm150a	4.73	cm151a	3.10	cm152a	5.26
cm162a	2.05	cm163a	2.75	cm42	3.99
cm82aa	10.16	cmb	8.00	cordic	6.95
count	6.40	cu	2.42	dalu	3.86
decod	0.65	des	1.97	example2	13.53
f51m	4.41	frg1	6.18	frg2	6.61
i1	10.46	i2	9.17	i3	12.60
i10	2.42	lal	4.84	majority	2.68
mux	12.88	myadder	4.57	pair	3.66
parity	9.74	pcle	1.54	pcler8	5.55
pm1	8.17	rot	4.41	tcon	7.49
term1	6.99	unreg	4.06	vda	6.67
x1	6.64	x2	4.19	x3	7.45

A. Algorithm MDE

For all implementations of every benchmark circuit (obtained as described above), we apply our implementations of TILOS

[5][10] describes how these values can be obtained from the reference model. Our approach is detailed in [20].

TABLE II
COMPLETE AREA-DELAY CURVE COMPARISON.

Circuit	Comparisons		$\lvert\Delta A_{\text{est}} - \Delta A_{\text{act}}\rvert$			
			All Comparisons		Incorrect Comparisons	
	Total	Incorrect	maximum(%)	average (%)	maximum(%)	average (%)
C432	102	3	21.85	6.42	11.33	9.51
C499	158	14	21.52	6.38	16.08	7.83
C880	41	3	13.95	4.11	9.89	6.93
C1355	136	10	28.61	8.17	18.11	8.02
C1908	113	4	25.72	5.62	5.22	4.00
C2670	121	1	18.87	3.49	3.28	3.28
C3540	101	5	18.24	4.49	14.51	8.12
C5315	163	8	27.51	7.24	5.09	2.34
C7552	30	2	25.08	7.02	4.70	2.68
C6288	57	10	22.50	4.65	11.62	4.85
apex6	127	8	13.62	3.37	5.09	2.13
b9	108	6	9.47	3.00	8.57	4.73
cm162a	162	12	17.90	6.53	17.87	11.40
x3	83	14	16.64	4.88	14.94	6.84
z4ml	64	12	16.43	3.31	9.99	5.01
i7	74	4	8.53	2.47	4.22	1.60
9symml	104	6	20.82	4.82	13.18	9.16
pair	89	10	23.51	5.93	14.24	7.01
pcler8	62	5	16.26	2.86	3.31	2.31
k2	95	3	24.31	4.89	13.91	5.99
ttt2	103	6	11.53	3.14	9.27	6.25
lal	82	0	14.12	3.60	0.00	0.00
Total	2175	146(6.71%)				

and Algorithm MDE, in order to determine the minimum achievable delay. Our goal is to measure the error between both delays obtained (exact, from TILOS and estimated, from Algorithm MDE).

Figs. 11 and 12 presents the comparison of Algorithm MDE with our implementation of TILOS for a few of the benchmark circuits. For each implementation, the first bar represents the delay of the unsized circuit. The second bar is the minimum delay obtained when the mapped circuit is sized using our implementation of TILOS, and the last bar is the minimum achievable delay estimated using Algorithm MDE. As can be seen by the correspondence between the last two bars for each implementation, our results agree with those obtained via TILOS. In every case, the execution time for our algorithm was less than a second, while our implementation of TILOS took from a few seconds for C17 up to more than 1500 seconds for C6288[6]. The average error for each circuit over all implementations is presented in Table I. Over all the benchmark circuits (59 in total), the average error is 6.01%. This error is due to the fact that most circuits are not trees, but have reconvergent fanouts. Our claim in Section IV-A, of assuming Delay-C_{in} curves of reconvergent fanouts to be independent is borne out by the small magnitude of the error.

B. Algorithm ADC

We next present the results of generating estimated area-delay curves of all implementations of a circuit, using Algorithm ADC. The first goal of this approach is to correctly predict which implementation has the lowest cost for different delay points. In order to measure this, we generate the area-delay curves for all implementations, using TILOS and Algorithm ADC. In the entire range of available delay values, we select ten equally spaced delay points. Note that the number of implementations that can be sized to meet a particular delay value varies by circuit, as can be seen from the area-delay curves shown in Fig. 2. We make pairwise comparisons between all implementations available at the selected delay point, and determine which implementation is better in each pair. In Table II, for each benchmark circuit, the number of comparisons made are shown in the second column. Next, we make the same comparison using the delay curves obtained from our implementation of TILOS. An incorrect comparison is when the ranking according to Algorithm ADC is different from that obtained from TILOS. As shown in the next column, incorrect comparisons occur only 6.71% of the time.

Next, we consider the error in the predicted area difference. When comparing implementations I_1 and I_2, the better implementation is the one with smaller area. Let the corresponding areas be $A_{I_1\text{est}}$ and $A_{I_2\text{est}}$, and assume $A_{I_1\text{est}} < A_{I_2\text{est}}$, so that I_1 is the better implementation. The difference between the estimated areas of I_1 and I_2, is calculated as $\Delta A_{\text{est}} = 100(1 - (A_{I_1\text{est}}/A_{I_2\text{est}}))$. The corresponding difference between the areas from the actual area-delay curves of I_1 and I_2, $A_{I_1\text{act}}$ and $A_{I_2\text{act}}$ is calculated as $\Delta A_{\text{act}} = 100(1 - (A_{I_1\text{act}}/A_{I_2\text{act}}))$. The maximum and average difference between ΔA_{est} and ΔA_{act} are presented in columns 4 and 5 of Table II. For example, for circuit C2670, Algorithm ADC overestimates the difference in areas between two implementations by an average of 3.49%, while the maximum error is 18.87%. The maximum error does not happen too often, and for all circuits, the average error is 5.07%, while the maximum error is 28.61%. The last two columns present the maximum and average errors in area estimation for comparisons that were mis-predicted. While the maximum is large, it is rare, the average error in this case is 5.71%. As mentioned previously, incorrect predictions themselves are very infrequent.

[6]This version does not use incremental timing analysis. However, given the large difference in run times, we expect our algorithm to still outperform TILOS, even if it were to use incremental timing analysis.

VI. Conclusion and Future Directions

In this paper, we identify and address two metrics that can be used to evaluate different implementations of a circuit. Using the algorithms presented in this paper, designers can quickly determine the minimum achievable delay that can be obtained by an implementation. We also present an algorithm for estimating the entire area-delay curve of all available implementations, so that given a target delay, the best implementation (in terms of area) can be selected. Both of these metrics are calculated without running an exact sizing tool, and are therefore fast, but do not sacrifice on accuracy, as shown by the results.

The concept of calculating and propagating Delay-C_{in} curves is general, and can be applied to different areas as well. For example, current placement tools try to provide a solution that is delay-optimal, among other objectives. However, they ignore the gains that may be obtained via sizing. Our approach can be used to guide the placement tool, in effect making it "transistor-sizing aware," so that the final solution is globally optimal. Another area of application is in technology mapping, where circuits are broken into trees which are mapped individually, simply estimating the load values at the output of each tree. In [21], Delay-C_{in} curves are used to determine the optimal assignment of loads at tree outputs, leading to superior mapped solutions.

References

[1] L. Stok, M. A. Iyer, and A. J. Sullivan, "Wavefront technology mapping," in *Proc. Design, Automation and Test in Europe Conf.*, Mar. 1999, pp. 531–536.

[2] B. Hu, Y. Watanabe, A. Kondratyev, and M. Marek-Sadowska, "Gain-based technology mapping for discrete-size cell libraries," in *Proc. IEEE/ACM Design Automation Conf.*, Jun. 2003, pp. 574–579.

[3] J. P. Fishburn and A. E. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design*, Nov. 1985, pp. 326–328.

[4] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S.-M. Kang, "An exact solution to the transistor sizing problem for CMOS circuits using convex optimization," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 12, no. 11, pp. 1621–1634, Nov. 1993.

[5] C.-P. Chen, C. C. N. Chu, and M. D. F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagragian relaxation," in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design*, Nov. 1998, pp. 617–624.

[6] V. Sundararajan, S. S. Sapatnekar, and K. K. Parhi, "Fast and exact transistor sizing based on iterative relaxation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 21, no. 5, pp. 568–581, May 2002.

[7] A. R. Conn, P. K. Coulman, R. A. Haring, G. L. Morrill, C. Visweswariah, and C. W. Wu, "JiffyTune: Circuit optimization using time-domain sensitivities," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, no. 12, pp. 1292–1309, Dec. 1998.

[8] X. Bai, C. Visweswariah, P. N. Strenski, and D. J. Hathaway, "Uncertainty-aware circuit optimization," in *Proc. IEEE/ACM Design Automation Conf.*, Jun. 2002, pp. 58–63.

[9] R. F. Sproull and I. E. Sutherland, "Theory of logical effort: Designing for speed on the back of an envelope," in *Proc. IEEE Advanced Research in VLSI*, Mar. 1991, pp. 1–16.

[10] I. Sutherland, B. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*. San Fransisco, CA: Morgan Kaufmann, 1999.

[11] S. K. Karandikar and S. S. Sapatnekar, "Fast comparisons of circuit implementations," in *Proc. Design, Automation and Test in Europe Conf.*, Feb. 2004, pp. 910–915.

[12] ——, "Fast estimation of area-delay tradeoffs in circuit sizing," in *Proc. IEEE Int. Symp. on Circuits and Systems*, May 2005, pp. 3575–3578.

[13] F. Beeftink, P. Kudva, D. Kung, and L. Stok, "Gate-size selection for standard cell libraries," in *Proc. IEEE/ACM Int.Conf. on Computer Aided Design*, Nov. 1998, pp. 545–550.

[14] W. Donath, P. Kudva, L. Stok, P. Villarrubia, L. Reddy, A. Sullivan, and K. Chakraborty, "Transformational placement and synthesis," in *Proc. Design, Automation and Test in Europe Conf.*, Mar. 2000, pp. 194–201.

[15] K. Sulimma, I. Neumann, L. van Ginneken, and W. Kunz, "Improving placement under the constant delay model," in *Proc.Design, Automation and Test in Europe Conf.*, Mar. 2002, pp. 677–682.

[16] L. Stok, D. S. Kung, D. Brand, A. D. Drumm, A. J. Sullivan, L. N. Reddy, N. Hieter, D. J. Geiger, H. H. Chao, and P. J. Osler, "BooleDozer: Logic synthesis for ASIC's," *IBM J. Res. Develop.*, vol. 40, no. 4, pp. 407–430, Jul. 1996.

[17] "Gain based synthesis: Speeding RTL to silicon," White Paper [Online]. Available: http://www.magma-da.com/articles/Magma_GBS_White_Paper.pdf

[18] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 2000, pp. 201–204.

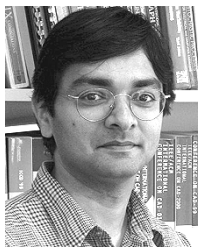
[19] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for sequential circuit synthesis," Univ. of California, Electronics Research Lab., Dep. EECS, Berkeley, Tech. Rep. UCB/ERL M92/41, May 1992.

[20] S. K. Karandikar, "Synthesis and performance prediction of VLSI designs," Ph.D. dissertation, Univ. of Minnesota, 2004.

[21] S. K. Karandikar and S. S. Sapatnekar, "Logical effort based technology mapping," in *Proc. IEEE/ACM International Conf. on Computer Aided Design*, Nov. 2004, pp. 419–422.

Shrirang K. Karandikar received the B.E. degree from the University of Pune, Pune, India, in 1994, the M. S. degree from Clarkson University,Potsdam, NY, in 1996, and the Ph.D. degree from the University of Minnesota, Minneapolis, in 2004.

He worked with Intel's Logic and Validation Technology group from 1997 to 1999, and is currently a researcher staff member at IBM's Austin Research Laboratory. His current interests are in the areas of logic synthesis and physical design of VLSI systems.



Sachin S. Sapatnekar received the B.Tech. degree from the Indian Institute of Technology, Bombay, India, in 1987, the M.S. degree from Syracuse University, Rochester, NY, in 1989, and the Ph.D. degree from the University of Illinois at Urbana-Champaign in 1992.

From 1992 to 1997, he was an assistant professor in the Department of Electrical and Computer Engineering at Iowa State University. He is currently the Robert and Marjorie Henle Professor in the Department of Electrical and Computer Engineering at the University of Minnesota, Minneapolis. He has authored several books and papers in the areas of timing and layout.

Dr. Sapatnekar has held positions on the editorial board of the IEEE Transactions on Very Large Scale Integration (VLSI) Systems, and the IEEE Transactions on Circuits and Systems—II: Regular Papers, *IEEE Design and Test*, and the IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems. He has served on the Technical Program Committee for various conferences, and as Technical Program and General Chair for Tau and ISPD, and Techical Program co-chair for DAC. He has been a Distinguished Visitor for the IEEE Computer Society and a Distinguished Lecturer for the IEEE Circuits and Systems Society. He is a recipient of the NSF Career Award, three best paper awards at DAC and one at ICCD, and the SRC Technical Excellence award.