Gate Sizing with Controlled Displacement*

Wei Chen, Cheng-Ta Hsieh, Massoud Pedram
Department of Electrical Engineering – System
University of Southern California, Los Angeles, CA 90089
{weich, chengtah, massoud} @zugros.usc.edu

Abstract - *In this paper, we present an algorithm for gate sizing with controlled displacement to improve the overall circuit timing. We use a path-based delay model to capture the timing constraints in the circuit. To reduce the problem size and improve the solution convergence, we iteratively identify and optimize the k-most critical paths in the circuit and their neighboring cells. All the operations are formulated and solved as mathematical programming problems by using efficient solution techniques. Experimental results on a set of benchmark circuits demonstrate the effectiveness of our approach compared to the conventional approaches, which separate gate sizing from gate placement.*

1 Introduction

Timing-driven CAD tools play an important role in the design of today's complex IC's. As the clock speed of VLSI circuits increases, the need for more aggressive timing optimization techniques and algorithms intensifies. Existing CAD tools and conventional design flows have not been able to cope with the rapidly tightening timing requirements in high-performance VLSI circuit. As a result, there is a great need for introducing new techniques and design flows for aggressive timing optimization. One class of techniques that appears to be particularly promising is the class of unification-based approaches, which attempt to combine certain optimization steps in the traditional design flows into one integrated step [1][2]. In this paper we present a unification-based algorithm for simultaneous gate sizing and placement of critical sections of a circuit.

Gate sizing, which has a significant impact on the circuit delay, has been an active research topic in recent years. In the conventional flows, in-place sizing follows timing-driven placement. Many approaches for gate sizing have been proposed. In general, these approaches can be divided into two categories: discrete and continuous sizing. In the discrete sizing methods, only a set of sizes is allowed for each gate. The best size for each gate in the circuit is determined by combinatorial or stochastic search. In [3], only a small section around the gate that is being sized is considered for timing recalculation to reduce the computation cost. The continuous sizing methods assume that the gate size of each gate type is a continuous variable, so the gate-sizing problem can be formulated as a mathematical programming problem. In TILOS [4], the area and delay are modeled by posynomial functions and only one gate is sized at a time. In [5], the area and delay of continuously sized gates are modeled piecewise linearly and all gates in the circuit are sized simultaneously. In [6] simultaneous gate sizing and wire sizing is solved by Lagrangian relaxation.

In both of these methods, only the gate sizes are adjusted to match the output loads of the gates, but the other dimension of optimization, i.e. adjusting the wire loads of the gates, is completely ignored. By moving the gates around, we can actually optimize the wire loads. That is especially important in deep sub-micron (DSM) designs where the effect of interconnects delay dominates the chip timing [7]. For DSM technologies, interconnect delay can easily account for more than 50% of the total delay. In this paper, we introduce a new iterative algorithm to tune both the gate sizes and wire loads (gate placement) for timing. Suppose an initial placement is given. The *k*-most critical paths in the placed circuit are identified and optimized. There are three timing-improvement steps used in our algorithm:

- Reposition the cells which are directly driven by the cells on the *k*-most critical paths;
- Size down the cells which are directly driven by the cells on the *k*-most critical paths;
- Simultaneously size and place the cells on the *k*-most critical paths.

Compared to the previous sizing approaches, which size one gate at a time in an iterative manner [3][4], our method has the advantage of sizing a relatively large number of gates (i.e. all gates on the *k*-most critical paths) at the same time. Furthermore we do gate sizing and placement of the immediate fan-out gates of the cells on the critical paths to reduce the load of the critical cells. We also perform simultaneous sizing and placement of the critical path cells. Compared to previous gate sizing approaches, which handle all the gates at the same time by using a mathematical programming formulation [5], our algorithm has the advantage of expanding the search space by doing simultaneous placement and sizing of the critical paths. Compared to [8] which formulates the problem of resizing and relocating gates of some initial placement as a linear program, we use a more accurate timing function and formulate the optimization problem as a generalized geometric program.

The rest of this paper is organized as follows. In Section 2, we present our timing model and the problem formulation. In Section 3, the optimization steps of our algorithm are discussed. The algorithm flow is given in Section 4. Techniques for solving GP and GGP problems are reviewed in Section 5. Experimental results and conclusions are given in Sections 6 and 7, respectively.

2 Timing Model and Problem Statement

The following notation will be used throughout this paper.

$d_{i,j}$	delay from output pin of gate g_i to output pin of gate g_j
$dint_{i,j}$	intrinsic delay of gate g_j for a transition coming from the input pin of the gate which is connected to the output of gate g_i
$rdr_{i,j}$	drive resistance of gate g_j for a transition coming from the input pin of the gate which is connected to the output of gate g_i
$cload_i$	input gate capacitance of the fan-outs of gate g_i

*This research was supported in part by the SRC contract No. 98-DJ-606.

$cnet_i$	lumped capacitance of the output net of gate g_i
$rnet_i$	lumped resistance of the output net of gate g_i
$cin_{i,j}$	input capacitance of gate g_j for the input pin of the gate which is connected to the output of gate g_i
a_i	actual arrival time of g_i
r_i	required arrival time of g_i
s_i	slack time of g_i
x_i,y_i	x & y coordinate of g_i
z_i	drive strength coefficient of g_i, used also to represent the size of g_i
C(k)	set of cells on the *k-most* critical paths in the circuit
Ne(i)	set of cells which have a directed shortest path from *C(k)* less or equal to *i* edges (i≥1)

2.1 Gate Delay Model With Continuous Sizing

Path delay in a circuit consists of net delay and gate delay. In this paper, net delay is calculated as a lumped model and added to the delay of the gate that drives this net. A gate level delay model similar to those used in [5] is adopted in this paper. Referring to Figure 1, $d_{i,j}$ is modeled as:

$$d_{i,j} = dint_{i,j} + rdr_{i,j} \cdot (cload_j + cnet_j) + rnet_j \cdot cload_j \quad (1)$$

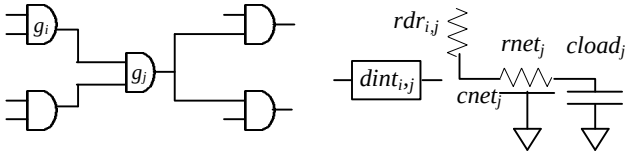

Figure 1. Gate delay model of g_j

$cload_j$ is given by: $$cload_j = \sum_{g_k \in fanout(g_j)} cin_{j,k} \quad (2)$$

In our formulation, gate g_j's size z_j is a variable, so $dint_{i,j}$, $rdr_{i,j}$ and $cin_{i,j}$ are all functions of z_j. We can use polynomial functions to fit $dint_{i,j}$, $rdr_{i,j}$ and $cin_{i,j}$ versus the gate size. For simplicity, we use first order polynomial functions. Notice however that any polynomial function can be used in our algorithm (cf. Section 5). In particular we use the following fitted equations:

$$\begin{aligned} dint_{i,j}(z_j) &= a1_{i,j} \cdot z_j + b1_{i,j} \\ rdr_{i,j}(z_j) &= \frac{a2_{i,j}}{z_j} + b2_{i,j} \\ cin_{i,j}(z_j) &= a3_{i,j} \cdot z_j + b3_{i,j} \end{aligned} \quad (3)$$

where $a1_{i,j}$, $a2_{i,j}$, $a3_{i,j}$ and $b1_{i,j}$, $b2_{i,j}$, $b3_{i,j}$ are the regression coefficients.

2.2 Wire Load Estimation

To estimate the wire load, the minimum bounding-box (MBB) is used here. Consider net net_i driven by gate g_i as shown in Figure 2

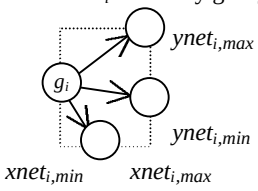

Figure 2. Net bounding-box model

the capacitance $cnet_i$ and resistance $rnet_i$ of net net_i is given by:

$$\begin{aligned} cnet_i &= \rho \cdot [C_{hor}(xnet_{i,\max} - xnet_{i,\min}) + C_{ver}(ynet_{i,\max} - ynet_{i,\min})] \\ rnet_i &= \rho \cdot [R_{hor}(xnet_{i,\max} - xnet_{i,\min}) + R_{ver}(ynet_{i,\max} - ynet_{i,\min})] \end{aligned} \quad (4)$$

where $\begin{cases} xnet_{i,\max} = \max_{g_j}\{x_j\}, xnet_{i,\min} = \min_{g_j}\{x_j\}, \\ ynet_{i,\max} = \max_{g_j}\{y_j\}\ ynet_{i,\min} = \min_{g_j}\{y_j\}. \end{cases}$

g_j is any gate connected to net_i; C_{ver}, C_{hor}, R_{ver} and R_{hor} are constants related to the process technology and geometry of wires, which describe the capacitance per unit length of vertical and horizontal wires and the resistance per unit length of vertical and horizontal wires, respectively. ρ is a parameter used to adjust the estimation error of the bounding box interconnect model [9]. For $n \le 10$, the values of ρ are produced in Table 1. We use equation (5) for $n>10$:

$$\lim_{n->\infty} \rho = (\sqrt{n} + 1)/2 \quad (5)$$

n	2	3	4	5	6	7	8	9	10
ρ	1	1	3/2	3/2	5/3	7/4	11/6	2	2

Table 1: Worst case equi-perimeter net lengths.

Combining equations (1), (2), (3) and (4), our pin-dependent delay model $d_{i,j}$ can be written as:

$$\begin{aligned} d_{i,j} = {} & dint_{i,j}(z_j) + rdr_{i,j}(z_j) \cdot \{\rho \cdot C_{hor}(xnet_{j,\max} - xnet_{j,\min}) \\ & + \rho \cdot C_{ver}(ynet_{j,\max} - ynet_{j,\min}) + \sum_{g_k \in fanout(g_j)} cin_{j,k}(z_k)\} \\ & + \rho \cdot \{R_{hor}(xnet_{j,\max} - xnet_{j,\min}) + R_{ver}(ynet_{j,\max} - ynet_{j,\min})\} \\ & \cdot \sum_{g_k \in fanout(g_j)} cin_{j,k}(z_k) \end{aligned} \quad (6)$$

Theorem The polynomial function corresponding to $d_{i,j}$ is a non-convex function of its variables $xnet_{j,max}$, $xnet_{j,min}$, $ynet_{j,max}$, $ynet_{j,min}$, z_j and z_k.

Proof Since $d_{i,j}$ is the product and sum of polynomial functions, it is a polynomial function itself. Hessian matrix F of function f is the matrix of the 2nd partial derivatives of f. Function f is convex over a convex set Ω containing an interior point if and only if the Hessian matrix F of f is positive semidefinite throughout Ω [10]. For $d_{i,j}$ given in equation (6), the Hessian matrix is not guaranteed to be positive semidefinite. So our delay mode is in general non-convex.

2.3 Timing Analysis

Let directed graph *G(V, A)* represent the netlist of a circuit. The vertex set *V* is in one-to-one correspondence with the set of gates whereas the edge set *A* represents the source-to-sink connections between gates. Associated with each gate g_i in the circuit, there exist a required arrival time r_i and an actual arrival time a_i. The arrival times for primary inputs and the required arrival times for primary outputs are specified by the designer of the circuit.

The actual arrival time a_j is given by

$$a_j = \max\{(a_i + d_{i,j}) \mid \forall (v_i, v_j) \in A\}$$

The required arrival time r_i is given by

$$r_i = \min\{(r_j - d_{i,j}) \mid \forall (v_i, v_j) \in A\}$$

where $d_{i,j}$ is defined in equation (6).

A *critical path* is a path in which the sequence of vertices $(v_i,\ldots,v_o)$, $v_i \in$ primary input, $v_o \in$ primary output which comprise the path, all have slack values less than or equal to zero. g_i's slack time s_i is defined $s_i = r_i - a_i$.

2.4 Initial Problem Formulation

A formulation of our problem can be written as (7). The last two equality constraints describe the center of mass constraints imposed during the optimization in order to spread the cells evenly on the whole chip. These constraints are used commonly in placement programs that interleave quadratic programming with circuit bipartitioning. Examples include Gordian[11], Speed[12]. In general *n* is the number of parts at the current partitioning step.

T_{start} is the actual arrival time at the circuit primary inputs. $part_j$ denotes a part, w_i is the area of g_i and xc_j, yc_j are the geometric centers of $part_j$.

$$
\begin{aligned}
&\textit{minimize} && t_{cycle} && \\
&\textit{s.t.} && a_j \ge a_i + d_{i,j} && \forall (v_i, v_j) \in A \\
& && a_j \le t_{cycle} && \forall v_j \in \textit{primary outputs} \\
& && a_j \ge T_{start} && \forall v_j \in \textit{primary inputs} \\
& && \frac{1}{|part_j|} \sum_{i \in part_j} w_i \cdot x_i = xc_j && j = 1,\ldots,n \\
& && \frac{1}{|part_j|} \sum_{i \in part_j} w_i \cdot y_i = yc_j && j = 1,\ldots,n
\end{aligned}
\qquad (7)
$$

Observation The timing constraint functions in the above problem are polynomial functions. In general, they are non-convex functions. The solution to this problem requires a non-convex programming algorithm.

Unfortunately, for a small circuit size (around 100 cells), the above formulation is still too complicate to be solved by mathematical programming packages. Furthermore, the above formulation results in a lot of gate overlaps, which is undesirable. Notice that it is difficult to use recursive circuit partitioning with this formulation since cuts in the previous levels may not maintain the cell area balance because size of the cells may change by a large amount in subsequent optimization steps.

3 Optimization Phase

By iteratively finding and optimizing *C(k)*, the timing of the whole circuit can be improved gradually while the problem size remains manageable. To solve the congestion problem, instead of the recursive partitioning approach, we limit the sizes and the locations of *C(k)* to a certain range in each iteration. De-congestion is applied at the end of placement to guarantee the placement is acceptable all the time. The precise formulation of *C(k)*'s sizing and placement problem is given in Section 3.3.

To reduce the delay of a certain cell, we can size down its fan-out cells or pull its fan-out cells closer to reduce its load. So to further improve the timing of the critical paths, the capacitance load imposed on *C(k)* by the corresponding *Ne(i)* should be considered too. (In this paper, only *Ne(1)* is sized down and re-placed). If however the sizes and positions of *Ne(1)* are changed without control, new critical paths (which go through *Ne(1)* or any other cell which is a transitive fan-in or fan-out of *Ne(1)*) may be created due to the drive strength or load changes of *Ne(1)*. The delay of these new critical paths may be even worse than the critical path under consideration. This may lead to cyclic timing violation problems or slow down the convergence speed.

To size down and place *Ne(1)* optimally while completely avoiding the cyclic timing violation problem, we should change the locations and sizes of both *C(k)* and *Ne(1)* simultaneously. If however the locations and sizes of *Ne(1)* are added to *C(k)* sizing and placement problem as variables, since either the number of the gates in *Ne(1)* or the number of the paths which pass through *Ne(1)* is often much larger than that of *C(k)*, the problem size would become unmanageable. So in our approach, we optimize *C(k)* and *Ne(1)* separately. Notice also that if the *Ne(1)* gates are sized down and repositioned simultaneously, the problem assumes a similar form as formulation (10) which is a non-convex problem. Again in general the number of cells in *Ne(1)* is much larger than that of *C(k)*; furthermore, it is not easy to control the number of gates in *Ne(1)* as it is to control the number of the gates in *C(k)* by reducing *k*. The optimization of *Ne(1)* is therefore done in two steps: *Ne(1)* re-placement and *Ne(1)* resizing which are discussed in Section 3.2 and Section 3.3. In our approach, we therefore end up with three optimization steps per iteration: *Ne(1)* re-placement, *Ne(1)* sizing down, and *C(k)* sizing and placement.

3.1 Ne(1) Re-placement

In this step only the locations of *Ne(1)* are variables. The mathematical formulation is:

$$
\begin{aligned}
&\textit{minimize} && t_{cycle} && \\
&\textit{s.t.} && a_j \ge a_i + d_{i,j} && \forall (v_i, v_j) \in A \\
& && a_j \le t_{cycle} && \forall v_j \in \textit{primary outputs and } v_j \in C(k) \\
& && a_j \le \gamma\, T_{critical} && \forall v_j \in \textit{primary outputs and } v_j \notin C(k) \\
& && a_j \ge T_{start} && \forall v_j \in \textit{primary inputs} \\
& && |x_i - x_i'| \le \Delta_x && \forall v_i \in Ne(1) \\
& && |y_i - y_i'| \le \Delta_y && \forall v_i \in Ne(1)
\end{aligned}
\qquad (8)
$$

where x_i', y_i' are the location coordinates of g_i from the previous iteration; Δ_x, Δ_y are the *position variable change regions* defined by the optimization schedule which will be discussed in Section 4. The new cell locations are controlled by Δ_x, Δ_y. $T_{ciritcal}$ is the latest arrival time of any primary output belonging to *C(k)* from the previous iteration; γ is a constant ($0<\gamma<1$) to control how much of the spare slack time of *Ne(1)* is used for re-placement so as to optimize the timing of *C(k)*. The third constraint ensures that after this step, no path with a delay longer than the delay of the current most critical path will be created so that the timing convergence is guranteed.

Considering the delay equation (6), since only the cell locations are variable, $d_{i,j}$ in formulation (8) becomes a linear function. So *Ne(1)* re-placement is a Linear Programming problem. We use the LP-Solver of [13] to solve this problem. Furthermore, if the cells connected to the net which is driven by g_j, are not changed, then delays $d_{i,j}$ will be constant. So these redundant timing constraints in (8) can be removed to reduce the problem size significantly.

Notice a non-overlapping constraint is not imposed in the formulation of (8). If the position change regions overlap, there may be cell congestion. This issue can become detrimental if we do not perform de-congestion. In our algorithm after problem (8) is solved, the size and ideal location of every cell is determined. Initially each cell is assigned to the row which is the closet to its ideal location (We assume row-based layout). Cells in the same row are placed in the order of their x-axis coordinates. Next one cell from the longest row is moved up/down to the shorter one of its immediately adjacent rows. The other cells in these two rows (i.e. longer and shorter adjacent rows) are shifted to close the gap or create the space as required. This process is repeated until all the rows have nearly the same length.

3.2 Ne(1) Sizing Down

In this step, only the sizes of *Ne(1)* are variables. The mathematical formulation is:

$$
\begin{aligned}
&\textit{minimize} && t_{cycle} && \\
&\textit{s.t.} && a_j \ge a_i + d_{i,j} && \forall (v_i, v_j) \in A \\
& && a_j \le t_{cycle} && \forall v_j \in \textit{primary outputs and } v_j \in C(k) \\
& && a_j \le \lambda\, T_{critical} && \forall v_j \in \textit{primary outputs and } v_j \notin C(k) \\
& && a_j \ge T_{start} && \forall v_j \in \textit{primary inputs} \\
& && |z_i - z_i'| \le \Delta_z && \forall v_i \in Ne(1)
\end{aligned}
\qquad (9)
$$

where z_i' is the size of g_i from the previous iteration; Δ_z is the *size variable change regions* defined by the optimization schedule. λ is a constant ($0<\lambda<1$) whose function is similar to that of γ in *Ne(1)* re-placement. Because of a similar reason as in *Ne(1)* re-

placement, there are many redundant timing constraints in formulation (9). The problem size can be reduced by deleting these redundant constraints. equation (9) can be solved by Geometric Programming (GP), which is discussed in Section 5.

3.3 C(k) Sizing and Placement

The mathematical formulation of the *C(k)* sizing and placement problem becomes:

$$\begin{aligned} &\textit{minimize} && t_{cycle} \\ &\textit{s.t.} && a_j \geq a_i + d_{i,j} \quad \forall (v_i, v_j) \in A \quad v_i, v_j \in C(k) \\ & && a_j \leq t_{cycle} \quad \forall v_j \in \textit{primary outputs}, v_i \in C(k) \\ & && a_j \geq T_{start} \quad \forall v_j \in \textit{primary inputs}, v_i \in C(k) \\ & && |x_i - x_i'| \leq \Delta_x \quad \forall v_i \in C(k) \\ & && |y_i - y_i'| \leq \Delta_y \quad \forall v_i \in C(k) \\ & && |z_i - z_i'| \leq \Delta_z \quad \forall v_i \in C(k) \end{aligned} \tag{10}$$

Equation (10) is a non-convex programming problem. It is the key operation for improving the timing of the cicuit and it is the most expensive step of our algorithm. Equation (10) can be solved by Generalized Geometric Programming (GGP) which is described in Section 5.

There may be a cell congestion problem after equation (10) is solved; the decongestion step described in Section 3.2 is therefore applied at the end of this optimization step.

4 Optimization Flow

In the mathematical formulations of (8), (9) and (10), there are predefined variable change ranges Δ_x, Δ_y, Δ_z. These change ranges are related to the convergence speed and the degree of cell congestion. If the region is too large, there will be a higher possibility of cell congestion. It is also possible that the result of the current timing optimization will adversely impact the timing of other paths. However, if the region is too small, more optimization passes will be probably needed.

Although we have incorporated some methods to improve the convergence speed of our algorithm, because the optimization is done locally, it is still possible that the solution does not converge at all or converges very slowly. To address this problem, we introduce a cooling schedule to control the variable freedom. As the iteration count increases, Δ_x, Δ_y, Δ_z are decreased. Finally, the freedom becomes so small that the circuit timing does not change. At that time, the process ends. The schedule also determines the total computation time. If the circuit designer is not too concerned with the runtime of the algorithm, a slower cooling schedule can be used to generate a higher quality result.

4.1 Selection of Discrete Gate Sizes

After solving equations (9) and (10) gate sizes are given as real numbers which will likely not match the given gate sizes in the standard cell library. Therefore at the end of these optimizations, we need to round the size of each gate to the size of the closest in the library. In general each continuous gate size can be matched to at most two discrete gate sizes; one which is just smaller, the other which is just larger than the specified size.We now consider the problem of discrete gate sizing for minimum delay along the set of paths in *Ne(1)* (for equation (9)) or *C(k)* (for equation (10)) when we are given at most two sizes for each gate. These sizes are derived from the continuous sizing solution as explained above. This problem is solved using a dynamic programming technique similar to that of [14]. In this way, we avoid the arbitrary and error-prone technique of simply rounding up the continuous sizing solution to a discrete solution.

4.2 Algorithm Flow

The main loop of this algorithm consists of three parts: (a) *Ne(1)* re-placement (b) *Ne(1)* sizing down and (c) simultaneous *C(k)* sizing and placement. Step (c) is the most effective and elaborate one and it should be done after both the cell sizes and locations are known. Therefore it should be done after steps (a) and (b). As mentioned before, step (a) may cause a congestion problem. Step (b) in general has more potential to improve the *C(k)* timing while creating less congestion problem. So we decide to use more of the spare slack time of *Ne(1)* during step (b) and do *Ne(1)* re-placement first to provide the correct cell locations for *Ne(1)* sizing down.

As the iteration count increases, the number of critical paths increases. So we end up increasing the maximum allowed size of *C(k)*. We keep doing this until the size of *C(k)* becomes too large to handle, in which case we stop the optimization process.

When all the optimization iterations end, the dynamic programming based discrete gate selection method is used to convert the continuous gate sizes to discrete gate sizes.

The complete flow of this algorithm is as following:

1) timing-driven initial placement
2) timing analysis
3) adjust Δ
4) *Ne(1)* re-placement
5) *Ne(1)* sizing
6) simultaneous *C(k)* sizing & placement
7) if (a)there is improvement or specification not satisfied, and (b) the problem is solvable, go to 2)
8) gate size selection
9) end

5 GP and GGP

We first give some definitions and theorems of GP and GGP.

Definition *Geometric Programming* (GP) is a class of nonlinear optimization problems having objective functions and constraint functions expressed as *posynomials*.

Definition *Generalized Geometric Programming* (GGP) is a class of nonlinear optimization problems having objective functions and constraint functions expressed as *polynomials*.

Note that GP is a convex programming problem[15], and GGP is a non-convex programming problem **[15]**.

Theorem Gate sizing problem as in equation (9) is a GP problem.

Theorem Simultaneous gate sizing and placement problem as in equation (10) is a GGP problem.

Proof Follows easily from the equation (9), (10) and the above definitions.

5.1 Geometric Programming (GP)

By using the variable substitution *ln(x)=w*, GP can be transformed to a linear programming problem. We use the method of[16]. Our computational results indicate that this GP algorithm leads to an effective and stable implementation for solving our problem.

5.2 Generalized Geometric Programming (GGP)

To solve GGP problem, we implement the algorithm [17]. This algorithm first introduces a new variable. The original nonlinear objective function is absorbed as an additional constraint.

$$\text{minimize} \quad x_0$$
$$\text{s.t.} \quad g_0(x) \le x_0,\ g_k(x) \le 0, \quad k = 1,2,...,m$$

where $g_0, g_1, \ldots\ldots, g_k$ are polynomial functions, g_0 is the original objective function.

Next, each polynomial is separated into its positive and negative parts, giving differences of pairs of posynomial functions:

$$\text{minimize} \quad x_0$$
$$\text{s.t.} \quad p_0^+(x) - p_0^-(x) \le x_0,\ p_k^+(x) - p_k^-(x) \le 0, \quad k = 1,2,...,m$$

where $p^+_k(x)$ and $p^-_k(x)$ are posynomial functions.

Then all the negative terms are brought to the right-hand side of the inequalities and divided through to yield a quotient form:

$$\text{minimize} \quad x_0$$
$$\text{s.t.} \quad \frac{p_0^+(x)}{p_0^-(x) + x_0} \le 1,\ \frac{p_k^+(x)}{p_k^-(x)} \le 1, \quad k = 1,2,...,m$$

Next, the denominator of each constraint is condensed at the operating point. Condensation is the process of approximating a posynomial function with a monomial function[18]. It is based on the weighted arithmetic-geometric (A-G) mean inequality:

$$\sum_i u_i \ge \prod_i \left(\frac{u_i}{\delta_i}\right)^{\delta_i}$$

where u_i is positive value, δ_i is positive weight and $\sum \delta_i = 1$. The approximation monomial produced is dependent on the selection of weights, which can be any set of positive values that sum to unity. One very useful choice is to set the weights equal to the fraction that each monomial term u_i of the posynomial function p contributes to the total value of the posynomial, when evaluated at some operating point x': $\delta_i = \frac{u_i(x')}{p(x')}$.

Condensing a posynomial to a monomial may be represented symbolically as:

$$C[p(x), x'] = \prod_{i=1}^{t} [u_i(x)/\delta_i]^{\delta_i}$$

Condensing the denominator of each constraints at the operating point results in a posynomial divided by an approximating monomial, that is an approximating posynomial that is always greater than or equal to the parent form.

$$\frac{p^+(x)}{C[p^-(x), x']} \ge \frac{p^+(x)}{p^-(x)}$$

Since the inequality relations hold for all positive values of x, the feasible side of the approximating posynomial constraint is a subset of the feasible side of the parent constraint. This is important because it shows that the approximation does not violate the original constraint.

GGP algorithm can be viewed as a loop. In the loop, the original GGP is condensed according to the variables' initial values, then it is transformed to a GP, and this GP is solved. The solution to this GP is used to condense the GGP at the next iteration.

Theorem The sequence of optimal solutions to the GP sequence converges to a point satisfying the Kuhn-Tucker necessary conditions for the optimality of GGP [17].

This algorithm requires a feasible initial solution at the beginning. For our problem, any initial placement of a mapped netlist forms a feasible solution. Of course, we are well-advised to start with a timing-driven placement result and a timing-driven technology mapped circuit.

6 Experimental Results

We have implemented our algorithm in C++ as a software package named SCD (Sizing with Controlled Displacement).

The following two figures show some snapshots of the results of SCD during the optimization of the benchmark C499. In this example, at the beginning Δ_x is set to 4 times of the average cell width, Δ_y is set to 4 times of the slot height plus routing channel height, and Δ_z is set to be maximum allowed change. Figure 3 is the result of one optimization iteration using the above values. Since each cell has a large change region, the critical path timing is improved by a lot. Note that the path layout is very different in the two cases shown in Figure 3. Particularly the path length is much shorter in Figure 3.2 than it is in Figure 3.1.

After a number of iterations, the cell freedom is reduced. In this case, it may take several iterations to optimize the most critical path until another path becomes the most critical. Figure 4 shows 3 consecutive iterations to optimize another path of C499. Here Δ_x is twice the average cell width and Δ_y is twice the slot height plus routing channel height, and Δ_z is half of the maximum allowed change. We can see that as a result of successive iterations, the path becomes more and more straight. However the change in path layout is less dramatic than that seen in Figure 4 because of small variable change ranges.

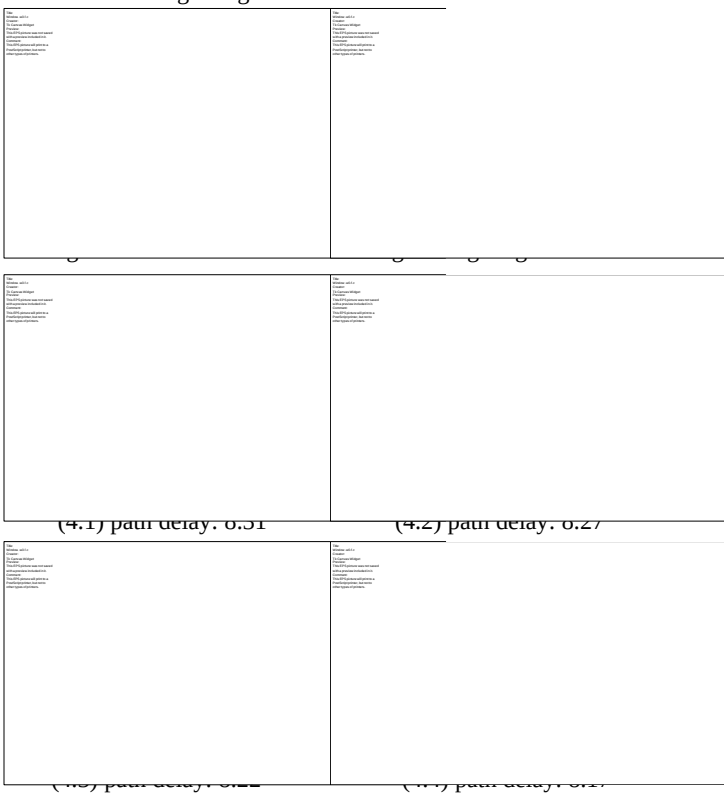

Figure 4 Results of multiple iterations with small change regions

We next calculate the cell slacks for a required arrival time at all primary outputs set to be T_{crit} where T_{crit} is the longest path delay. We define the normalized slack of a cell as the ratio of the cell slack compared to the longest path delay in the circuit. For example, a normalized slack of 0 means the cell is on the critical timing path. Note that a normalized slack of 1 can never be reached (It means zero delay path exists). In Figure 5, we draw the normalized slack distribution plot for C499 before and after optimization by SCD. Note that T_{crit} before SCD optimization is

13.91ns and after SCD optimization it is 6.04ns. The plot clearly shows that as a result of SCD optimization, 1) the number of cells with the same normalized slack value has increased, and 2) the percentage of critical cells in the circuit have increased, that is, the path delay distribution of cells has narrowed down. Therefore, we conclude that SCD achieves to improve timing by balancing the path delays, i.e. longer delay paths get shorter as the expense of shorter delay paths getting longer.

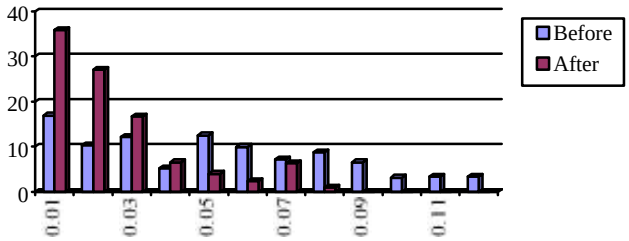

Figure 5 Distribution of the normalized slack time

Our algorithm has been applied to ISCAS benchmark circuits mapped to a 0.35μ industrial cell library. In this library, we have four gate sizes per gate type. The result is compared with the in-place gate-sizing (IPS) results. For both methods, the circuit is placed by TimberWolf first. The first method does in-place gate-sizing which keeps the cell locations fixed. The second method uses the SCD approach. The number of most critical paths considered in the optimization, i.e. *k*, was set such that $C(k) \leq 100$ for each benchmark. The average improvement is about 15% (We also have generated results with initial placement done by Gordian+Domino. Those results, which are similar to the ones reported in Table 2 are not reported here). The SCD runtime is on average 20 times higher than that of IPS. However the timing improvement justifies the increased runtime. (Software programs are run on Pentium II 300). All results are reported after detailed placement and detailed routing using YACR. The delays include the gate delay and post-layout interconnect delays.

7 Conclusions

We presented a new algorithm to do placement and gate-sizing simultaneously. Our algorithm improves the timing performance by decreasing the delay of the *k*-most critical paths iteratively. During each iteration both the cells on these critical paths and the immediate fan-outs of those cells are sized and placed. Appropriate mathematical programming methods are used to solve these problems. Future work will include integration of more powerful logic restructuring techniques with cell placement.

Chip	Cell	Level	Delay of TW Placement	Delay of In-Place Sizing(IPS)	Area of Chip (IPS)	In-Place Sizing CPU Time (s)	Delay of SCD	Area of Chip (SCD)	SCD CPU Time (s)	Improvement (%) over IPS
C432	215	31	20.90	9.42	299	50	8.56	303	584	9.1
C880	383	43	21.94	8.92	531	42	7.64	532	798	14.3
C1355	432	20	14.60	7.15	610	40	6.06	614	583	15.2
C1908	453	34	20.28	9.63	654	96	8.22	660	1356	14.6
i6	485	8	9.96	4.94	670	24	4.33	675	367	12.3
C499	502	21	13.91	6.89	712	101	6.04	724	1726	12.4
t481	713	18	12.76	6.31	1002	120	5.36	1031	2508	15.1
C2670	848	24	19.60	8.97	1150	33	7.51	1173	1070	16.3
k2a	922	22	20.04	10.10	1394	113	8.62	1420	2394	14.7
C3540	1151	48	32.93	17.24	1502	211	14.57	1523	5230	15.5
C5315	1640	33	28.85	14.50	2301	121	12.43	2398	2826	14.3
C7552	2156	55	45.72	20.12	3169	251	16.90	3241	6349	16.0
des	3059	29	22.49	11.50	4238	510	9.40	4302	22023	18.2

Table 2. Experimental results

Reference:

[1] J.Lou, A.Salek, M.Pedram, *"An Exact Solution to Simultaneous Technology Mapping and Linear Placement Problem"*, *Proc. Intl. Conf. on CAD*, pp.671-675, Nov 1997.

[2] A.Salek, J.Lou, M.Pedram, *"A Simultaneous Routing Tree and Fanout Optimization Algorithm"*, *Proc. Intl. Conf. on CAD*, pp.625-630, Nov 1998.

[3] O.Coudert, R. Haddad, *"New Algorithms for Gate Sizing: a Comparative Study"*, *Proc. 33rd DAC*, pp.734-739, Jun 1996.

[4] J.P.Fishburn, A.E.Dunlop, *"TILOS: a Posynomial Programming Approach to Transistor Sizing"*, *Proc. Intl. Conf. on CAD*, pp.326-328, Nov 1985.

[5] M. Berkelaar, J. Jess, *"Gate Sizing in MOS Digital Circuits with Linear Programming"*, *Proc. European DAC*, pp.217-221, 1990.

[6] C.P.Chen, C.C.N.Chu, D.F.Wong, *"Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation"*, *Proc. Intl. Conf. on CAD*, pp.617-624. Nov 1998.

[7] *"National Technology Roadmap"*, Semiconductor Industry Association, 1997.

[8] W. Chuang, I.N.Hajj, *"Delay and Area Optimization for Compact Placement by Gate Resizing and Relocation"*, *Proc. Intl. Conf. on CAD*, pp.145-148, Nov 1994.

[9] F.R.K.Chung, F.K.Hwang, *"The Largest Minimal Rectilinear Steiner Trees for a Set of N Points Enclosed in a Rectangle with Given Perimeter"*, *"Networks"*, 9:19-36, 1979.

[10] D.Luenberger, *"Linear and Nonlinear Programming"*, pp.180, 1984.

[11] J.M.Kleinhans, G.Sigl, F.M.Johannes, K.J.Antreich, *"GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization"*, *IEEE Trans. on Computer-Aided Design*, vol.10, No.3, pp.356-365, Mar 1991.

[12] B.M. Riess, G.G. Ettelt, *"SPEED: Fast and Efficient Timing Driven Placement"*, *Proc. Intl. Symposium of Circuits and Systems*, pp.377-380, 1995.

[13] M. Berkelaar, *"Area-Power-Delay Trade-off in Logic Synthesis"*, Ph.D Thesis, Eindhoven University of Technology, 1992.

[14] P.K.Chan, *"Algorithms for Library-specific Sizing of Combinational Logic"*, *Proc. 27th DAC*, pp.353-356, 1990.

[15] C.Beightler, D.T.Philips, *"Applied Geometric Programming"*, 1976.

[16] K. O. Kortanek, X. Xu, Y.Ye, *"An infeasible interior-point algorithm for solving primal and dual geometric programs"*, *Mathematical Programming* 76, pp.155-181, 1996.

[17] M.Avriel, R.Dembo, U.Passy, *"Solution of Generalized Geometric Programming"*, *International Journal for Numerical Methods in Engineering*, vol.9, 1975.

[18] R.J. Duffin, *"Linearizing Geometric Programs"*, *SIAM Review*, vol.12, pp.211-237, 1970.