Simultaneous Gate Sizing and Placement

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. More precisely in each iteration, we perform three operations: a) reposition the immediate fan-outs of the gates on the k-most critical paths; b) size down the immediate fan-outs of the gates on the k-most critical paths; c) simultaneously reposition and resize the gates on the k-most critical paths. Each of these operations is formulated and solved as a mathematical program 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. This trend is expected to escalate as the minimum feature size scales to the sub-quarter-micron region. 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. Examples include techniques for simultaneous technology mapping and placement[1], simultaneous fan-out optimization and Steiner routing[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 (currently practiced in industry) timing driven cell placement is followed by in-place gate sizing in order to correct timing violations that may have remained in the circuit after technology mapping and cell placement.

Many approaches for gate sizing have been proposed. In general, these approaches can be divided into two categories: discrete sizing and continuous sizing. In the discrete gate sizing method, 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. Note that when the size of a gate in the circuit is changed, the signal arrival time for all the gates that are in the transitive fan-in or fan-out cones of the sized gate may change. As a result, the circuit timing analysis must be repeated after each sizing step. The cost of such dynamic timing update is quite high. In [3], only a small section around the gate that is being sized is considered for timing recalculation to reduce the computation cost. For libraries with a large number of choices for each gate type, this method may become slow. For libraries with a small number of choices for each gate type, the discrete gate sizing may outperform the continuous sizing method due to the highly discrete nature of the optimization variables. The continuous sizing methods assume that the gate size of each gate type is a continuous variable. As a result, 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. Simultaneous gate sizing and wire sizing is solved by Lagrangian relaxation in [6].

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 explore the other dimension i.e. changing 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. It is necessary to develop algorithms and tools for concurrent gate sizing and placement as well as computational delay models that account for both gate and interconnect delays during this joint optimization process.

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.

The first two steps are used to reduce the loads of the cells on the critical paths. The last step is used to optimize the cells on the critical paths directly. Each optimization step is formulated as a mathematical programming problem. We solve the first problem by Linear Programming and the second one by

Geometric Programming. The third problem, which is the most complicated case, is a non-convex, non-linear problem. We solve it by Generalized Geometric Programming (cf. Section 6). A heuristic is used to simplify the third problem to an acceptable size. The optimization process is terminated when there is no improvement in the current iteration compared to the previous *t* iterations or the specification is achieved.

Compared to the previous sizing approaches, which size one gate at a time in an iterative manner (or by using simulated annealing)[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 size and place the immediate fan-out gates of cells on the critical paths to reduce the load of these critical cells. Finally we perform simultaneous sizing and placement of the critical path cells. Hence better solution quality can be obtained. Compared to previous gate sizing approaches, which handle all the gates at the same time by using a linear programming formulation[5], our algorithm has the advantage of expanding the search space by doing simultaneous placement and sizing of gates (although we do not optimize all the gates in the circuit in one shot due to the problem complexity). Compared to [8] which formulates the problem of resizing and relocating gates from some initial placements as a piecewise linear program, we use a more accurate timing function and formulate the optimization problem as a generalized geometric program.

The remainder of this paper is organized as follows. In Section 2, we present our gate sizing model and the initial problem formulation. In Section 3 simultaneous gate sizing and placement on the *k*-most critical paths is discussed. The optimization of the fan-outs of the *k*-most critical paths is given in Section 4. Algorithm flow is shown in Section 5. Technologies for solving GP and GGP problems are described in Section 6. Experimental results and conclusions are given in Sections 7 and 8, respectively.

2 Timing Model and Problem Statement

The following notation will be used throughout this paper.

$G(V,A)$	A directed acyclic graph representation of the circuit; V is the vertex set (cells), A is the edge set (cell connections), PI is the set of primary inputs and PO is the set of primary outputs; n is the number of cells, e is the number of edges, q is the number of primary inputs and outputs
$d_{i,j}$	delay from the output pin of gate g_i to the 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
$cnet_i$	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 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	x-axis coordinate of g_i
y_i	y-axis coordinate of g_i
z_i	drive strength 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; *n'* is the number of cells, *e'* is the number of edges, and *q'* is the number of primary inputs and outputs in *C(k)*
Ne(k,i)	set of cells which are in the transitive fanout of *C(k)* and receive a directed shortest path from *C(k)* which is *i* edges or fewer; *n"* is the number of cells in *Ne(k,i)*
$DVCR_i$	dynamic variable change region of g_i

The reader should especially note the definitions of *C(k)* and *Ne(k,i).*

2.1 Gate Delay Model With Continuous Sizing

Path delay in a circuit consists of two components: net delay and gate delay. In this paper, the 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, $d_{i,j}$ can be thought as the delay from the input pin of g_j, which connects the output of g_i, to the input pin of the gate which is driven by g_j. $d_{i,j}$ is modeled as:

$$d_{i,j} = d\,int_{i,j} + rdr_{i,j} \cdot (cload_j + cnet_j) + rnet_j \cdot cload_j \tag{1}$$

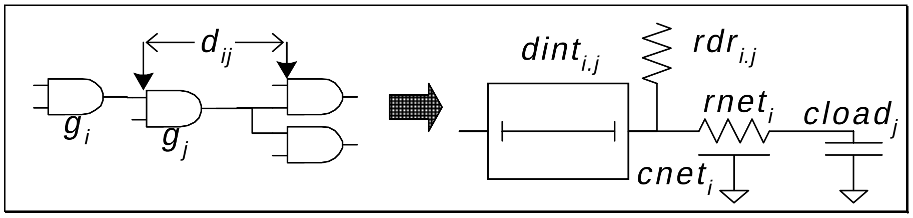


Figure 1. Gate delay model

$cload_j$ is given by:

$$cload_j = \sum_{g_k \in fanout(g_j)} cin_{j,k} \tag{2}$$

When the gate size is fixed, $dint_{i,j}$, $rdr_{i,j}$ and $cin_{i,j}$ are constants. 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 order polynomial function can be used in our algorithm (cf. Section 6). In particular we use the following fitted equations:

$$\begin{aligned} d\mathrm{int}_{i,j}(z_j) &= \alpha 1_{i,j} \cdot z_j + \beta 1_{i,j} \\ rdr_{i,j}(z_j) &= \frac{\alpha 2_{i,j}}{z_j} + \beta 2_{i,j} \\ cin_{i,j}(z_j) &= \alpha 3_{i,j} \cdot z_j + \beta 3_{i,j} \end{aligned} \tag{3}$$

where $\alpha 1_{i,j}$, $\alpha 2_{i,j}$, $\alpha 3_{i,j}$ and $\beta 1_{i,j}$, $\beta 2_{i,j}$, $\beta 3_{i,j}$ are the regression coefficients.

2.2 Wire Load Estimation

To make the timing formulation continuous, the minimum bounding-box (MBB) model is used to estimate the wire load. More precisely, the delay of a net is related to the half-perimeter length of the MBB of the net. Consider net net_i driven by gate g_i as shown in Figure 2.

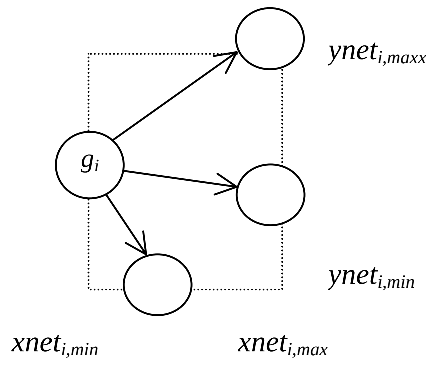


Figure 2. Net bounding-box model.

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

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

where $xnet_{i,\max} = \max_{g_j}\{x_j\}, xnet_{i,\min} = \min_{g_j}\{x_j\}, ynet_{i,\max} = \max_{g_j}\{y_j\} \text{ and } ynet_{i,\min} = \min_{g_j}\{y_j\}$. And 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 $n \leq 10$, the values of ρ are produced in table 1. We use equation (5) for $n > 10$.

$$lim_{n->\infty}\ \rho = \frac{\sqrt{n}+1}{2} \tag{5}$$

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, load-dependent delay model* $d_{i,j}$ can be written as:

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

(6)

Theorem 1 The polynomial function corresponding to $d_{i,j}$ is a non-convex function of its variable $xnet_{j,max}$, $xnet_{j,min}$, $ynet_{j,max}$, $ynet_{j,min}$, 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 semi-definite throughout Ω [10]. For $d_{i,j}$ given in equation (6), the Hessian matrix is not guaranteed to be positive semi-definite. 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 with signal flow information. 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. Recall that we denote the number of vertices by n and the number of edges by e. 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 times for primary outputs are specified by the environment or the designer of the circuit. (Alternatively, the designer can specify a cycle time T that would then be satisfied by setting the input arrival times to zero and the output required times to T).

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 Global Problem Formulation

The simultaneous cell sizing and global placement problem can be formulated as:

$$\begin{array}{lll}
minimize & \mathrm{t}_{cycle} & \\
s.t. & a_j \geq a_i + d_{i,j} & \forall(v_i, v_j) \in A \\
 & a_j \leq T_{start} + \mathrm{t}_{cycle} & \forall v_j \in \mathrm{PO} \\
 & a_j \geq T_{start} & \forall v_j \in \mathrm{PI} \\
 & \dfrac{\sum_{i \in part_j} w_i \cdot x_i}{\sum_{i \in part_j} w_i} = xc_j & \mathrm{j} = 1, \ldots, \mathrm{L} \\
 & \dfrac{\sum_{i \in part_j} w_i \cdot y_i}{\sum_{i \in part_j} w_i} = yc_j & \mathrm{j} = 1, \ldots, \mathrm{L}
\end{array} \qquad (7)$$

where $part_j$ denotes the jth part, $w_i = f(z_i)$ is a *library function* relating area of g_i to its size z_i, and xc_j, yc_j are the geometric centers of $part_j$. The first three inequality constraints describe the timing relations, arrival and required time requirements in the circuit. The last two equality constraints describe the center of mass constraints imposed during the optimization in order to spread the cells evenly across the whole chip. The center of mass constraints are commonly used in placement programs that interleave quadratic programming with circuit bi-partitioning. Examples include Gordian [11], Speed [12]. In general L is the number of parts at the current partitioning step (L is a power of two due to recursive bi-partitioning step).

Theorem 2 Problem formulation (7) provides a correct statement of the simultaneous gate sizing and placement problem.

Proof The constraints of formulation (7) account for all the timing relations of the circuit and enforce even distribution across the *n* parts, so the solution produces a minimum cycle time for the circuit while satisfying the timing requirements and the center of mass constraints. ■

Theorem 3 Problem formulation (7) is a non-convex problem, which requires a non-convex programming algorithm to solve it.

Proof The timing constraint functions in (7) are polynomial functions, which are non-convex functions. So (7) is a non-convex problem, which requires a non-convex programming solver [15]. ■

Recall that *n* denotes the number of vertices (gates in the circuit), *e* denotes the number of edges, and *q* denotes the number of primary inputs and outputs in the circuit. There are four variables associated with each gate g_i in the circuit: x_i, y_i, z_i and a_i. So in total there are $4n$ variables in the formation (7). The number of constraints is $(e+q+2L)$.

Unfortunately, even for a small circuit (i.e., one with a few hundred cells), formulation (7) results in a non-linear optimization problem with a large number of variables and equations, which is too complex to be solved by standard mathematical programming solvers in any reasonable amount of time. Furthermore 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 due to changes in cell sizes in the subsequent optimization. To overcome these difficulties, we simplify (7) as is detailed next.

To reduce the problem complexity problem, we focus on optimizing the timing of *C(k)*. By iteratively finding and optimizing *C(k)*, the timing of the whole circuit can be improved gradually while the problem size remains manageable. We must however continue to address the congestion (or area balance) problem, which refers to the problem whereby certain regions of the chip are overpopulated by cells while other regions are underpopulated. Since we start with an existing "balanced placement" solution where all the cells are uniformly distributed across the parts, we can solve the congestion problem by restricting the change in sizes and locations of *C(k)* to relatively small ranges during each iteration. In this way, we ensure that the resulting placement and sizing solution is a only *perturbation* of the original balanced placement solution and therefore is not very unbalanced. Still, we apply a de-congestion step to create perfect balance after each optimization step. Without this de-congestion step, the perturbations may add up and after a few iterations, the placement solution may become completely unbalanced.

3 Optimizing *C(k)*

Throughout this section, we assume that an initial balanced placement and sizing of all gates is provided. Our goal is to iteratively improve the circuit timing through resizing and/or re-positioning of gates.

3.1 Iterative Optimization Problem Formulation

Consider a case where only sizes and locations of *C(k)* in (7) are variables of optimization and all other gates have fixed sizes and locations. Recall that *n'* denotes the number of gates in *C(k)*. Then we have *(3n'+n)* variables since there are *3n'* variables corresponding to *x*, *y*, and *z* values of the gates in *C(k)* and *n* variables corresponding to the arrival times of all gates. Notice that we ought to keep all of the variables for the arrival times because the arrival time of each gate in the circuits is related to the arrival times of the gates in its *transitive fan-in cone* and directly influences the arrival times of the gates in *its transitive fan-out cone*. Therefore, to capture the effect of any resizing or re-positioning of gates in *C(k)* on the arrival times of the circuit primary outputs (and hence on the minimum cycle time), we must do complete (although implicit) timing analysis during the optimization.

For example in figure 3, i_0, i_1, i_2, i_3 are primary inputs and o_0, o_1, o_2 are primary outputs. Assume i_1, g_1, g_2, g_3, g_4 and o_1 form the critical path, i.e., they are elements of *C(1)* which is the target of our. If the size and location of g_2 are changed, the capacitive loads of g_1, g_5, g_6 are changed while the arrival times of g_3, g_7 are changed. These changes in turn propagate through the rest of the circuit to all of the primary outputs. In other words the arrival times of all gates (except for primary inputs and g_{10}) are changed.

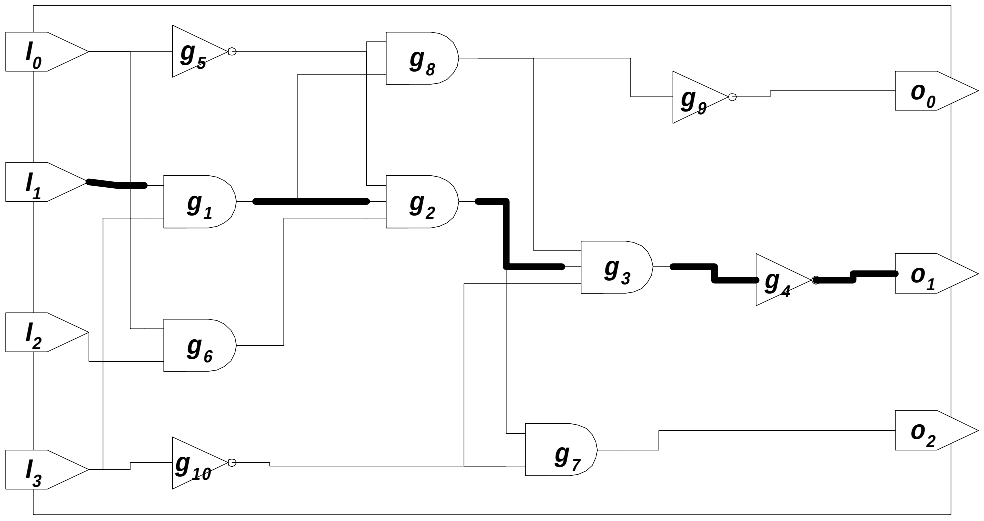


Figure 3 Circuit example to illustrate the need for keeping the arrival time variables.

The formulation of this problem is:

$$\begin{array}{lll} minimize & t_{cycle} & \\ s.t. & a_j \geq a_i + d_{i,j} & \forall (v_i, v_j) \in A \\ & a_j \leq T_{start} + t_{cycle} & \forall v_j \in PO \\ & a_j \geq T_{start} & \forall v_j \in PI \\ & \hat{x}_i^- \leq x_i \leq \hat{x}_i^+ & \forall v_i \in C(k) \\ & \hat{y}_i^- \leq y_i \leq \hat{y}_i^+ & \forall v_i \in C(k) \\ & \hat{z}_i^- \leq z_i \leq \hat{z}_i^+ & \forall v_i \in C(k) \end{array} \tag{8}$$

where $\hat{x}_i^-, \hat{y}_i^-, \hat{z}_i^-, \hat{x}_i^+, \hat{y}_i^+, \hat{z}_i^+$ are the lower and uppers bound on the location and size of g_i. They set the *variable change regions (VCRs).* These change regions are specified so that the resulting solution does *not* constitute a significant change to the original placed netlist. The optimization variables of formulation (8) are x_i, y_i, z_i of the *n'* gates in *C(k)* and the arrival times a_i of all the circuit gates. With this formulation, the number of variables is only reduced to *(n+3n')*; the number of constraints is now *(e+q+3n').*

Theorem 4 Problem formulation (8) provides a correct statement of the simultaneous gate re-sizing and re-positioning problem for *C(k).*

Proof The timing constraints of formulation (7) are all preserved, so the solution to the problem (8) minimizes the clock cycle time while accounting for all the timing relations in the circuit. ■

Theorem 5 Problem formulation (8) is a non-convex problem, which requires a non-convex programming algorithm to solve it.

Proof The proof is similar to that of Theorem 3. ■

A *VCR schedule* can be used so that in the early optimization iterations the *VCR*s are large whereas toward the end of the optimization loop, the *VCR*s become small. We do not dwell on this point any more since this is not the formulation that we will eventually use. The reason is that even though the number of variables is reduced greatly, the mathematical problem is still too large for a non-convex programming solver. So the problem has to be simplified further.

3.2 Simplified Problem Formulation

Suppose that the sizes and locations of *C(k)* are optimized within a *dynamically-controlled VCR* which can in turn guarantee that after the optimization, delay of no path outside *C(k)* can become larger than the delay of the current most critical path. We can then focus on optimizing *C(k)* and need not worry about the timing of other paths in the circuit. Consequently, the variables for the arrival time of the gates that are not in *C(k)* can be dropped from the formulation, and only the timing constraints on *C(k)* need to be taken into account. The trick however is in dynamically determining the range of values for *x*, *y*, *z* variables of the gates in *C(k)* to satisfy the above-mentioned requirement.

The new problem formulation is shown below:

$$\begin{array}{lll}
\textit{minimize} & t_{cycle} & \\
s.t. & a_j \geq a_i + d_{i,j} & \forall (v_i, v_j) \in A,\ v_i, v_j \in C(k) \\
& a_j \leq T_{start} + t_{cycle} & \forall v_j \in PO \text{ and } v_j \in C(k) \\
& a_j \geq T_{start} & \forall v_j \in PI \text{ and } v_j \in C(k) \\
& \hat{x}_i^- \leq x_i \leq \hat{x}_i^+ & \forall v_i \in C(k) \\
& \hat{y}_i^- \leq y_i \leq \hat{y}_i^+ & \forall v_i \in C(k) \\
& \hat{z}_i^- \leq z_i \leq \hat{z}_i^+ & \forall v_i \in C(k)
\end{array} \tag{9}$$

where $\hat{x}_i^-, \hat{y}_i^-, \hat{z}_i^-, \hat{x}_i^+, \hat{y}_i^+, \hat{z}_i^+$ are the lower and upper bounds on the location coordinates and size of g_i. They set the *dynamic variable change regions (DVCRs).*

Theorem 6 If $\hat{x}_i^-, \hat{y}_i^-, \hat{z}_i^-, \hat{x}_i^+, \hat{y}_i^+, \hat{z}_i^+$ in problem formulation (9) can be set correctly so as to guarantee that the changes in *C(k)* do not increase the delay of any path outside of *C(k)* beyond that of the current most critical path, then problem formulation (9) provides a correct statement of the simultaneous gate re-sizing and re-positioning problem for *C(k)*.

Proof $\hat{x}_i^-, \hat{y}_i^-, \hat{z}_i^-, \hat{x}_i^+, \hat{y}_i^+, \hat{z}_i^+$ enforce the timing constraints outside of *C(k),* and the arrival time equations of *C(k)* keeps the timing constraints in *C(k).* So the solution of problem formulation (9) satisfies all the timing requirements. ■

Theorem 7 Problem formulation (9) is a non-convex problem, which requires a non-convex programming algorithm to solve it.

Proof Similar to the proof of Theorem 3. ■

Theorem 8 Problem formulation (9) is a generalized geometric programming problem.

Proof Both the objective function and the constraint equations are polynomial functions, so (9) is a generalized geometric programming problem (please refer to section 6 for details). ■

Notice that in this formulation, there are only *4n'* variables, which correspond to the x_i and y_i coordinates, the sizing variable z_i, and the arrival time a_i of the *n'* gates in *C(k)*; There are only *(3n'+q'+e')* constraints where *q'* is the number of primary inputs and outputs and *e'* is the number of edges within *C(k)*; the *3n'* constraints are due to the *DVCR* constraints. Consequently by controlling the size of *C(k)*, i.e., the number of critical paths being optimized simultaneously, the problem size can be made quite manageable. In addition, congestion problem is also addressed since $\hat{x}_i^-, \hat{y}_i^-, \hat{x}_i^+, \hat{y}_i^+$ limit the gates movement. No serious congestion can occur if these variable ranges are set appropriately.

The main task now is to set *DVCRs* so as to satisfy the condition of theorem (6). More precisely, the solution to problem formulation (9) should make use of the *slack time* of the gates on the non-critical paths to optimize the gates on the critical paths while not allowing the delay of any non-critical path to exceed that of the current most-critical path. The latter requirement (which is precisely the condition of theorem (6)) is essential in achieving a *monotonically reducing objective function value* in this iterative optimization process. Notice also that the slack time for all gates on the most critical path is normalized to zero. All gates on the non-critical paths will therefore have positive slack times.

Exact calculation of *DVCRs* is a difficult task itself. We adopt a heuristic technique to do the calculation as explained next. We start from two simple cases and then generalize to the common case: a) there is only one gate g_i to be repositioned: $\hat{x}_i^-, \hat{y}_i^-, \hat{x}_i^+, \hat{y}_i^+$ are to be determined, all the others are 0; b) there is only one gate g_i to be resized: $\hat{z}_i^-, \hat{z}_i^+$ are to be determined, all the others are 0.

We point out at this time that all of the timing constraints for the paths included in *C(k)* are explicitly accounted for in the formulation (9) and hence we need to pay attention only to the inputs and outputs of *C(k)* which are not in *C(k)* in order to derive the *DVCR*s of gates in *C(k)*. This is an important observation.

3.2.1 Location change region of a single gate

A change in the position of g_i, which is on the critical path, influences the arrival time of its *immediate fan-in* gates because such a change will affect the routing length of its fan-in nets. In figure 4, g_p, g_i, g_q form a critical path (shown in thicker lines), and g_j, g_k and g_l are off-critical-path fan-in and fan-out of g_i .

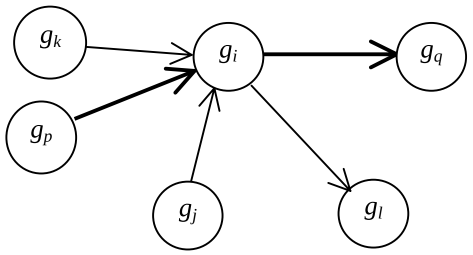


Figure 4 Example to illustrate fan-in and fan-out induced *DVCRs*.

From equation (6), the change in the arrival time of g_j (denoted by Δt_j) as a result of a change in the bounding box of the output net of g_j (denoted by $\Delta xnet_j$, $\Delta ynet_j$) is given by:

$$\Delta t_j = \rho \cdot rdr_j \cdot \{ C_{hor} \cdot \Delta xnet_j + C_{ver} \cdot \Delta ynet_j \} + \rho \cdot \{ R_{hor} \cdot \Delta xnet_j + R_{ver} \cdot \Delta ynet_j \} \cdot cload_j \tag{10}$$

Here only g_i is movable, and since the sizes of g_j and all its fan-outs (including g_i itself) are fixed, their current sizes can be used. Notice that $rdr_j = max\{rdr_{u,j}\}$ for the worst-case analysis.

When using a *net bounding box model*, a change in the coordinates of a gate may not change the bounding box of the input net that is connected to the gate. This occurs, for example, when the driver of the input net is driving more than one gate (as is the case in Figure 5). In this figure, as long as g_i is moved in the bounding box determined by g_j, g_s, g_t, the net load of g_j will not change, therefore the arrival time of g_j will not change. Similarly, a change in the coordinates of g_i may not change the bounding box of its output net (not shown in the figure).

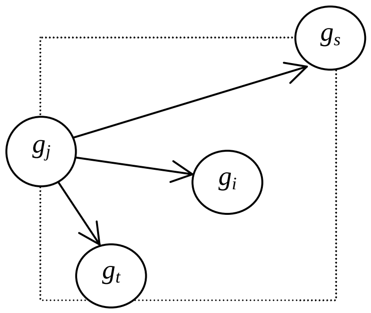


Figure 5. Example to illustrate the effect of the input net bounding box on the *DVCRs* of a cell.

From (10) and the above discussion, we calculate the *fan-in induced DVCR of g_i from input g_j* as follows:

$$\begin{aligned}
FI\hat{x}^{-}_{j,i}(s_j) &= xnet_{j,min} - \frac{s_j}{\rho \cdot rdr_j \cdot \{ C_{hor} + C_{ver} \} + \rho \cdot \{ R_{hor} + R_{ver} \} \cdot cload_j} \\
FI\hat{y}^{-}_{j,i}(s_j) &= ynet_{j,min} - \frac{s_j}{\rho \cdot rdr_j \cdot \{ C_{hor} + C_{ver} \} + \rho \cdot \{ R_{hor} + R_{ver} \} \cdot cload_j} \\
FI\hat{x}^{+}_{j,i}(s_j) &= xnet_{j,max} + \frac{s_j}{\rho \cdot rdr_j \cdot \{ C_{hor} + C_{ver} \} + \rho \cdot \{ R_{hor} + R_{ver} \} \cdot cload_j} \\
FI\hat{y}^{+}_{j,i}(s_j) &= ynet_{j,max} + \frac{s_j}{\rho \cdot rdr_j \cdot \{ C_{hor} + C_{ver} \} + \rho \cdot \{ R_{hor} + R_{ver} \} \cdot cload_j}
\end{aligned} \tag{11}$$

where s_j is the allowed range of change for the arrival time of g_j i.e., its timing slack. $xnet_{j,min}$, $ynet_{j,min}$, $xnet_{j,max}$, $ynet_{j,max}$ is the current bounding box of g_j's output net. Note that we have divided the slack equally in the *x* and *y* directions.

So if g_i moves in the rectangle defined by (11), the arrival time of g_j would not be any later than its current arrival time plus s_j, according to bounding box model. Similarly, we calculate the *fan-in induced DVCR of* g_i *from input* g_k: $FI\hat{x}^-_{k,i}(s_k), FI\hat{y}^-_{k,i}(s_k), FI\hat{x}^+_{k,i}(s_k), FI\hat{y}^+_{k,i}(s_k)$. Obviously, to satisfy the timing requirements of both fan-ins, g_i's movement should be limited by the intersection of $FI\hat{x}^-_{j,i}(s_j), FI\hat{y}^-_{j,i}(s_j), FI\hat{x}^+_{j,i}(s_j), FI\hat{y}^+_{j,i}(s_j)$ and $FI\hat{x}^-_{k,i}(s_k), FI\hat{y}^-_{k,i}(s_k), FI\hat{x}^+_{k,i}(s_k), FI\hat{y}^+_{k,i}(s_k)$.

In general, we calculate the *fan-in induced DVCR* of g_i as follows:

$$\begin{aligned}
FI\hat{x}^-_i &= \max_j(FI\hat{x}^-_{j,i}(s_j))\\
FI\hat{y}^-_i &= \max_j(FI\hat{y}^-_{j,i}(s_j))\\
FI\hat{x}^+_i &= \min_j(FI\hat{x}^+_{j,i}(s_j))\\
FI\hat{y}^+_i &= \min_j(FI\hat{y}^+_{j,i}(s_j))
\end{aligned}$$

Notice that the intersection maybe a polygon, it is simplified to a minimum size rectangle for computational efficiency.

Similarly, a change in the position of g_i influences the arrival time of its fan-outs. The change in the arrival time of g_k (denoted by Δt_k) as a result of change in the bounding box of the output net of g_I (denoted by $\Delta xnet_i$, $\Delta ynet_i$) is given by:

$$\Delta t_l = \rho \cdot rdr_i \cdot \{C_{hor} \cdot \Delta xnet_i + C_{ver} \cdot \Delta ynet_i\} + \rho \cdot \{R_{hor} \cdot \Delta xnet_i + R_{ver} \cdot \Delta ynet_i\} \cdot cload_i \qquad (12)$$

Here only the bounding box of g_i's output net needs to be considered. We calculate a *fan-out induced DVCR* of g_i directly:

$$\begin{aligned}
FO\hat{x}^-_i &= xnet_{i,min} - \frac{s_l}{\rho \cdot rdr_i \cdot \{C_{hor} + C_{ver}\} + \rho \cdot \{R_{hor} + R_{ver}\} \cdot cload_i}\\
FI\hat{y}^-_i &= ynet_{i,min} - \frac{s_l}{\rho \cdot rdr_i \cdot \{C_{hor} + C_{ver}\} + \rho \cdot \{R_{hor} + R_{ver}\} \cdot cload_i}\\
FO\hat{x}^+_i &= xnet_{i,max} + \frac{s_l}{\rho \cdot rdr_i \cdot \{C_{hor} + C_{ver}\} + \rho \cdot \{R_{hor} + R_{ver}\} \cdot cload_i}\\
FI\hat{y}^+_i &= ynet_{i,max} + \frac{s_l}{\rho \cdot rdr_i \cdot \{C_{hor} + C_{ver}\} + \rho \cdot \{R_{hor} + R_{ver}\} \cdot cload_i}
\end{aligned} \qquad (13)$$

Notice that in equation (13), s_l is calculated as the minimum slack time of any non-critical output of g_i.

Subsequently, the *DVCR* of g_i is the intersection of *fan-in induced* and *fan-out induced DVCRs* of g_i:

$$\hat{x}_i^- = max(FI\hat{x}_i^-, FO\hat{x}_i^-)$$
$$\hat{y}_i^- = max(FI\hat{y}_i^-, FO\hat{y}_i^-)$$
$$\hat{x}_i^+ = min(FI\hat{x}_i^+, FO\hat{x}_i^+)$$
$$\hat{y}_i^+ = min(FI\hat{y}_i^+, FO\hat{y}_i^+)$$

In the example of figure 6, the dotted rectangle is the move *DVCR* of g_2, which is the intersection of (11) applied to non-critical fan-in gates g_5 and g_6 and (13) applied to non-critical fan-out gates g_7.

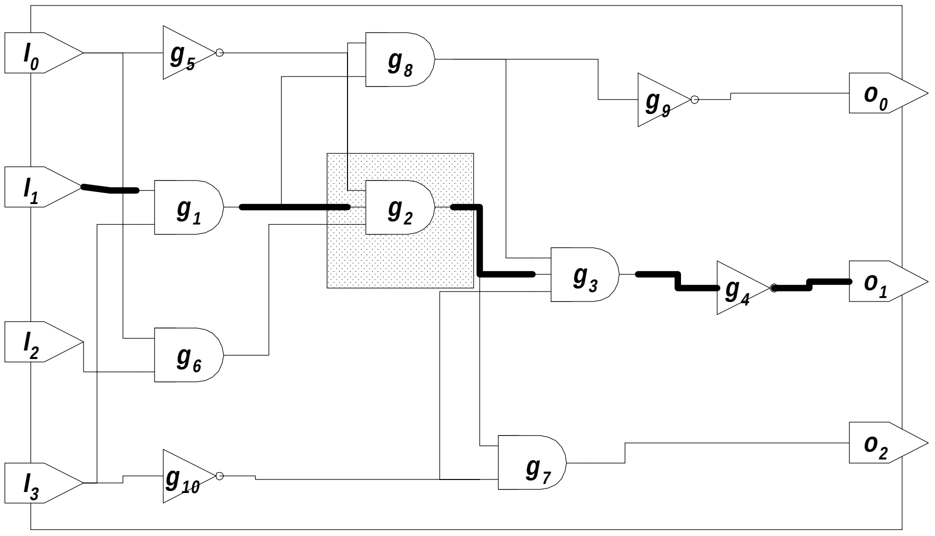


Figure 6. Example for the calculation of $\Delta x_2, \Delta y_2, \Delta z_2$.

3.2.2 Size change region of a single gate

We consider here the case where only the size of g_i, which is on the critical path, is changeable. First, the slack times of the off-critical-path fan-ins of g_i set an upper bound on the size of g_i. As in section 3.2.1, based on equation (6), we write equations to transform the slack times of fan-ins to $\hat{z}_i^+$:

$$\hat{z}_i^+ = \hat{z}_i + \min_j\left(\frac{s_j}{\{rdr_j + rnet_j\} \cdot \alpha 3_{j,i}}\right) \qquad (14)$$

where $\hat{z}_i$ is g_i's current size, g_j is g_i's fan-in, $rdr_j = max\{rdr_{u,j}\}$ for worst-case analysis, and s_j is the (available) slack time of g_j.

Similarly, the slack times of the off-critical-path fan-outs of g_i set a lower bound on the size of g_i. We obtain $\hat{z}_i^-$ by finding the positive root of the following quadratic equation (where w_l is the variable):

$$\hat{z}_i^- = \max_l (\hat{z}_i - w_l)$$
$$s_l = w_l \cdot \alpha 1_i + (\frac{\alpha 2_i}{z_i + w_l} - \frac{\alpha 2_i}{z_i}) \cdot (cnet_i + cload_i) \tag{15}$$

where $\alpha 1_i = max\{\alpha 1_{j,i}\}$, $\alpha 2_i = max\{\alpha 2_{j,i}\}$ for worst-case analysis, and $\hat{z}_i$ is the current size of g_i. s_l is the (available) slack time s_l of g_l, and g_l is g_i's fan-out.

3.2.3 Location and size change regions of all gates in *C(k)*

In practice, we would like to change the locations and sizes of all the gates in *C(k)* simultaneously. So the slack time of the fan-ins and fan-outs need to be allocated between position and sizing parameters. Furthermore, since the area congestion problem should also be considered, one must impose maximum gate move values (Δx and Δy) to ensure that the next placement solution is not very different from the current placement solution. For each gate in *C(k)*, $\hat{x}_i^-, \hat{y}_i^-, \hat{x}_i^+, \hat{y}_i^+$ are first calculated as in section 3.2.1, then

$$\hat{x}_i^- = max(\hat{x}_i^-, \hat{x}_i - \Delta x)$$
$$\hat{y}_i^- = max(\hat{y}_i^-, \hat{y}_i - \Delta y)$$
$$\hat{x}_i^+ = min(\hat{x}_i^+, \hat{x}_i + \Delta x)$$
$$\hat{y}_i^+ = min(\hat{y}_i^+, \hat{y}_i + \Delta y)$$

where $\hat{x}_i, \hat{y}$ are the current position coordinates of g_i. Next, for each fan-in and fan-out gate of *C(k)*, if the gate has available slack time after considering the effect of moving gates in *C(k)*, then the available slack times are used to set $\hat{z}_i^-, \hat{z}_i^+$ as in section 3.2.2.

Since the fan-ins and fan-outs of *C(k)* may share the slack time of some common path, the location and size change regions may indeed be correlated. Therefore, if we use the upper and lower bounds on the change regions as above, we may overestimate the *DVCRs* by neglecting the correlations between the change regions of different gates in *C(k)*. In practice, we rely on a user-defined parameter μ ($0<\mu<1$) to uniformly scale down the *DVCRs*. The exact value of μ is determined from the size of circuit and how closely the paths are related. At the beginning of the optimization loop, the timing slacks are large and we are more tolerant of errors in *DVCR* calculation due to the path correlation effects, μ is thus set close to 1. As the iterative optimization process progresses, μ is decreased gradually (and in this case, linearly) to keep the solution convergent.

In figure 8, the dotted rectangles show the move *DVCR* of the gate in *C(k)*, the maximum size are controlled by Δx and Δy. The cross-lined cycles show the size *DVCR* of gates in *C(k)*. g_3's size is fixed in this figure.

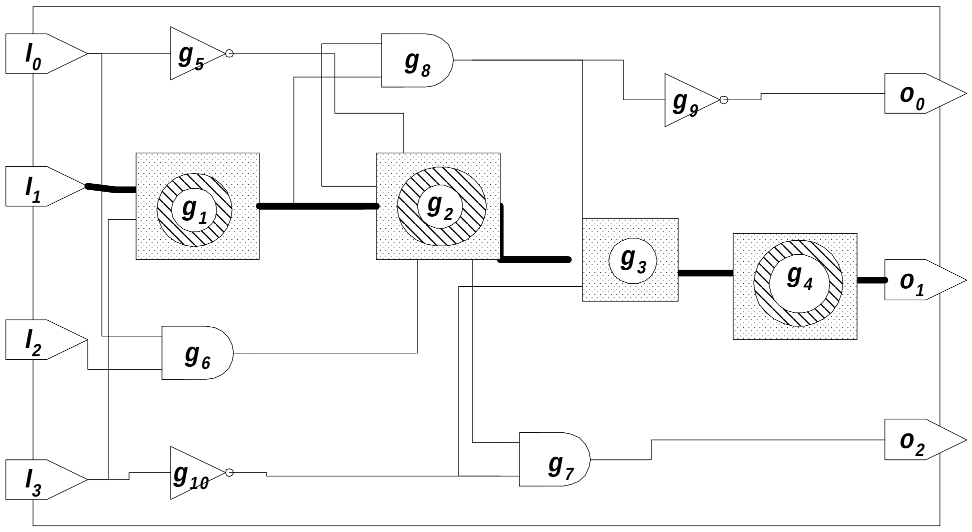


Figure 8. Example to illustrate the position and size *DVCRs* .

3.3 Decongestion

Here no non-overlapping constraint is imposed in the formulation of (9). 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 (9) is solved, the size and ideal location of every cell is determined. Initially each cell is assigned to the row that is the closet to its ideal location[1]. Cells in the same row are placed in 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. the longest row and its shorter adjacent row) 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.

4 Optimizing the Neighborhood of *C(K)*

4.1 *Ne(k,1)* Re-placement

Notice that to reduce the delay of a certain cell, not only can we size up the cell and move the cell closer to its fan-outs, but also we can size down its fan-out cells or pull its fan-out cells closer to reduce its load. So to improve the timing of the critical paths more, the capacitance load imposed on *C(k)* by the corresponding

[1] We assume row-based layout.

Ne(k,i) should be considered too. (In this paper, only *Ne(k,1)* is sized down and re-placed to optimize the timing property of the critical paths).

As the example in figure 9, g_7 and g_8 are *Ne(k,1)* and g_9 is *NC(2)*. g_6 is the fan-in of g_2 and fan-out of two uncritical primary inputs, so g_6 is not considered. So are g_5 and g_{10}. The delay $d_{1,2}$ may be decreased by moving g_8 closer to g_1 or by sizing down g_8. However these changes may increase $d_{5,8}$ so that the delay through i_0, g_5, g_8, g_9 and o_0 may become even larger than the delay of the current critical path i_1, g_1, g_2, g_3, g_4, and o_1.

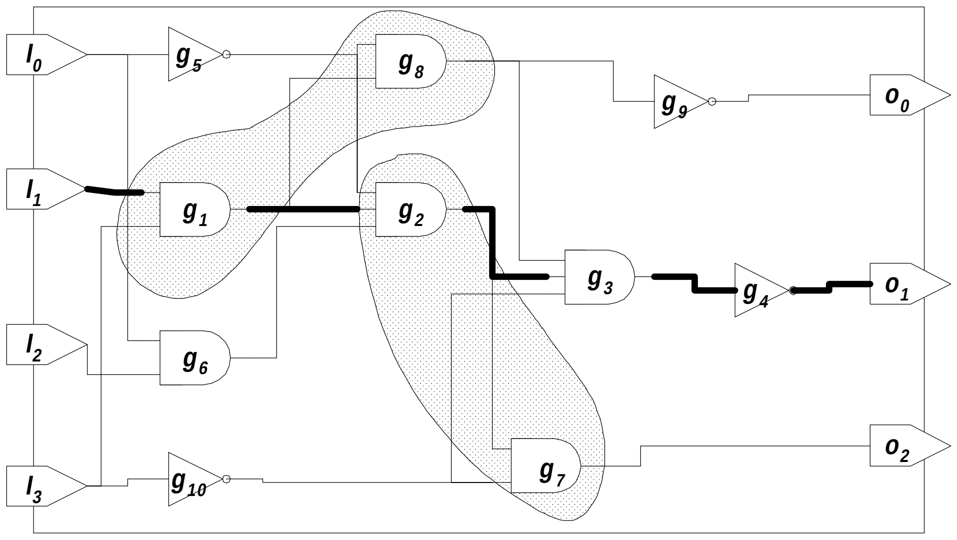


Figure 9. Circuit example for *Ne(k,1)* re-placement.

To size down and place *Ne(k,1)* optimally without violating timing constraints, we should change the locations and sizes of *Ne(k,1)* at the same time. But as specified before, gate sizing and placement simultaneously is a non-convex problem. If the locations and sizes of *Ne(k,1)* are all formulated as variables in our problem, either the number of the gates in *Ne(k,1)* or the number of the paths which pass through *Ne(k,1)* is often much larger than that of *C(k)*, the problem size would become unmanageable. So in our approach, we optimize *Ne(k,1)* separately: a) *Ne(k,1)* re-placement and b) *Ne(k,1)* resizing. As for the example of Figure 9, we do g_7, g_8 re-placement; g_7, g_8 resizing.

In this section, *Ne(k,1)* re-placement is discussed. The mathematical formulation is as following. Here only the locations of *Ne(k,1)* are changeable.:

$$\begin{array}{lll} \textit{minimize} & t_{cycle} & \\ s.t. & a_j \geq a_i + d_{i,j} & \forall(v_i, v_j) \in A \\ & a_j \leq T_{start} + t_{cycle} & \forall v_j \in PO \\ & a_j \geq T_{start} & \forall v_j \in PI \\ & | x_i - \hat{x}_i | \leq \Delta x & \forall v_i \in Ne(k,1) \\ & | y_i - \hat{y}_i | \leq \Delta y & \forall v_i \in Ne(k,1) \end{array} \qquad (16)$$

where $\hat{x}_i, \hat{y}_i$ are the location coordinates of g_i from the previous iteration; Δx, Δy are the *position VCRs*. Since all the timing constraints are kept here so that we do not need to worry about the timing violation. The reason for having *position VCRs* is that the area congestion has to be controlled. Without Δx, Δy all the gates may be attracted to the center of the chip. Notice however that these *VCR*s are uniformly and statically defined as Δx, Δy in section 3.2.3 and are not based on the available slack times of the side inputs and outputs of *Ne(1,k)*. A *VCR* schedule is however used whereby as the iterative optimization process progresses the *VCR*s are reduced from one iteration to next so as the convergence is achieved.

Theorem 9 Formulation (16) is a Linear Programming problem.

Proof Refer to the delay model (6), since no size variables, delay becomes a linear function. So (16) is a Linear Programming problem. ■

Similar to the discussion in Section 3.1, although only *Ne(k,1)* are movable, the timing of the whole circuit must be considered. So there are *(n+2n")* variables in (16), where *n"* is the number of the gates in *Ne(k,1)*. There are *(e+q+2n")* constraints. Although the number of variables and constraints maybe large, this is not a major concern since a Linear Programming problem can be solved very efficiently. We use LP-Solver of [13] to solve (16). Notice however that to improve the runtime of the LP solver, problem formulation (16) can be approximated by using a similar transformation to that which was used to obtain problem formulation (9) from problem formulation (8). In practice, we do not do this since the LP solver can handle problem formulation (16) directly.

There may be cell congestion problem after (16) is solved; the decongestion step described in Section 3.3 is therefore applied at the end of this optimization step.

4.2 *Ne(k,1)* Re-sizing

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

$$
\begin{array}{llll}
minimize & t_{cycle} & & \\
s.t. & a_j \geq a_i + d_{i,j} & \forall (v_i, v_j) \in A & \\
 & a_j \leq T_{start} + t_{cycle} & \forall v_j \in PO & \\
 & a_j \geq T_{start} & \forall v_j \in PI &
\end{array}
\qquad (17)
$$

Notice there is no size change region constraint, since all the timing relations are formulated in constraints, and in-place sizing would not incur serious congstion problem.

Theorem 10 Formulation (17) is a Geometric Programming problem.

Proof Refer to the delay model (6), the wire length is known now. Delay is a posynomial funciton. So (17) is a Linear Programming problem. (Refer to Section 6) ■

Similar to the discussion in Section 3.1, although only *Ne(k,1)* are resized, the timing of the whole circuit must be considered. So there are *(n+n")* variables in (17), where *n"* is the number of the gates in *Ne(k,1).* There are *(e+q)* constraints. Although the number of variables and constraints maybe large, again this is not a major concern since a Geometric Programming problem can be solved very efficiently. By using the variable substitution *ln(z)*=*w*, (17) is transformed to a Linear Programming (LP) problem. GP is described briefly in Section 6. Notice again that to improve the runtime of the GP solver, problem formulation (17) can be approximated by using a similar transformation to that which was used to obtain problem formulation (9) from problem formulation (8). In practice, we do not do this since the LP solver can handle problem formulation (17) directly.

5 Optimization Flow

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 or it converges very slowly. To address this problem, we introduce a cooling schedule to control the variable freedom. As the iteration count increases, μ decreases, so Δx, Δy, Δz_i all decrease. Finally the freedom becomes so little 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 schedule can be used to generate a higher quality result.

5.1 Selection of Discrete Gate Sizes

After solving equations (17) and (9) gate sizes are given as real numbers which will likely not match the given gate sizes in the ASIC library. Therefore at the end of these optimizations, we need to round the size of each gate to the closet size 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.

Next consider the problem of discrete gate sizing for minimum delay along the set of paths in *Ne(k,1)* (for Equation (17)) or *C(k)* (for Equation (9)) 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.

5.2 Algorithm Flow

The main loop of this algorithm consists of three parts: (a) *Ne(k,1)* re-placement; (b) *Ne(k,1)* sizing down; and (c) simultaneous *C(k)* sizing and placement. These three steps are all done locally and somewhat independently. The order among these three steps may be changed. Considering that the step (c) which does the simultaneous sizing and placement of *C(k)* optimizes the timing of *C(k)* directly and tunes both the sizes and locations of the gates of *C(k)*. It is the most effective and the most elaborate step of our optimization flows. To make use of its optimization capability completely and correctly, it should be done when all the other cells' sizes and locations are known and accurate. So simultaneous *C(k)* sizing and placement is done last.

The purpose of *Ne(k,1)* sizing down and *Ne(k,1)* re-placement is to make use of the slack time of *Ne(k,1)* to optimize *C(k)* without creating new critical paths. *Ne(k,1)* re-placement may cause the congestion problem. During *Ne(k,1)* re-placement the higher the use of slack of *Ne(k,1)*, the more serious the congestion problem. *Ne(k,1)* sizing in general has more potential to improve the overall *C(k)* timing while creating less congestion problem. So we would like to use more of slack time of *Ne(k,1)* during the *Ne(k,1)* sizing, that can be achieved by setting a small Δx_i, Δy_i in *Ne(k,1)* re-placement. And do *Ne(k,1)* re-placement first to provide correct cell locations for *Ne(k,1)* sizing.

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 gate selection method [14] is used to convert the continuous gate size to discrete available gate size.

The complete flow of this algorithm is as shown below:

```
1.  begin
2.  timing-driven initial placement
3.  timing analysis to select C(k)
```

4. *adjustment of Δx, Δy for Ne(k,1)*
5. *Ne(k,1) re-placement*
6. *de-congestion*
7. *Ne(k,1) sizing*
8. *gate size selection from the cell library*
9. *calculation of* $\hat{x}_i^-, \hat{y}_i^-, \hat{z}_i^-, \hat{x}_i^+, \hat{y}_i^+, \hat{z}_i^+$
10. *simultaneous C(k) sizing & placement*
11. *gate size selection from the cell library*
12. *de-congestion*
13. *if (a) there is improvement in the last t iterations or specification not satisfied, and (b) the problem is solvable, go to 2)*
14. *end*

Figure 10. Algorithm flow.

6 GP and GGP

Since our problem formulation is in the form of a polynomial function, GP and GGP can be used to solve them. In this section, we will describe the GP and GGP approach. The GP problem can be solved efficiently by the infeasible interior-point method of [16]. To solve a GGP problem, the original GGP is transformed into a sequence of GP problems by a process commonly referred to as condensation [18].

6.1 Background

A function *u(x)* is *monomial*, if it is of the following form:

$$u(x) = b\prod_{i=1}^{n} x_i^{a_i}$$

where x = ($x_1, \ldots\ldots x_n$) are strictly positive in value. The exponent a_i and the coefficient *c* are real constants. a_i is unrestricted in sign while *b* is required to be positive.

A function *p(x)* is *posynomial* if it is of the following form:

$$p(x) = \sum_i c_i u_i(x)$$

where $u_i(x)$ is monomial and the coefficients c_i is positive.

A function g*(x)* is *polynomial* if it is of the following form:

$$g(x) = \sum_i c_i u_i(x)$$

where $u_i(x)$ is monomial and the coefficients c_i are not restricted in sign.

Definition *Geometric Programming* (GP) is a class of nonlinear optimization problems having objective functions and constraint functions expressed as *posynomials*. A GP problem with *n* variables and *m* constraints is as follows:

$$\begin{array}{lll} \min imize & p_0(x) & \\ s.t. & p_k(x) \leq 0, & k = 1,2,....,m \end{array}$$

where $p_0, p_1, \ldots\ldots, p_k$ are posynomial functions, $x=\{x_1, x_2, \ldots x_n\}$.

Definition *Generalized Geometric Programming* (GGP) is a class of nonlinear optimization problems having objective functions and constraint functions expressed as *polynomials*. A GGP problem with *n* variables and *m* constraints is as follows:

$$\begin{array}{lll} \min imize & g_0(x) & \\ s.t. & g_k(x) \leq 0, & k = 1,2,....,m \end{array}$$

where $g_0, g_1, \ldots\ldots, g_k$ are *polynomial* functions, $x=\{x_1, x_2, \ldots x_n\}$.

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

6.2 Geometric Programming (GP)

By using the variable substitution *ln(x)=w*, GP can be transformed to a linear programming problem. There are many algorithms to solve a GP problem. We use the method of [16]. The approach is by means of a primal-dual algorithm developed simultaneously for (i), the dual geometric program after logarithmic transformation of its objective function and (ii), its Lagrangian dual program. Under rather general assumptions, the mechanism defines a primal-dual infeasible path from a specially constructed, perturbed Karush-Kuhn-Tucker (KKT) system. Subfeasible solutions are generated for each program whose primal and dual objective function values converges to the respective primal and dual program values. The basic technique is one of a predictor-corrector type involving Newton's method applied to the perturbed KKT system, coupled with effective techniques for choosing iterate directions and step length. Sophisticated implementation techniques and advanced sparse matrix factorization are used to take advantage of the very special structure of the Hessian matrix of the logarithmically transformed dual objective function.

Our computational results indicate that this GP algorithm leads to an efficient and stable implementation for solving our problem.

6.3 GGP Condensation

To solve the GGP problem, we implement the algorithm described in [17]. This algorithm takes advantage of the arithmetic-geometric mean inequality and transforms the original non-convex GGP to a sequence of convex GPs.

The GGP algorithm first introduces a new variable. The original nonlinear objective function is absorbed as an additional constraint. So that the objective function becomes linear:

$$\begin{array}{lll} \text{minimize} & x_0 & \\ \text{s.t.} & g_0(x) \le x_0 & \\ & g_k(x) \le 0, & k = 1,2,\ldots,m \end{array}$$

where $g_0, g_1, \ldots\ldots, g_k$ are polynomial functions.

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

$$\begin{array}{lll} \text{minimize} & x_0 & \\ \text{s.t.} & p_0^+(x) - p_0^-(x) \le x_0 & \\ & p_k^+(x) - p_k^-(x) \le 0 & k = 1,2,\ldots,m \end{array}$$

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 then divided through to yield a quotient form:

$$\begin{array}{lll} \text{minimize} & x_0 & \\ \text{s.t.} & \dfrac{p_0^+(x)}{p_0^-(x) + x_0} \le 1 & \\ & \dfrac{p_k^+(x)}{p_k^-(x)} \le 1 & k = 0,1,\ldots m, \end{array}$$

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

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.

$$\sum_i u_i \geq \prod_i (\frac{u_i}{\delta_i})^{\delta_i}$$

where u_i is positive value, the δ_i is positive weight and $\sum\delta_i$=1. When applied to a posynomial, the A-G inequality converts the posynomial into an approximating monomial. The 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':

$$\delta_i = \frac{u_i(x')}{p(x')}$$

It can be seen that all u_i/δ_i are equal when u_i is evaluated at the operating point.

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

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

Condensing the denominator of each constraint 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.

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

Since the inequality relation 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.

The 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 the corresponding GP is solved. The solution to the GP is used to condense the GGP at the next iteration.

Theorem 11 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].

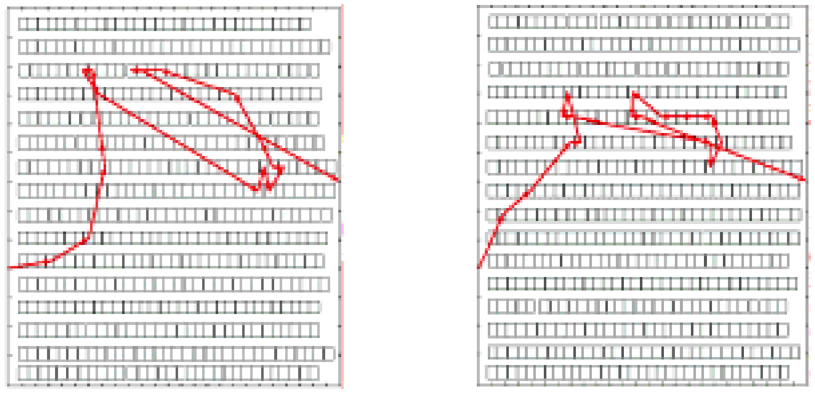
This algorithm requires a feasible initial solution at the beginning. For our problem, any initial placement of a mapped netlist forms a feasible solution[2].

7 Experimental Results

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

7.1 SCD in Action

The following are some snapshots of the placement and sizing results of SCD during the optimization of the benchmark circuit C499. In this example, at the beginning μ is set to be 0.8. The initial size of all the gates is 1. Figure 11 is the result of one optimization iteration using the above change region 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 11. Particularly, the path length is much shorter in (2) compared to (1).



(1) path delay: 12.43 (2) path delay:12.02

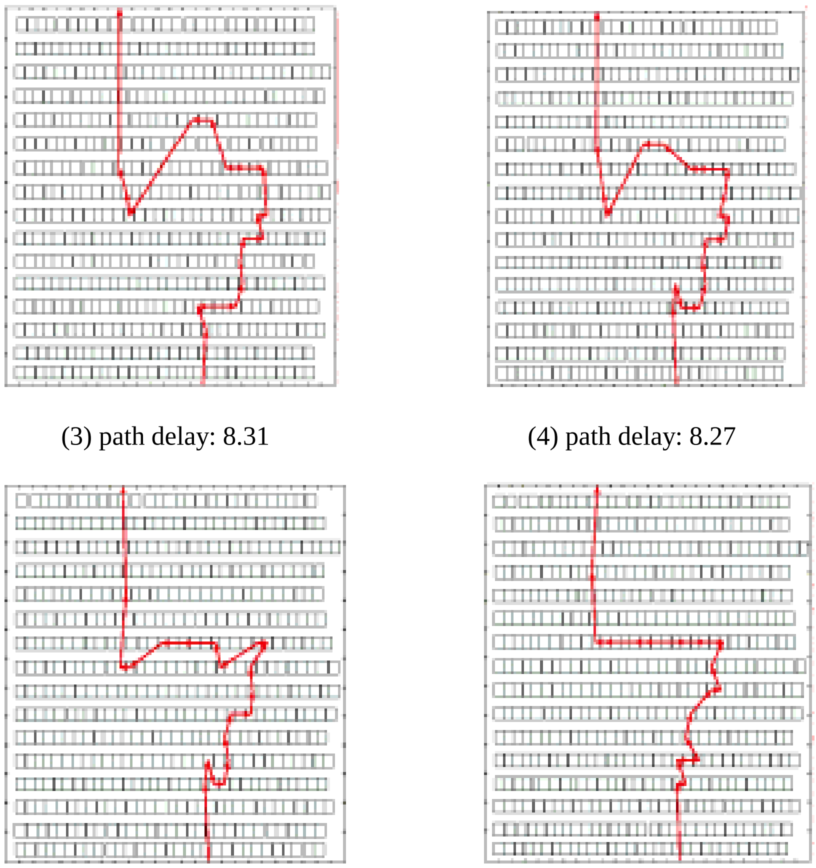
Figure 11. Result of iteration with large change regions.

After a number of iterations the cell freedom is reduced. It may take several iterations to optimize the most critical path until another path becomes the most critical. Figure 12 shows 3 consecutive iterations to optimize the same path of C499. Here μ is set to 0.4. We can see that as a result of successive iterations the

[2] Of course, we are well-advised to start with a timing-driven placement result and a timing-driven technology mapped circuit.

path becomes more and more straight. However the change in path layout is less dramatic than that seen in Figure 11 because of small variable change ranges.

We next calculate the cell slacks for a required arrival 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 and a normalized slack of 1 means can never be reached (means zero delay path exist). In Figure 13, 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 achieve improved timing by balancing the path delays, i.e. longer delay paths get shorter as the expense of shorter delay path getting longer.


(3) path delay: 8.31

(4) path delay: 8.27

(5) path delay: 8.22

(6) path delay: 8.17

Figure 12. Results of multiple iterations with small change regions.

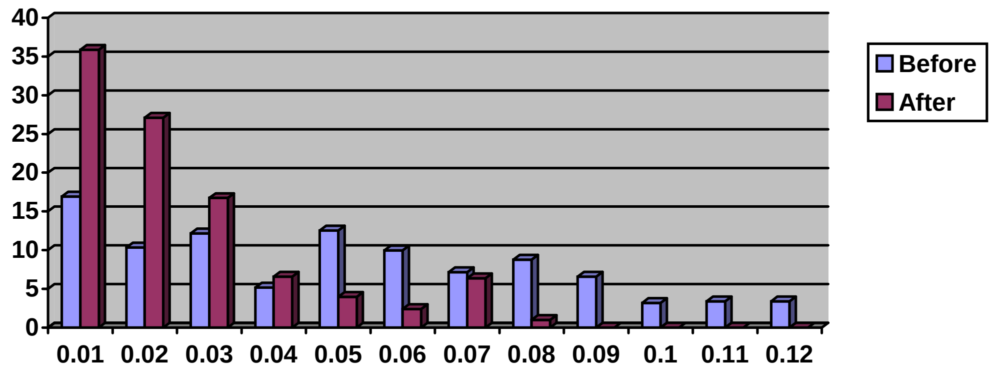


Figure 13. Distribution of the normalized slack time before and after optimization.

7.2 Benchmark Results

Our algorithm has been applied to ISCAS benchmark circuits mapped to a 0.35μ industrial 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 the cardinality of *C(k)*≤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.) As expected, the SCD runtime is higher than that of IPS. However the timing improvement justifies the increased runtime. Programs are run on Pentium II 300; the SCD runtimes are range from one minute for the smallest circuit to under an hour for the largest circuit.

Chip	Cell	Level	Delay of TW Place-ment	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)	Improve-ment (%) over IPS
C432	215	31	20.90	9.42	299	50	8.56	303	584	9.1
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
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
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	470	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	2326	14.3
C7552	2156	55	45.72	20.12	3169	201	16.90	3241	5349	16.0
des	3059	29	22.49	11.50	4238	510	9.40	4302	25023	18.2

Table 2. Experimental results.

All results are reported after detailed placement and detailed routing using YACR. The delays include the gate delay and post-layout interconnects delays.

8 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 recurrent techniques with cell placement.

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.