Delay and Area Optimization for Compact Placement by Gate Resizing and Relocation

Weitong Chuang
AT&T Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974

Ibrahim N. Hajj
Dept. of Electrical & Computer Eng.
and Coordinated Science Lab.
University of Illinois

Abstract

In this paper, we first present an efficient algorithm for the gate sizing problem. Then we propose an algorithm which performs delay and area optimization for a given compact placement by resizing and relocating cells in the circuit layout. Since the gate sizing procedure is embedded within the placement adjustment process, interconnect capacitance information is included in the gate size selection process. As a result, the algorithm is able to obtain superior solutions.

1 Introduction

A standard cell library typically contains several versions of any given gate type, each of which has a different gate size with different driving capacity. The gate sizing problem is that of choosing optimal gate sizes from the library to minimize a cost function (such as total circuit area), while meeting the timing constraints. This is usually done after technology mapping, where the logic function of each gate is determined, and before placement. A drawback of such an approach is that accurate interconnect wire lengths are not available during the gate sizing procedure. The gate size selected optimally at that stage may no longer be optimal after the physical design stage where large interconnect capacitances are introduced at the output of each gate. This problem is emerging as the size of today's VLSI circuits become increasingly larger, the delays of a circuit become dominated by interconnect delays. To deal with this problem, it is desirable that gate sizing and placement be incorporated into a single procedure.

In this paper, we first present an efficient algorithm for the gate sizing problem. Then we propose an algorithm which combines the gate sizing problem and timing-driven placement into one procedure. By considering these two problems together, the value of interconnect capacitance is known during the selection stage of the automatic sizing procedure. Therefore, optimal gate sizes can be chosen for each gate based on layout information. In this work, we use row-based layout style. In the following, the terminologies "gate" and "cell" are used interchangeably. Both refer to a module in the circuit.

2 Automatic Gate Sizing

The sizing problem can formulated as

$$\begin{array}{ll} \text{minimize} & Area \\ \text{subject to} & Delay \quad \leq T_{spec}. \end{array} \tag{1}$$

In the discrete gate sizing problem for standard-cell based designs, only a limited number of choices are available for each gate. This problem has been shown to be NP-complete [1].

2.1 Delay modeling

The delay of a gate g_i, denoted by d_i, in a standard cell library can be expressed by

$$d_i \;=\; R^i_{out} \times C^i_{out} + \tau_i = \frac{R_u}{w_i} \times C^i_{out} + \tau_{i1} \cdot w_i + \tau_{i2} \tag{2}$$

R^i_{out} is the equivalent output resistance of g_i, C^i_{out} is the output load capacitance of g_i, R_u represents the on-resistance of a unit transistor, τ_i is the intrinsic delay of the gate, w_i is called the nominal gate size of g_i. Therefore, the delay of each gate can be parameterized by a number, w, referred to as the (nominal) gate size.

The output load capacitance of logic gate j (size $= w_j$) as seen by logic gate i can be approximated by [2]

$$cap(i,j) = \alpha_{ij} \cdot w_j + \beta_{ij} \tag{3}$$

Based on (2) and (3), we deduce that the delay of a gate is a sum of functions of the form $g(w,z) = z/w$; the numerator is proportional to the size of the gate to be driven, and the denominator is proportional to that of the driving gate. It can be observed that the function z/w is fairly smooth and "nearly" convex. It follows that the gate delay $D(w_i, w_1, \ldots, w_f)$ of gate g_i with size w_i, and fanout gate sizes $w_1 \cdots w_f$ can be approximated by a convex piecewise linear function with q regions, $(\mathbf{R}_1, \cdots, \mathbf{R}_q)$, as follows [2]:

$$\begin{aligned} d_i &= \hat{D}(w_i, w_1, \cdots, w_f) \\ &= \begin{cases} \hat{a}_1 w_i + \hat{b}_{1,1} w_1 + \cdots + \hat{b}_{1,f} w_f + \hat{c}_1, (w_i, w_1 \cdots w_f) \in \mathbf{R}_1 \\ \quad \vdots \\ \hat{a}_q w_i + \hat{b}_{q,1} w_1 + \cdots + \hat{b}_{q,f} w_f + \hat{c}_q, (w_i, w_1 \cdots w_f) \in \mathbf{R}_q \end{cases} \\ &= \max_{1 \leq i \leq q} \; (\hat{a}_i w_i + \hat{b}_{i,1} w_1 + \cdots \hat{b}_{i,f} w_f + \hat{c}_i) \\ &\qquad \forall\, (w_i, w_1 \cdots w_f) \in \textstyle\bigcup_{1 \leq i \leq q} \mathbf{R}_i \end{aligned} \tag{4}$$

2.2 Formulation of the linear program

The delay specification states that all path delays must be bounded by T_{spec}. Since the number of PI-PO paths could be exponential, the set of constraining delay equations could potentially be exponential in number. We thus introduce additional variables, m_i, $i = 1 \cdots M$ (M = number of gates), to reduce the number of constraints; m_i corresponds to the worst-case delay from the primary inputs to the output of gate i. Using these variables, for each gate i with delay d_i, we have $m_j + d_i \leq m_i, \forall\, j \in Fanin(i)$. We now formulate the linear program as

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{M} \gamma_i \cdot w_i & \\ \text{subject to} & \text{For all gates } i = 1 \cdots M & \\ & m_j + d_i & \leq m_i \forall\, j \in Fanin(i) \\ & m_i & \leq T_{spec} \forall \text{ gates } i \text{ at } PO\text{'s} \\ & d_i & \geq \hat{D}(w_i, w_{i,1}, \cdots, w_{i,fo(i)}) \\ & Minsize(i) \leq w_i & \leq Maxsize(i) \end{array} \tag{5}$$

where γ_i is the *area coefficient*, a constant associated with gate i. The area of gate i is $\gamma_i \cdot w_i$ if gate i has size w_i.

2.3 Mapping algorithm

The set of permissible sizes for gate i is $\mathcal{S}_i = \{w_{i,1} \cdots w_{i,p_i}\}$, where p_i is the cardinality of $\mathcal{S}_i$. In general, the solution of the linear program provides a gate size $w_i \notin \mathcal{S}_i$. If so, we consider the two permissible gate sizes that are closest to w_i; we denote the nearest larger (smaller) size by w_{i+} (w_{i-}).

We formulate the following smaller problem:

$$\text{For all } i = 1 \cdots M: \quad \text{Select } w_i = w_{i+} \text{ or } w_{i-} \text{ such that Delay} \leq T_{spec}.$$

Although the complexity has been reduced from $O(\prod_{i=1}^{M} p_i)$ to $O(2^M)$, this is still an NP-complete problem. In this section we present an implicit enumeration algorithm for mapping the gate sizes obtained using linear programming onto permissible gate sizes.

The rationale behind our enumeration algorithm is based on the following observation. Given the solution of the linear programming, the majority of the gates remain at their smallest sizes. Only a small portion of the gates in the circuit are moved to a larger size. This may be explained as follows. For a typical circuit, although there may be a huge number of long paths, the number of gates on these long paths are, in general, relatively small.

Based on this observation, during the implicit enumeration procedure we may ignore those gates which are assigned to have their smallest size by the solution of the linear programming, and concentrate on those gates that have been assigned larger sizes in the LP solition and are probably on long paths. We call them *critical gates*.

We modify the circuit topology by adding a source node *so* and a sink node *si* [3]. A dummy edge is added from node *so* to each of the input nodes and from each of the output nodes to the node *si*. Next, for each gate i we define *max-delay-to-sink*, denoted by $mds(i)$, to be the maximum of the delays of all the possible paths starting from gate i to the sink node *si*. That is $mds(i) = \max_{j \in fo(i)}\{mds(j) + d_j\}$

A breadth-first search is applied to levelize the circuit from the sink node *backwards*. The level of a gate g_i in this levelization is called its *backward circuit level*, $c\_level(g_i)$. By definition, $c\_level(si) = 0$. Starting from *si*, we form a state space tree by implicitly enumerating critical gates. During the enumeration, non-critical gates remain at their minimum size and need not be enumerated. Each level in the state space tree corresponds to a critical gate. The corresponding critical gate of level i is gate k. We also define a function $\mathcal{F}(i)$, which is used to indicate the corresponding critical gate of level i. Therefore, if gate k is the corresponding critical gate of level i, then $k = \mathcal{F}(i)$. Similarly, the corresponding level of a critical gate k in the state-space tree is called the gate's *tree level*, $t\_level(k)$. Therefore $t\_level(\mathcal{F}(i)) = i$. Each node at level i in the state-space tree is a *cell configuration*, which represents a possible realization of its corresponding gate. Let $C(i,j)$ denote the jth node at level i, and $anc(i,j)$ be its ancestor node.

Definition 1 *A cell configuration, $C(i,j)$ is a triple (W_{ij}, A_{ij}, D_{ij}),*

$$\begin{aligned} W_{ij} = W_{C(i,j)} &\in \{w_{\mathcal{F}(i)+}, w_{\mathcal{F}(i)-}\}, \\ A_{ij} = A_{C(i,j)} &= \gamma_{\mathcal{F}(i)} \cdot W_{ij} + A_{anc(i,j)}, \\ D_{ij} = D_{C(i,j)} &= max\ \{mds(k)\},\ where\ k\ is\ a\ gate\ in \\ &\quad the\ circuit\ (not\ necessarily\ a\ critical\ gate), \\ &\quad which\ satisfies\ c\_level(k) = c\_level(\mathcal{F}(i)) + 2. \end{aligned}$$

A_{ij} is called the *accumulated area* from the root to $C(i,j)$. (Notice that $\gamma_{\mathcal{F}(i)} \cdot W_{ij}$ is the cell area of gate $\mathcal{F}(i)$, given that its size is W_{ij}.)

In the state space tree, each node has no more than two successors since there are at most two choices for the gate size. The root of the tree is, by definition, assigned a null cell configuration $(0,0,0)$. We begin with the critical gate that has the smallest backward circuit level, and implicitly enumerate the two possible realizations of each gate $\mathcal{F}(i)$, $w_{\mathcal{F}(i)+}$ and $w_{\mathcal{F}(i)-}$. [1] The delay of each gate is dependent on its own size and on the size of the gates that it fans out to. Therefore, once $g_{\mathcal{F}(i)}$ has been enumerated, the delay associated with the predecessor of $g_{\mathcal{F}(i)}$ can be calculated, and the remaining critical gates can be enumerated.

During enumeration, it is possible to prune the search space. A node $C(i,j)$ with a cell configuration (W_{ij}, A_{ij}, D_{ij}) is bounded if there is a cell configuration, (W_{ik}, A_{ik}, D_{ik}), at the same level of the tree such that

(1) $A_{ik} \le A_{ij}$ and $D_{ik} < D_{ij}$, or

(2) $A_{ik} < A_{ij}$ and $D_{ik} \le D_{ij}$.

After all of the critical gates have been implicitly enumerated, we keep calculating max-delay-to-sink for each remaining gate. However, since non-critical gates have fixed sizes, no enumeration is necessary. Rather, we simply propagate the values toward the source node. For each leaf node of the state space tree, the max-delay-to-sink of the source node corresponding to that node is calculated and denoted by D'_{ij}. The cell configuration which has the largest D'_{ij}, and satisfies $D'_{ij} \le T_{spec}$ is selected. By performing a trace-back from the selected leaf node to the root of the tree, the size of each critical gate is determined from the cell configurations at each traversed node.

3 Timing-Driven Placement with Gate Sizing

A circuit can be modeled as a set of M gates (cells), $\mathcal{G} = \{g_1, \cdots, g_M\}$, interconnected by a set of N nets, $\mathcal{N} = \{n_1, \cdots, n_N\}$, that attach to the cells at pins. For the sake of better description, we assume that all gates in the circuit are of single-output. Therefore, net n_i is associated with gate g_i. Hence the same index i can be referred to both a gate and a net. The physical location of a cell i on the chip is represented by (x_i, y_i), where x_i (y_i) is the x (y) coordinate of the center of cell i. The position of a pin is represented by the location of the cell to which the specific pin belongs. The positions of I/O pads are fixed and located on the perimeter of the chip. These constraints act as the boundary conditions.

There are three categories of constraints in our LP formulation; namely physical, timing, and sizing constraints.

3.1 Physical constraints

We approximate the wire length of an individual net by the half-perimeter of the smallest rectangle enclosing the pins of the net [4]. The bounding box for net i is denoted by four parameters, the northern most (η_i), southern most (σ_i), eastern most (ε_i), and western most (ω_i) extents of the pins of the net. Mathematically, the bounding box constraints can be expressed as follows. $\varepsilon_i \ge x_{ij}$, $\omega_i \le x_{ij}$, $\eta_i \ge y_{ij}$, $\sigma_i \le y_{ij}$, $\forall\ 1 \le j \le p_i$, where p_i is the number of pins associated with net i, j is a pin of net i.

Let C_h and C_v denote the unit length wire capacitance in horizontal and vertical layers, respectively. Then the interconnect capacitance, C_i, of net i can be estimated by $C_i = C_h(\varepsilon_i - \omega_i) + C_v(\eta_i - \sigma_i)$. Similarly, the length of net i, l_i, is $l_i = (\varepsilon_i - \omega_i) + (\eta_i - \sigma_i)$. Therefore the total wire length is $\sum_{i=1}^{N} l_i$.

3.2 Timing and sizing constraints

The delay of gate g_i can be contributed to loading capacitance of its fanout gates, plus wire capacitance of its fanout net n_i. Let CL^i_{ij} represent the loading capacitance of gate g_{ij} as seen by g_i. Then the delay of gate g_i is

$$\begin{aligned} d_i &= \frac{R_u}{w_i} \cdot (C_i + \textstyle\sum_{j \in fo(i)} CL^i_{ij}) \qquad (6) \\ &= \frac{R_u}{w_i} \cdot \{C_h(\varepsilon_i - \omega_i) + C_v(\eta_i - \sigma_i) + \sum_{j \in fo(i)} (\alpha_{ij} \cdot w_{ij} + \beta_{ij})\} \end{aligned}$$

where $fo(i)$ is the fanout set of gate i. As in section 2.1, this is a sum of functions of the form z/w. Therefore, it can be approximated by a piecewise linear function.

[1] If there are more than one critical gate which have the same backward circuit level, one of them is randomly chosen.

3.3 Objective function

The objective function of our optimization problem can be formulated as

$$\min \left(\sum_{i=1}^{M} \gamma_i \cdot w_i + \Upsilon \cdot \sum_{i=1}^{N} l_i\right) \tag{7}$$

where l_i is the length of net i, Υ is a constant that is related to the sum of the width of interconnect wire and the minimum distance between two adjacent wires.

The objective function in this formulation represents two important quantities to be minimized in physical design. The first term is the total area of cells. The second term represents the total area taken by interconnect wires.

3.4 Final LP

After introducing the constraints and objective function, we are in a position to formulate the following linear programming.

$$\begin{array}{ll} \text{minimize} & \left(\sum_{i=1}^{M} \gamma_i \cdot w_i + \Upsilon \cdot \sum_{i=1}^{N} l_i\right) \\ \text{subject to} & \text{For all gates } i = 1 \cdots M \\ & m_j + d_i \le m_i \quad \forall\, j \in Fanin(i) \\ & m_i \le T_{spec} \quad \forall\, gates\ i\ at\ PO's \\ & d_i \ge \tilde{D}(w_i, w_{i,1}, \cdots, w_{i,fo(i)}, \varepsilon_i, \omega_i, \eta_i, \sigma_i) \\ & Minsize(i) \le w_i \le Maxsize(i) \\ & \varepsilon_i \ge x_{ij},\ \omega_i \le x_{ij},\ \eta_i \ge y_{ij},\ \sigma_i \le y_{ij} \\ & \qquad \forall\, 1 \le j \le p_i \end{array} \tag{8}$$

The above is a linear program in the variables $w_i, m_i, d_i, x_i, y_i, \varepsilon_i, \omega_i, \eta_i$, and σ_i.

4 A Unified Algorithm for Adjusting Placement and Gate Sizing

Although it is possible to solve Eq.(8) directly, due to the large number of variables and constraints, the execution time may be excessively large. In this section, we present an algorithm which tackles this problem indirectly.

Timing-driven placement algorithms generally require that gate sizes be selected before placement procedure; and gate sizes are fixed during placement [4, 5]. This imposes a restriction on the placement tool in searching for a good placement with minimum wire length. On the other hand, although conventional placement tools can obtain placement with minimal wire length, the delay of the circuit based on that placement may exceed timing constraints. Recently, it has been suggested that a placement which violates the timing constraint could be made to satisfy delay bound by adjusting sizes of some gates, followed by altering the placement in an iterative loop [6]. However, if gate sizing is separated from the placement procedure, excessively large loading capacitances may have been introduced to cells on critical paths for a given placement. In that case, any resizing effort will not be able to obtain a solution. However, if, in addition to gate resizing, cells are allowed to move to different locations at the same time, large wiring capacitances introduced by placement could be reduced. That way, it becomes possible to obtain solutions even for tight delay constraints. In the following, we propose an algorithm which combines gate resizing and relocation in one procedure to satisfy timing constraints for a given compact placement; in the meanwhile the total circuit area (including cell area and wire length) is kept minimum.

First, all of the gates in the circuit are set to their minimum sizes. A compact placement is obtained with the objective of minimizing total wire length. This can be done by using existing placement packages. After that, wiring capacitance associated with the output of each gate is calculated. Based on this information, together with the circuit structure, optimal gate sizes are selected using the gate size optimization algorithm described in section 2. In general, some gates will be selected to have larger sizes. This may cause overlap between cells. This problem can be solved by shifting cells to avoid overlap. In general, however, the perturbation on the delay of individual gate may cause the circuit delays to exceed delay constraints. Once the algorithm detects such delay violations, a number of gates are selected as described below. These gates will be resized and/or moved to different locations in order to satisfy time constrains and to minimize total circuit area (including cell area and wire length). The outline of our algorithm is shown is Figure 1.

```
ALGORITHM Resizing & Relocation()
 1.  do initial placement;
 2.  do initial gate sizing for all cells;
 3.  while (timing constraints not satisfied) {
 4.    select gates belonging to type 1, 2, and 3;
 5.    formulate LP (Eq. 8) for these gates;
       (remaining cells serve as boundary conds.)
 6.    solve the LP;
 7.    use mapping algorithm (sec.2.3) to obtain
         permissible size for each gate;
 8.    adjust cell locations to avoid overlap;
 9.  }
10.  report final placement;
```

Figure 1: Resizing & Relocation algorithm.

We introduce the *required signal arrival time*, r_i. The required signal arrival time, as defined below, is the latest time a signal has to be present at the output of gate i in order to satisfy delay constraints at POs.

$$r_i = \begin{cases} T_{spec}, & \text{if gate } i \text{ at PO} \\ \max\{r_j - d_j \mid \forall\, j \in Fanout(i)\}, & \text{otherwise} \end{cases} \tag{9}$$

For each gate i, we also define a slack $s_i = r_i - m_i$. An *active gate* i is a gate with $s_i < 0$. The timing of a circuit layout is said to be satisfied if and only if $s_i \ge 0$ for every gate i in the circuit.

The unified optimization algorithm begins by calculating the slack of each gate. Then three types of gates are selected for improvement.

1. The first type is active gates. These gates will be allowed to change their sizes and to move to new locations.
2. The second type involves those gates with nonnegative slacks less than a small specified value, δ. During this phase output load capacitances of certain gates will be perturbed, which results in delay changes. Therefore, it is advantageous to include those gates with small nonnegative slacks in the formulation to avoid additional iteration.
3. The third type of gates includes those that are directly connected to the outputs of active gates. This is because in order to reduce the delays of those active gates, besides changing their sizes, can also be accomplished by reducing their output load capacitances.

Gates belonging to types 1, 2, and 3 are put into the linear program, Eq.(8), and a new solution is obtained. In principle, to obtain a better solution, it is necessary to include all three types of gates in the linear program. In practice, however, to maintain the efficiency of the program, it is necessary to limit the number of gates to be included. Since many gates' locations are fixed, they serve as boundary conditions for physical constraints. To avoid drastically changing the solution, each selected gate is allowed to change to its nearest larger or smaller size only.

The solution of a such formulated linear program gives a new size and a new position for each selected cell. The

Table 1: Performance comparison of GALANT with simulated annealing.

Circuit	gates	T_{spec}	Simulated Annealing		GALANT		
			Area (A_{SA})	Runtime	Area (A_G)	Runtime	A_G/A_{SA}
c432	160	16.0	2372	19m 53s	2376	4.82s	1.002
		14.0	2515	21m 17s	2515	5.38s	1.000
		12.0	2950	24m 27s	2983	7.72s	1.011
c2670	1193	17.0	17623	5h 22m	17623	4m 12s	1.000
		16.0	17772	5h 42m	17790	4m 30s	1.001
		14.0	18929	8h 12m	19079	7m 8s	1.008
c7552	3512	18.0	50557	22h 5m	50604	35m 49s	1.001
		17.0	50740	23h 20m	51254	52m 27s	1.010
		16.0	52069	24h 5m	52563	1h 11m	1.009

Table 2: Experimental results of PRECISE.

Circuit	T_{spec}	Iterative approach			PRECISE			$\frac{A_P}{A_M}$	$\frac{L_P}{L_M}$
		cell area (A_M)	wire length (L_M)	runtime	cell area (A_P)	wire length (L_P)	runtime		
c432	14.0	3111	785	31.22s	3061	732	45.77s	0.984	0.922
	13.0	3726	912	31.58s	3484	796	59.26s	0.935	0.873
	12.0	-	-	-	4344	913	1m 52s	-	-
c2670	25.0	18015	11243	12m 31s	17680	10710	7m 26s	0.981	0.953
	23.0	18648	11462	14m 14s	18408	10840	8m 11s	0.987	0.946
	21.0	-	-	-	19692	11788	8m 57s	-	-
c7552	27.0	50968	43625	2h 3m	51046	42524	1h 0m	1.007	0.975
	24.0	-	-	-	52699	43613	1h 14m	-	-
	23.0	-	-	-	54088	43673	2h 2m	-	-

mapping algorithm described in section 2.3 is used to obtain permissible gate size. Since many cells are moved to new locations, and some of them are replaced with templates of different sizes, there may be overlap among cells. Therefore it is necessary to move cells into (slightly) different location to avoid overlapping. Due to space limitation, details of the moving algorithm is omitted.

If necessary, the above procedure is repeated until delay constraints are all satisfied.

5 Experimental Results and Conclusion

The above algorithms have been implemented in C in programs GALANT (for gate sizing) and PRECISE (for placement with cell resizing and relocation), on a Sun Sparc 10 Station. Each cell in the standard-cell library has four realizations with different sizes and different driving capabilities.

To prove the efficacy of the gate sizing approach described in section 2, a simulated annealing algorithm was implemented for comparison. The results for 3 ISCAS 85 benchmark circuits are shown in Table 1. It should be mentioned that the chief component (over 95%) of the runtime of GALANT was the linear programming algorithm; the mapping algorithm was extremely fast in comparison.

The experimental results of the program PRECISE, is summarized in Table 2. At present, we use Fiduccia's min-cut partitioning algorithm [8] to obtain a compact placement. More compact placement can be obtained by using other algorithms (e.g., TimberWolf). For comparison, we also perform placement and gate sizing based on purely iterative approach. That is, placement adjustment and gate resizing are executed separately and are included in an iteration loop. The experimental results show that PRECISE is able to obtain better solutions than the iterative approach. Moreover, for very tight timing bounds, the iterative approach fails to obtain solutions (indicated by "-" in the table). This is because cell locations are fixed in the iterative approach, and excessively large capacitances may have been introduced at the output of some gates on critical paths. On the other hand, in addition to resizing cells, PRECISE also moves cells to different locations to reduce large wiring capacitance. Therefore it is able to obtain solutions even for tight delay bounds. Furthermore, instead of resizing all cells, PRECISE resizes only a small portion of cells when timing bounds are violated. As a result, its execution time is faster in general.

In summary, for the first time, the gate sizing problem is combined with placement into one formulation. The experimental results are very encouraging. At present, we are investigating more accurate delay models and more powerful optimization techniques to tackle this problem.

REFERENCES

[1] P. K. Chan, "Algorithms for library-specific sizing of combinational logic," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 353–356, 1990.

[2] W. Chuang, *Timing and area optimization for VLSI circuit and layout.* PhD thesis, University of Illinois at Urbana-Champaign, 1994.

[3] S. H. Yen, D. H. Du, and S. Ghanta, "Efficient algorithms for extracting the K most critical paths in timing analysis," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 649–654, 1989.

[4] M. A. Jackson and E. S. Kuh, "Performance-driven placement of cell based IC's," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 370–375, 1989.

[5] W. E. Donath, et al, "Timing driven placement using complete path delay," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 84–89, 1990.

[6] S. Kim, et al, "APT: An area-performance-testability driven placement algorithm," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 141–146, 1992.

[7] S. Lin, M. Marek-Sadowska, and E. S. Kuh, "Delay and area optimization in standard-cell design," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 349–352, 1990.

[8] C. Fiduccia and R. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 175–181, 1982.