Survey on Transistor Sizing for Automatic Timing-Driven Layout Synthesis

Xiaoshu Chen

Supervisors: Anthony J. Kusalik and Carl McCrosky

Abstract

The increasing complexity of modern VLSI circuits, the requirement for a rapid design cycle, and the need for efficient implementations all make design automation an absolute necessity, especially when timing constraints are of great concern. In this survey, we examine the basic theory on how to meet timing requirements while using a minimum amount of chip area. We study **Timing-Driven Layout Synthesis**. Possible solutions for this problem are logic simplification, optimal I/O pin assignment, cell placement improvement, routing topological optimization, wiresizing, appropriate transistor ordering, and transistor sizing. Several techniques of **Transistor Sizing** for automatic synthesis are investigated and compared, among which **Convex Programming** is the best; it is guaranteed to find an optimal solution at a relatively low computational cost.

1 Introduction

The overall design flow of an IC chip is [1]: 1) problem definition in hardware description language; 2) logic representation at gate level; 3) circuit design at transistor level; 4) layout design at silicon level. The first two steps are technology-independent, whereas the rest are technology-dependent; technology mapping is needed to fill in the gap. Several technologies are in fashion now, such as PLAs, gate array, sea-of-gates (SOG), and standard cells.

Due to rapid progress in VLSI technology, the overall complexity of chips has increased dramatically since 1980's. On one hand, more and more powerful functions are integrated together; on the other hand, higher and higher speed is achieved.

The design objectives for VLSI design are:

1. Logic accuracy;
2. Maximize transistor density, minimize active chip area, and control the X/Y aspect of chip;
3. Meet timing constraints;
4. Meet power dissipation constraints, and
5. Maintain reasonable noise margins.

Timing-Driven Layout Synthesis discussed in this paper focuses on timing constraints and area efficiency.

The delay of a circuit network is due to the electrical behavior of cells in the circuit, the interconnects, and to their interactions. From the design process viewpoint, several roads can lead to the reduction of signal delay:

- First, logic reorganization results in fewer stages between an input node and an output node. Generally speaking, the signal path will be shortened.

- Second, cell placement and interconnect topology influence timing. Architectures with shorter wires will run faster.
- Third, in the case of long buses, buffer insertion is necessary because the timing of the bus is proportional to the square of the bus length.
- Fourth, theoretical analysis and experimental results [2] show resizing of wires is a good candidate for improving circuit speed.
- Fifth, when circuits contain a series chain of transistors, transistor reordering can be employed to achieve speed improvement with nearly zero penalty (i.e., very little, if any, increase in layout area). This technique can also be used to decrease dynamic power dissipation. It is proposed [8] to consider only two extreme cases of reordering which sorts the inputs according to arrival time and places the latest arriving signal next to the output (respectively, power) node.
- Sixth, most current research work is conducted at the transistor level, which is the most significant contributor to circuit delay. As we will discuss later, the size of transistors determines their timing characteristics. This paper focuses on **Transistor Sizing**.

2 Timing Model and Estimation

The first step is to model the timing behavior of the circuit and understand the determinants of circuit speed. Then, the model can be used to calculate the delay of a specific circuit given its parameters; these parameters are taken as design variables and are adjusted to improve the speed of the circuit and meet the requirements of the problem.

Although sequential circuits are different from combinational circuits, sequential circuits can be decomposed into multiple stages of combinational logic blocks that lie between latches. As shown in Figure 1, the clock arrival times at these latches are used to determine the timing specifications of the combinational blocks [10]. So we treat the two kinds of circuit the same.

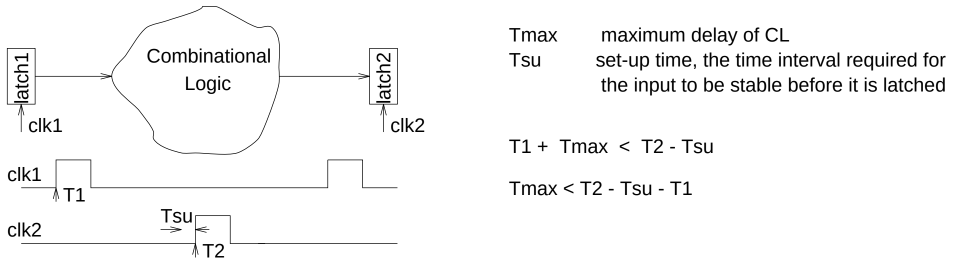

Figure 1: Combinational subnetwork extraction

Device level simulations like SPICE produce accurate and detailed delay information, but analytical delay models are required in general, because SPICE computation time increases rapidly with the number of transistors.

2.1 RC Model

A CMOS transistor is modeled as a voltage-controlled switch with on-resistance R_{on} between drain and source and three grounded capacitances C_d, C_s, and C_g from the drain, source, and gate terminals, respectively, to ground. Strictly speaking, interconnect is a distributed RC system, with cross-coupling capacitances between lines. All of these are illustrated in Figure 2.

As indicated in Equation 1, using the resistance and capacitance of transistor and wire as the parameters of timing is a broadly accepted model which will be accepted here without justification [7].

$$\tau = R * C \tag{1}$$

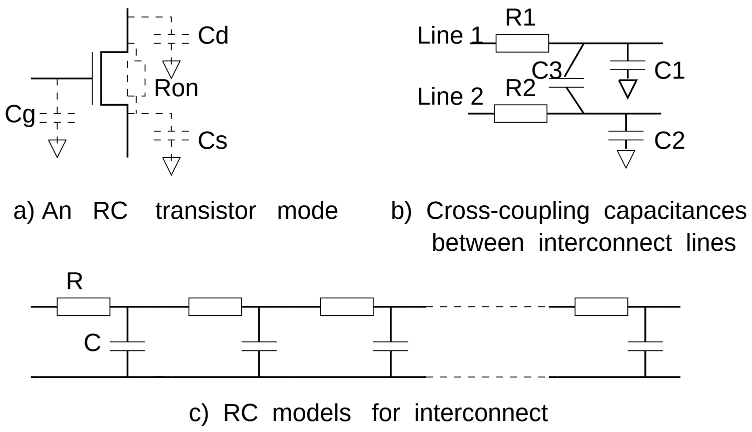

Figure 2: RC representation of a CMOS circuit

Circuits are treated as RC networks. The delay at any node of the network is well approximated by what is called Elmore delay [3]. For a network with $\mathbf{n}$ nodes, let $\vec{\mathbf{C}}$ denote corresponding $\mathbf{n}$ grounded capacitors, and $\mathbf{G}$ be a $\mathbf{n} * \mathbf{n}$ matrix with $\mathbf{G_{ij}}$ representing the branch conductance between node i and j, then Elmore delay $\mathbf{T_{D_i}}$ to the $\mathbf{i^{th}}$ node is given by Equation 2, where $\vec{\mathbf{T}}_{\mathbf{D}} \in \mathbf{R^n}$ and $\mathbf{T_{D_i}} = (\vec{\mathbf{T}}_{\mathbf{D}})_{\mathbf{i}}$.

$$\vec{\mathbf{T}}_{\mathbf{D}} = \mathbf{G^{-1}}\vec{\mathbf{C}} \tag{2}$$

The inverting of $\mathbf{G}$ is computationally expensive. However, in the case of an RC tree, the computing of $\mathbf{T_{D_i}}$ becomes much easier. Let $\mathbf{R_{ij}}$ be the sum of resistances along the common portion of the paths from input node to node $\mathbf{i}$ and node $\mathbf{j}$. Equation 2 is simplified to

$$\mathbf{T_{D_i}} = \sum_{\mathbf{j=0}}^{\mathbf{n}} \mathbf{R_{ij}C_j} \tag{3}$$

2.2 Macromodeling

Modeling a circuit at a logic gate level eliminates all network nodes internal to gates, as a consequence, computational complexity will be greatly reduced. Moreover, it incorporates the effects of input waveform which might contribute significantly to gate delay, while RC modeling can not do it.

The idea here is:

1. Decompose the network into channel-connected components, each component corresponds to a set of transistors that are connected by drain and source nodes,

2. Then, map each component onto a basic element, say, an inverter, with some component-dependent parameters. An example is given in Figure 3 on how to map a two-input NOR gate to an equivalent inverter.

3. At last, look up in a table, taking those parameters as table entry; or calculate an analytical expression, using those parameters as actual arguments; both will give the delay of the component.

2.3 Worst-Case Delay and Critical Path

Among all the paths between an input and an output, the delay along different paths will, in general, be different. At the circuit level, the path with the maximum delay, namely, the *critical path* (CP), must meet the design constraint. We do not need to consider other cases. At the component level, the worst-case delay is defined as the latest rising and falling transition time at the output node of a component.

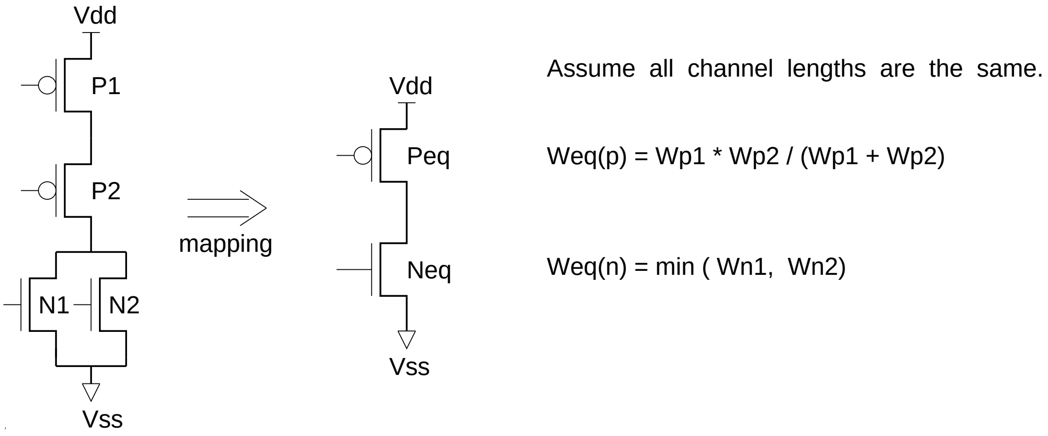

Figure 3: An example of Macromodeling

Then the question arises, how can the slowest one be distinguished out of so many paths? From the example in Figure 4, the method is quite straightforward:

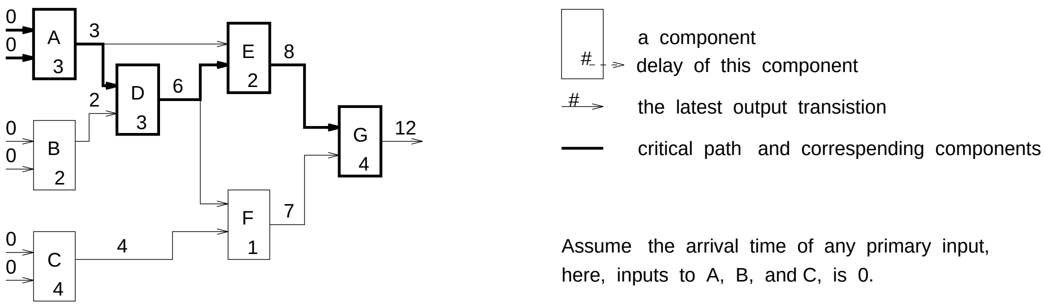

Figure 4: Circuit delay computation

1. Enumerate all paths between the output node and V_{DD} or V_{SS}, in the component, then determine the worst one. That is the delay of this component.

2. Stage-by-stage, component-by-component, add the latest arriving input time to the delay of the component, to obtain the worst-case transition time at the output.

3. Find the primary output with the latest transition time. The component associated with this output is the last component on the CP.

4. Backtrack along this component, and identify its latest arriving input. The component that causes this input is the preceding component on the CP.

5. Recursively, CP can be found.

Since the number of paths could grow exponentially with the number of transistors, certain pruning strategies have to be employed to improve the computing speed. A frequently used approach is based on the fact that the *largest resistive path* (LPR) is often the worst-case path in most CMOS circuits. LPR is a path of largest resistance between the output and power or ground, and through the transistor with the latest arrival input.

3 Transistor Sizing

The underlying motivation for transistor sizing is apparent in Equations 2 and 3, where C_g, C_d, and C_s are, respectively, the capacitances at the gate, drain, and source of the transistor. W and L are the channel width and channel length of the transistor [7], respectively.

$$R \;\propto\; \frac{L}{W} \tag{4}$$

$$C_g, C_d, C_s \;\propto\; L * W \tag{5}$$

In general, L is kept at the minimal allowable value of channel length, that is a constant related to the specific silicon technology. So W is the only factor which actually affects R and C. At the transistor level, the delay is expressed as

$$\tau \;=\; \frac{R}{W} \;*\; C_L \tag{6}$$

where R is the resistance of a transistor with a unit width, and C_L is the load capacitance, which is determined by the capacitances of all fanout transistors and interconnect wire.

Increasing the size of a transistor,here W,will certainly decrease the timing of this transistor due to resistance reduction.However,it will also increase the capacitive load,C_L,seen by those driving transistors. Consequently,timing associated with those transistors will be increased. So,transistor sizing will not definitely result in the reduction of delay. The delay-area relationship is nonmonotonic [4].

3.1 Problem Definition

The transistor sizing problem is normally described as

$$Area \;=\; \sum_{i=1}^{n} w_i \tag{7}$$

$$\begin{array}{ll} Minimize & Area \\ Subject\ to & Delay \leq T_{spec} \end{array} \tag{8}$$

Where n is the number of transistors in the circuit, w_i is the channel width of the i^{th} transistor. Without loss of accuracy and generality, we may approximate the active circuit area with the sum of all transistor channel width.

3.2 TILOS

The main idea in TILOS [5], which stands for **TImed LOgic Synthesizer**, is that the area and delay can be formulated as posynomial functions of transistor sizes.

3.2.1 Posynomial

A posynomial is a function $\mathcal{F}$ of a positive variable $\vec{X} \in \mathbf{R^n}$,

$$\mathcal{F}(\vec{X}) \;=\; \sum_j \gamma_j \prod_{i=1}^{n} x_i^{\alpha_{ij}} \tag{9}$$

where $\alpha_{ij} \in \mathbf{R}$ and $\gamma_j > 0$. Equation 7 is a posynomial. The unimodality of posynomial means it has only one relative extremum which is naturally a peak value. Further analysis demonstrates the convex property of a posynomial which means the extremum is a minimum. Therefore, before it reaches the minimal point, an increase of x_i results in decrease of $\mathcal{F}$. TILOS makes use of this property.

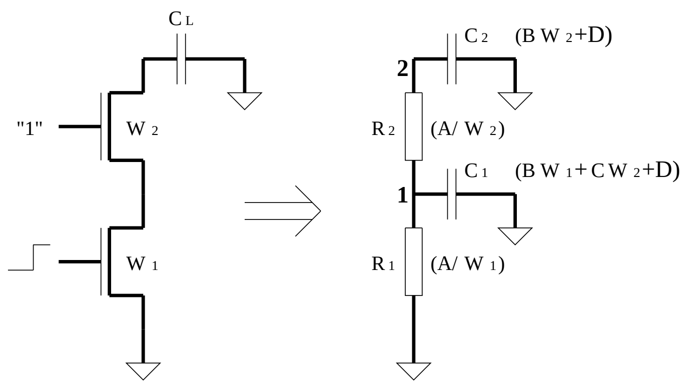

Figure 5: A pulldown network and its equivalent RC representation

3.2.2 Delay Model

As an example, the path delay in Figure 5 is given as follows,

$$R_1C_1 + (R_1 + R_2)C_2 \tag{10}$$

From Equations 4 and 5, C_i is the capacitance between node i and ground, which consists of the wire capacitance (a constant, D) and capacitance of transistors connected to node i (proportional to w_j), while R_i is inversely proportional to the corresponding w_i. As shown in Figure 5, Equation 10 can be transformed to Equation 11, which is a posynomial.

$$(\frac{A}{w_1})(Bw_1 + Cw_2 + D) + (\frac{A}{w_1} + \frac{A}{w_2})(Bw_2 + E) \tag{11}$$

3.2.3 Algorithm

Initially assuming all transistors are at the minimum allowable size, the TILOS optimization follows the steps listed below:

1. Static timing analysis is used to find the CP of the circuit.
2. Along the CP backward, whenever an output node of a component is visited, TILOS will extract the LPR through this node.
3. The delay along the LPR can be expressed in the form of Equation 11, a posynomial.
4. The sensitivity, which is the reduction of circuit delay per increment of transistor size (the derivative of delay expression), of each transistor along LPR is derived. The most sensitive transistor is examined.
5. The size of the transistor with the greatest sensitivity is increased by multiplying it by a constant.
6. Repeat above process, until either all the constraints are satisfied and solution is found, or any increase in transistor sizes would make the circuit slower rather than faster in which case TILOS fails to find a solution.

It is necessary to point out that a change of transistor size along the current CP could increase the delay along some other paths, resulting in a new CP, and a circuit delay is the maximum of all path delays. So, instead of minimizing the current CP delay by setting the size of the most sensitive transistor to a specific value (minimum point of CP delay expression), TILOS attempts to approach the optimal point gradually as Step 5 stated; otherwise it will probably make a new path critical with even longer delay at the cost of area increasing.

3.3 Method of Feasible Directions (*MFD*)

MFD [6] is a two-phase algorithm for transistor sizing. The problem is abstracted as a nonlinear mathematical programming problem with a set of constraint functions (here, delay constraints) and a cost function (here, area function). At the first phase, it generates an initial solution which is inside the feasible region (the set of transistor sizes where all delay constraints are satisfied); at the second phase, it reduces the value of the cost function without leaving the feasible region. Generalized MFD proceeds as follows:

- Find a search direction $\vec{\mathbf{d}}$, a vector in the n-dimensional space of the design parameters (here, channel width w), based on the gradients of the cost function and some of the constraints. It can be proved there exists a step small enough along $\vec{\mathbf{d}}$ that decreases the cost function and remains feasible.
- Once $\vec{\mathbf{d}}$ has been found, a step along $\vec{\mathbf{d}}$ is computed so that the decrease in the cost and constraint function is large enough.
- In theory, it converges to a point where the length of $\vec{\mathbf{d}}$ is zero. In practice, it stops when the length of $\vec{\mathbf{d}}$ is small enough.

The converging process is very slow mainly because of the problem size, so only a modified version of MFD is in practical use. The transistors on the worst-delay paths are selected as design parameters, the number of which is certainly much less than the number of transistors in the entire network. TILOS is employed to produce the initial guess.

The algorithm in the form of pseudo-code is:

$$
\begin{array}{l}
do \;\; \{ \\
\qquad Extract\ P_1,\ P_2,\ ...,\ P_k, \quad the\ k\ worst-delay\ paths, \\
\qquad Let\ \vec{W}\ =\ \{w_i\}, \quad the\ set\ of\ design\ parameters \\
\qquad Solve\ the\ optimization\ problem \\
\qquad\qquad minimize \qquad \sum_{w_i \in \vec{W}} w_i \\
\qquad\qquad subject\ to \quad P_i(\vec{W}) \leq T_{sepc} \qquad i\ =\ 1, ..., k \\
\} \quad until\ (all\ constraints\ are\ met)
\end{array}
$$

3.4 Two-Step Optimizer

A typical transistor sizing problem has as many variables as the number of transistors in the circuit, which is so large that it's difficult for the algorithm to handle effectively. Hierarchy can be used to get around the problem. This is the motivation for *two-step optimization* [9].

In the first step, macromodeling is performed on each component of the circuit and the corresponding equivalent primitive is formed, for example, an inverter. The delay of these primitives is taken as the *Timing Budget* of the associated component.

In the second step, at the component level, a smaller transistor sizing problem is solved, where the area of the component is minimized such that its delay is within its timing budget.

4 Convex Programming for Transistor Sizing

As discussed in Section 3, the transistor sizing problem is defined in the form of a posynomial, which, in turn, can be transformed to a convex function by letting $w_i = e^{z_i}$. After the abstraction, we get a *convex programming* problem which is "how to minimize a convex function (the area) over a convex constraint set (the delay constraint)?"

The algorithm works in the space $\mathbf{R^n}$, where $\mathbf{n}$ is the number of transistors in the network. Global optimization over the parameter space of all transistor sizes is performed and an exact solution can be given.

Equation 8 is converted to the n-dimensional convex programming problem in the form of

$$\begin{aligned} &Minimize \qquad Area(\vec{z}) = \sum_{i=1}^{n} e^{z_i} \\ &Subject\ to \quad Delay(\vec{z}) \leq T_{spec} \end{aligned} \tag{12}$$

4.1 Algorithm

At first, we introduce the *feasible set*

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

$\mathcal{S}$ defines the set of points which meet the delay constraints. The solution of Equation 12, $\vec{z}_{opt}$, is a point within $\mathcal{S}$. The detailed steps are:

1. Initially, a polytope (i.e., a bounded polyhedron) $\mathcal{P}$ is chosen, where $\mathcal{P} \subset \mathbf{R^n}$, and $\vec{z}_{opt} \in \mathcal{P}$. Generally, $\mathcal{P}$ is selected as:

$$\mathcal{P} = \{\vec{z} : log_e(w_{min}) \leq z_i \leq log_e(w_{max})\} \tag{14}$$

2. Locate the center $\vec{z}_c$ in space $\mathcal{P}$, whose property is that any separating hyperplane passing through it will cut the polytope $\mathcal{P}$ into two almost equal sub-polytopes. In fact, finding such a point is quite difficult; a point that is reasonably deep within the polytope is computed as a substitution [1].

3. Compute $Delay(\vec{z}_c)$. $\mathcal{P}$ is cut into two parts by a separating hyperplane through $\vec{z}_c$. The part where $\vec{z}_{opt}$ is located can be identified and it is the new value of $\mathcal{P}$.

4. Repeat step 2 until $\mathcal{P}$ is sufficiently small. In fact, while near the optimum, the polytope becomes flat in the direction normal to the gradient of the area. Process terminates based on this property.

4.2 An Example

A simple example of a two dimensional transistor sizing problem (two transistors in the circuit) in the form of convex programming is given in Equation 15. The following diagram clearly exposes the underlying rationale of convex programming.

$$\begin{aligned} &Minimize \qquad Area(\vec{W}) = \sum_{i=1}^{2} w_i \\ &Subject\ to \quad (w_1,\ w_2) \in \mathcal{S} \end{aligned} \tag{15}$$

where $\mathcal{S}$ is a convex set (the feasible set), and *Area* is a convex function which will be minimized.

- The initial two dimensional polytope $\mathcal{P}$ is given in Figure 3(a). $\mathcal{S}$ should be a subset of $\mathcal{P}$, and $\vec{Z}_{opt}$ is a point within $\mathcal{P}$. The center $\vec{z}_c$ of $\mathcal{P}$ can be found. Also it is obvious that *Area* decreases while either w_1 or w_2 decreases, which contains some hints on how to select one of the two sub-polytopes to replace $\mathcal{P}$ for continuous computation.

- Depending on whether $\vec{z}_c$ lies within $\mathcal{S}$ or not, a different strategy is invoked to generate a dividing plane across $\vec{z}_c$. Figure 3(b), (c), and (d) illustrate the different cases respectively. From the logic point of view, the principle of choosing one polytope to continue the algorithm must ensure that $\vec{z}_{opt}$ will definitely locate there, and it is already verified in [1] that it is mathematically solvable and also computable.

- Continue till $\mathcal{P}$ is less than a user-defined value.

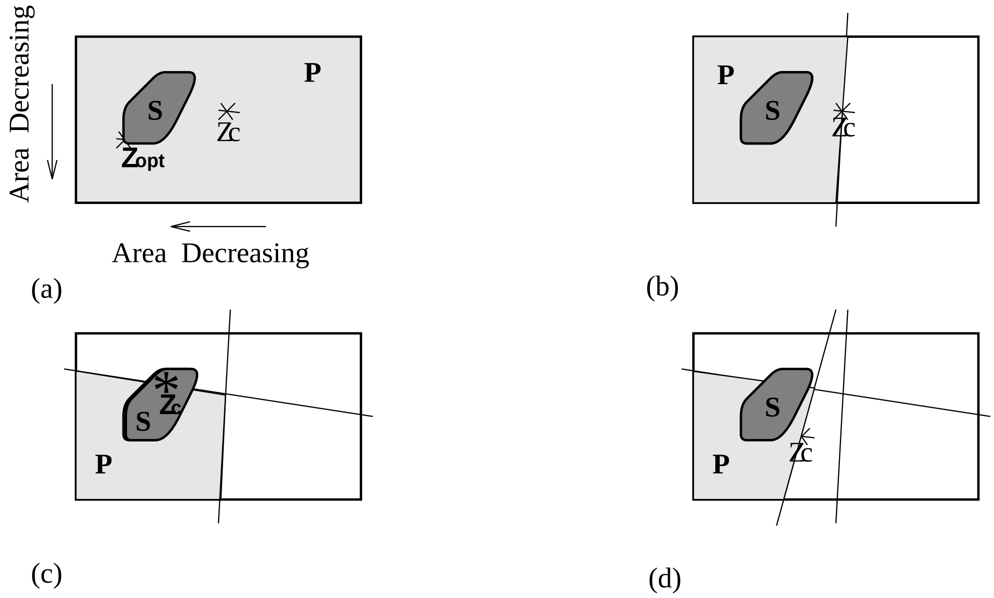

Figure 6: Example of Convex Programming Algorithm

5 Comparisons and Conclusion

Due to the huge runtime cost associated with the transistor sizing problem, most practical algorithms focus on heuristic simplifications of the problem. These algorithms lead to computational efficiency at the cost of either finding no solution or a suboptimal solution.

TILOS fails to guarantee optimality for three reasons: 1) it only picks the greatest-sensitivity transistor to size up, so it lacks a global view of the whole circuit; 2) it increases the size of the transistors only, while in certain case it may be effective to decrease the size of some devices; 3) as the maximum function of all possible path delays within the circuit, signal delay is of discontinuity over transistor sizes.

MFD is more advanced than TILOS, it works on the parameter space consisting of all the transistors on the worst-delay paths, but it is still a reduced space, so it does not guarantee a global minimum solution.

The two-step method is an aggressive attempt to solve the problem. Unfortunately, since the allocation of time budget to each component may not be the same as the delay of the component for the optimal solution, the sum of local optimizations is not automatically a global optimization.

Convex programming is advantageous over other transistor sizing approaches in several ways. First, it is strictly defined as a mathematical problem, and it can be verified that it guarantees to find the global optimal solution. Second, the delay constraints are easily incorporated into the computation of $Delay(\vec{z}_c)$ and the generation of separating hyperplane. Third, its computational complexity is related to the number of transistors, the size of the initial space, and the termination criterion, and is relatively small with respect to other afore-mentioned methods. The computational complexity of those other methods is more or less dependent on the number of constraints, which grows exponentially with circuit complexity in the terms of the number of transistors. Experimental data [1] shows convex programming outperforms other heuristic algorithms, especially when timing specifications become tight.

6 Future Work

Up to date, the main difficulty of the transistor sizing problem is the huge computational cost. As an extension of logic programming (LP), CLP (constraint logic programming) [11] inherits the feature of rapid prototyping from LP. It is very convenient for exploring the requirements of a problem in an experimental way. Also as an improved version, it integrates the resolution mechanism of LP with some specific constraint solving techniques and enhances the efficiency of the resolution. Even more, CLP goes beyond an exploratory prototyping. It may be a good approach to the transistor sizing problem, which includes many constraint functions. So next step is to study the functionality of CLP and find a bridge to link it to our interest in circuit optimization.

References

[1] Sapatnekar, S. and Kang, S. *Design Automation for Timing-Driven Layout Synthesis.* Kluwer Academic Publishers, 1993.

[2] Cong, J., Leung, K. and Zhou, D. Performance-Driven Interconnect Design Based on Distributed RC Delay Model. In *30th ACM/IEEE Design Automation Conference*, June 1993, pp. 606-611.

[3] Wyatt, Jr. Signal Delay in RC mesh networks. *IEEE Transactions on Circuits and Systems*, 17 (May 1985), pp. 507-510.

[4] Hedlund K. AESOP: A tool for automated transistor sizing. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, June 1987, pp. 114-120.

[5] Fishburn, J. and Dunlop, A. TILOS: A posynomial programming approach to transistor sizing. In *Proceedings of the 1985 International Conference on Computer-aided Design*, Nov. 1985, pp. 326-328.

[6] Shyu, J., Fishburn, J.P., Dunlop, A.E. and Sangiovanni-Vincentelli, A.L. Optimization-based transistor sizing. *IEEE Journal of Solid-State Circuits*, 34 (Apr. 1988), pp. 400-409.

[7] Weste, H.E. and Eshraghian, K. *Principles of CMOS VLSI design, A System Perspective.* Addison-Wesley, 1992.

[8] Carlson, B.S. and Chen, C.Y. Performance Enhancement of CMOS VLSI Circuits by Transistor Reordering. In *30th ACM/IEEE Design Automation Conference*, June 1993, pp. 361-366.

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

[10] Lai, F. and Kang, S.M. and Trick, T.N. A timing error corrector for VLSI synchronous path circuits. In *Proceedings of the 1987 IEEE International Conference on Computer-aided Design*, 1987, pp. 48-51.

[11] Rueher, M. and Legeard, B. Which role for CLP in software engineering? In *Proceedings of the 1992 International Conference on the Practical Application of Prolog*, April 1992.