Annals of Operations Research 105, 37–60, 2001

VLSI Circuit Performance Optimization by Geometric Programming *

CHRIS CHU cnchu@iastate.edu
Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011, USA

D.F. WONG wong@cs.utexas.edu
Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712, USA

Abstract. Delay of VLSI circuit components can be controlled by varying their sizes. In other words, performance of VLSI circuits can be optimized by changing the sizes of the circuit components. In this paper, we define a special type of geometric program called *unary geometric program*. We show that under the Elmore delay model, several commonly used formulations of the circuit component sizing problem considering delay, chip area and power dissipation can be reduced to unary geometric programs. We present a greedy algorithm to solve unary geometric programs optimally and efficiently. When applied to VLSI circuit component sizing, we prove that the runtime of the greedy algorithm is linear to the number of components in the circuit. In practice, we demonstrate that our unary-geometric-program based approach for circuit sizing is hundreds of times or more faster than other approaches.

Keywords: VLSI design, unary geometric programming, circuit performance optimization, transistor sizing, gate sizing, wire sizing, Lagrangian relaxation

1. Introduction

Since the invention of integrated circuits almost 40 years ago, sizing of circuit components (e.g., transistors and wire segments) has always been an effective technique to achieve desirable circuit performance. The reason is both resistance and capacitance of a circuit component are functions of the component size. Since the delay of a circuit component can be modeled as a product of the resistance of the component and the capacitance of the subcircuit driven by the component, the delay of a circuit can be minimized by sizing of circuit components.

Both transistor/gate sizing [6,12,15,16,21] and wire sizing [2,4,7,9,18,20] have been shown to be effective to reduce circuit delay. Transistor sizing is the problem of changing the channel length of transistors. Gate sizing is basically the same as transistor sizing. Gate is a collection of transistors working together to perform a specific logic function. Gate sizing refers to the problem of sizing the transistors inside a gate simultaneously by the same factor. Wire sizing refers to the problem of determining the width of the wires at every point along the wires. To make the design and fabrication process eas-

* This work was partially supported by the Texas Advanced Research Program under Grant No. 003658288 and by a grant from the Intel Corporation.

ier, wires are usually divided into fixed-length segments and every point in a segment is sized to the same width. Since transistor/gate sizes affect wire-sizing solutions and wire sizes affect transistor/gate-sizing solutions, it is beneficial to simultaneously size both transistors/gates and wires [2,7,8,17,19]. However, the simultaneous problem is harder to solve.

In this paper, we consider performance optimization of VLSI circuits by sizing components. In order to simplify the presentation, we illustrate the idea by simultaneous gate and wire sizing. All techniques introduced in this paper can be easily applied to simultaneous transistor and wire sizing. The widely used Elmore delay model [11] is used here for delay calculation.

Various formulations of the sizing problem considering delay, chip area and power dissipation have been proposed. When delay alone is considered, two commonly used formulations are minimizing a weighted sum of the delay of components, and minimizing the maximum delay among all circuit outputs. Besides delay, it is desirable to minimize the chip area occupied by the circuit and the power dissipation of the circuit as well. All these objectives can be optimized effectively by circuit component sizing. However, these objectives are usually conflicting. As a result, to consider the tradeoff of these design objectives, formulations like minimizing the maximum delay among all circuit outputs subject to bounds on area/power, and minimizing area/power subject to delay bounds on all circuit outputs have been proposed.

Fishburn and Dunlop [12] have already pointed out that for the transistor sizing problem, several formulations can be written as geometric programs [10]. In fact, by generalizing the idea of [12], it is not difficult to see that all formulations listed above can be written as geometric programs. However, it would be very slow to solve them by some general-purpose geometric programming solver. So instead of solving it exactly, many heuristics were proposed [6,12,15,16]. Sapatnekar et al. [21] transformed the geometric programs for transistor sizing into convex programs and solved them by a sophisticated general-purpose convex programming solver based on interior point method. This is the best known previous algorithm that can guarantee exact transistor sizing solutions. However, to optimize a circuit of only 832 transistors, the reported runtime is already 9 hours on a Sun SPARCstation 1.

In this paper, we define a special type of posynomial [10] and geometric program:

Definition 1. A *unary posynomial* is a posynomial of the following form:

$$u(x_1, \ldots, x_n) = \sum_{1 \leqslant i \leqslant n} \frac{\alpha_i}{x_i} + \sum_{1 \leqslant i \leqslant n} \beta_i x_i + \sum_{1 \leqslant i, j \leqslant n} \gamma_{ij} \frac{x_i}{x_j},$$

where α_i, β_i and γ_{ij} for all i and j are non-negative constants.

Definition 2. A *unary geometric program* is a geometric program which minimizes a

unary posynomial subject to upper and lower bounds on all variables. In other words, it is a geometric program of the following form:

$$\text{Minimize } u(x_1, \ldots, x_n) = \sum_{1 \leqslant i \leqslant n} \frac{\alpha_i}{x_i} + \sum_{1 \leqslant i \leqslant n} \beta_i x_i + \sum_{1 \leqslant i, j \leqslant n} \gamma_{ij} \frac{x_i}{x_j}$$
$$\text{subject to} \quad L_i \leqslant x_i \leqslant U_i \quad \text{for all } 1 \leqslant i \leqslant n,$$

where α_i, β_i, γ_{ij}, L_i and U_i for all i and j are non-negative constants.

As we show in section 2, the formulation of minimizing weighted component delay is a unary geometric program. For all other formulations, Chen, Chu and Wong [3] showed that they can be reduced by the Lagrangian relaxation technique to problems very similar to weighted component delay problems. We observe that these problems are also unary geometric programs. In other words, by solving unary geometric programs, all formulations of circuit component sizing above can be solved.

To solve unary geometric programs, we present a greedy algorithm which is both optimal and efficient. In particular, for unary geometric programs corresponding to VLSI circuit component sizing, we prove that the runtime of the greedy algorithm is linear to the number of components in the circuit.

The rest of this paper is organized as follows. In section 2, we explain why different formulations of the circuit sizing problem can be reduced to unary geometric programs. In section 3, we present the greedy algorithm to solve unary geometric programs, prove its optimality and analyze its convergence rate. In section 4, we analyze the runtime of the greedy algorithm when applied to VLSI circuits. In section 5, experimental results to show the runtime and storage requirements of our approach to circuit sizing are presented.

2. Reduction to unary geometric programs

In this section, we show that any formulation of circuit component sizing with one of delay, area and power constituting the objective function and with constraints on the other two can be reduced to unary geometric programs.

For a general VLSI circuit, we can ignore all latches and optimize its combinational subcircuits. Therefore, we focus on combinational circuits below. Figure 1 illustrates a combinational circuit. We call a gate or a wire segment a *circuit component*. Let n be the number of component in the circuit. The circuit component sizing problem is to optimize some objective function subject to some constraints involving delay, area and power. For $1 \leqslant i \leqslant n$, let x_i be the gate size if component i is a gate, or the segment width if component i is a wire segment. Let L_i and U_i be, respectively, the lower bound and upper bound on the component size x_i, i.e., $L_i \leqslant x_i \leqslant U_i$.

In section 2.1, we first introduce the model that we use for delay calculation. In section 2.2, we show that the formulation of minimizing a weighted sum of component delays can be written directly as a unary geometric program. In section 2.3, we show that all other formulations can be reduced to unary geometric programs.

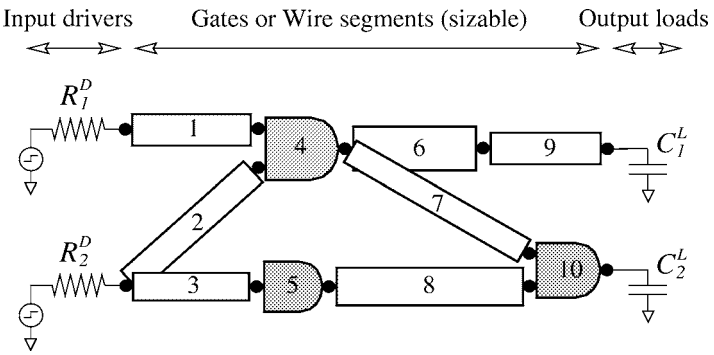

Figure 1. A combinational circuit.

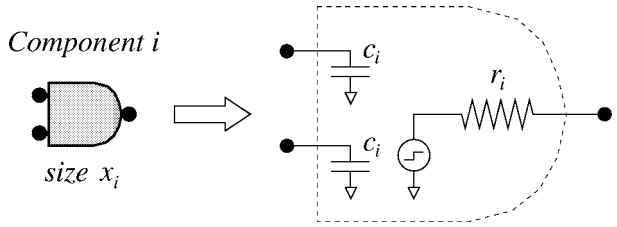

Figure 2. The model of a gate by a switch-level RC circuit. Note that $r_i = \widehat{r}_i/x_i$ and $c_i = \widehat{c}_i x_i + f_i$, where $\widehat{r}_i$, $\widehat{c}_i$ and f_i are the unit size output resistance, the unit size gate area capacitance and the gate perimeter capacitance of the gate i respectively. Although the gate shown here is a 2-input AND gate, the model can be easily generalized for any gate with any number of input pins.

2.1. *Delay model*

For the purpose of delay calculation, we model circuit components as RC circuits. A gate is modeled as a switch-level RC circuit as shown in figure 2. See [22] for a reference of this model. For this model, the output resistance $r_i = \widehat{r}_i/x_i$, and the input capacitance of a pin $c_i = \widehat{c}_i x_i + f_i$, where $\widehat{r}_i$, $\widehat{c}_i$ and f_i are the unit size output resistance, the unit size gate area capacitance and the gate perimeter capacitance of gate i respectively. (To simplify the notations, we assume the input capacitances of all input pins of a gate are the same. We also ignore the intrinsic gate delay. It is clear that all our results will still hold without these assumptions.)

A wire segment is modeled as a π-type RC circuit as shown in figure 3. For this model, the segment resistance $r_i = \widehat{r}_i/x_i$, and the segment capacitance $c_i = \widehat{c}_i x_i + f_i$, where $\widehat{r}_i$, $\widehat{c}_i$ and f_i are the unit width wire resistance, the unit width wire area capacitance and the wire fringing capacitance of segment i respectively. The classical Elmore delay model [11] is used for delay calculation. The delay of each component is equal to the delay associated with its resistor. The delay associated with a resistor is equal to its resistance times its *downstream capacitance*, i.e., the total capacitance driven by the resistor. The delay along a signal path is the sum of the delays associated with the resistors in the path.

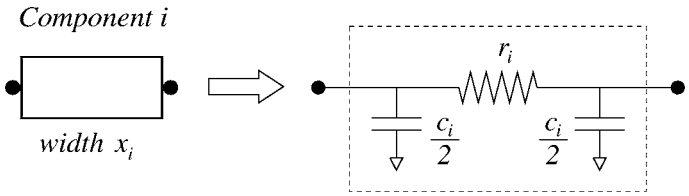

Figure 3. The model of a wire segment by a π-type RC circuit. Note that $r_i = \widehat{r}_i/x_i$ and $c_i = \widehat{c}_i x_i + f_i$, where $\widehat{r}_i$, $\widehat{c}_i$ and f_i are the unit width wire resistance, the unit width wire area capacitance and the wire fringing capacitance of the segment i respectively.

2.2. *Weighted component delay formulation*

In this subsection, we show that the problem of minimizing a weighted sum of the delays of components can be written directly as a unary geometric program.

According to section 2.1, the capacitance of component i is a linear function in x_i and the capacitances of output loads are constants. So the downstream capacitance of each component is a linear function in $x_1, \ldots, x_n$. For example, the downstream capacitance of component 6 is $(\widehat{c}_6 x_6 + f_6)/2 + (\widehat{c}_9 x_9 + f_9) + C_1^L$. Since the resistance of component i is inversely proportional to x_i and the resistance of drivers are constants, the delay associated with each component in the circuit can be written as a unary posynomial in $x_1, \ldots, x_n$ plus a constant. For example,

$$\begin{aligned}\text{Delay of component } 6 &= \frac{\widehat{r}_6}{x_6}\left(\frac{\widehat{c}_6 x_6 + f_6}{2} + (\widehat{c}_9 x_9 + f_9) + C_1^L\right)\\ &= \frac{\widehat{r}_6(f_6/2 + f_9 + C_1^L)}{x_6} + \widehat{r}_6\widehat{c}_9\frac{x_9}{x_6} + \frac{\widehat{r}_6\widehat{c}_6}{2}.\end{aligned}$$

It is clear that a weighted sum of the component delays is also a unary posynomial plus a constant. Together with upper and lower bounds on component sizes, the weighted component delay formulation can be written as a unary geometric program.

2.3. *Other formulations*

Chen, Chu and Wong [3] showed that the Lagrangian relaxation technique can be used to handle different formulations of circuit component sizing. Lagrangian relaxation is a general technique for solving constrained optimization problems. In Lagrangian relaxation, "troublesome" constraints are "relaxed" and incorporated into the objective function after multiplying them by constants called Lagrange multipliers, one multiplier for each constraint. For any fixed vector $\boldsymbol{\lambda}$ of the Lagrange multipliers introduced, we have a new optimization problem (which should be easier to solve because it is free of troublesome constraints) called the Lagrangian relaxation subproblem.

Chen, Chu and Wong [3] showed that there exists a vector $\boldsymbol{\lambda}$ such that the optimal solution of the Lagrangian relaxation subproblem is also the optimal solution of the original circuit component sizing problem. The problem of finding such a vector $\boldsymbol{\lambda}$ is called the Lagrangian dual problem. The Lagrangian dual problem can be solved by the

classical method of subgradient optimization [1]. Therefore, if the Lagrangian relaxation subproblem can be solved optimally, the original circuit sizing problem can also be solved optimally.

For all formulations stated in section 1, the corresponding Lagrangian relaxation subproblems are indeed very similar to a weighted component delay problem. No matter which of delay, area or power is in the objective function and which are in the constraints, after incorporating the constraints into the objective function, the resulting objective function is always a weighted sum of total component area, total power dissipation, component delays and input driver delays.

The total component area, the total power dissipation and the input driver delays are all linear functions in $x_1, \ldots, x_n$. It is obvious for the case of area. For power dissipation, power is dissipated mainly when charging and discharging capacitances in the circuit. Power dissipation is a linear function in the capacitances of components. Since the capacitance of component i is linear to its size x_i, the total power dissipation is a linear function in $x_1, \ldots, x_n$. For input driver delays, note that the resistance of each input driver is a constant and the total capacitance driven by each input driver is a linear function in $x_1, \ldots, x_n$. So the delay associated with each input driver is a linear function in $x_1, \ldots, x_n$.

As a result, for any formulation considered, the objective function of the Lagrangian relaxation subproblem is a unary geometric function. Together with the upper and lower bounds on component sizes, the Lagrangian relaxation subproblem is a unary geometric program.

3. Greedy algorithm for solving unary geometric program

In this section, we present a greedy algorithm which can solve unary geometric programs very efficiently and optimally. In section 3.1, we present the greedy algorithm. In section 3.2, we prove that if we use $(x_1, \ldots, x_n) = (L_1, \ldots, L_n)$ as the starting solution, the algorithm always converges to the optimal solution. In section 3.3, we prove that if ($\alpha_i \neq 0$ or $\beta_i \neq 0$) for all i, then the greedy algorithm will converge linearly to the optimal solution from any starting solution.

3.1. The greedy algorithm

The basic idea of the greedy algorithm is to iteratively adjusting the variables. In each iteration, the variables are examined one by one. When x_k is examined, it is adjusted optimally while keeping the values of all other variables fixed. We call this operation an optimal local adjustment of x_k. The following lemma gives a formula for optimal local adjustment.

Lemma 1. For a solution $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ of a unary geometric program, the optimal local adjustment of x_k is given by

$$x_k = \min\left\{U_k, \max\left\{L_k, \sqrt{\frac{\sum_{1\leqslant i\leqslant n,\, i\neq k} \gamma_{ik} x_i + \alpha_k}{\sum_{1\leqslant j\leqslant n,\, j\neq k} \gamma_{kj}/x_j + \beta_k}}\right\}\right\}.$$

Proof.

$$\begin{aligned} u(x_1, \ldots, x_n) &= \sum_{1\leqslant i\leqslant n} \frac{\alpha_i}{x_i} + \sum_{1\leqslant i\leqslant n} \beta_i x_i + \sum_{1\leqslant i,j\leqslant n} \gamma_{ij}\frac{x_i}{x_j} \\ &= \frac{1}{x_k}\left(\sum_{1\leqslant i\leqslant n,\, i\neq k} \gamma_{ik} x_i + \alpha_k\right) + x_k\left(\sum_{1\leqslant j\leqslant n,\, j\neq k} \gamma_{kj}\frac{1}{x_j} + \beta_k\right) \\ &\quad + \text{terms independent of } x_k. \end{aligned}$$

So by the Kuhn–Tucker conditions [13], the optimal value of x_k between L_k and U_k which minimize $u(x_1, \ldots, x_n)$ is

$$x_k = \min\left\{U_k, \max\left\{L_k, \sqrt{\frac{\sum_{1\leqslant i\leqslant n,\, i\neq k} \gamma_{ik} x_i + \alpha_k}{\sum_{1\leqslant j\leqslant n,\, j\neq k} \gamma_{kj}/x_j + \beta_k}}\right\}\right\}. \qquad \square$$

The greedy algorithm is given below.

Greedy algorithm for unary geometric program.

S1. Let $(x_1, \ldots, x_n)$ be some starting solution.

S2. **for** $k := 1$ to n **do**

$$x_k = \min\left\{U_k, \max\left\{L_k, \sqrt{\frac{\sum_{1\leqslant i\leqslant n,\, i\neq k} \gamma_{ik} x_i + \alpha_k}{\sum_{1\leqslant j\leqslant n,\, j\neq k} \gamma_{kj}/x_j + \beta_k}}.\right\}\right\}$$

S3. Repeat step S2 until no improvement.

3.2. Optimality of the greedy algorithm

In this subsection, we prove that if we use $(x_1, \ldots, x_n) = (L_1, \ldots, L_n)$ as the starting solution, the algorithm always converges to the optimal solution.

Let

$$\boldsymbol{x} = (x_1, \ldots, x_n), \quad A_k(\boldsymbol{x}) = \sum_{1\leqslant i\leqslant n,\, i\neq k} \gamma_{ik} x_i, \quad B_k(\boldsymbol{x}) = \sum_{1\leqslant j\leqslant n,\, j\neq k} \gamma_{kj}\frac{1}{x_j}.$$

Note that $u(\boldsymbol{x})$ is a posynomial in $\boldsymbol{x}$. It is well known that under a variable transformation, a posynomial is equivalent to a convex function. So $u(\boldsymbol{x})$ has a unique global minimum

and no other local minimum. We show in the following two lemmas that with the starting solution $(x_1, \ldots, x_n) = (L_1, \ldots, L_n)$, the greedy algorithm always converges to the global minimum.

Lemma 2. If the greedy algorithm converges, then the solution is optimal.

Proof. Suppose the algorithm converges to $\boldsymbol{x}^* = (x_1^*, \ldots, x_n^*)$. Then for $1 \leqslant k \leqslant n$, by lemma 1,

$$x_k^* = \min\left\{U_k, \max\left\{L_k, \sqrt{\frac{A_k(\boldsymbol{x}^*) + \alpha_k}{B_k(\boldsymbol{x}^*) + \beta_k}}\right\}\right\}.$$

Note that $u(\boldsymbol{x})$ is a posynomial in $\boldsymbol{x}$, and that under the transformation $x_k = \mathrm{e}^{z_k}$ for $1 \leqslant k \leqslant n$, the function $h(\boldsymbol{z}) = u(\mathrm{e}^{z_1}, \ldots, \mathrm{e}^{z_n})$ is convex over $\Omega_z = \{\boldsymbol{z}\colon L_k \leqslant \mathrm{e}^{z_k} \leqslant U_k, 1 \leqslant k \leqslant n\}$. Let $\boldsymbol{z}^* = (z_1^*, \ldots, z_n^*)$ where $x_k^* = \mathrm{e}^{z_k^*}$ for $1 \leqslant k \leqslant n$. We now consider 3 cases:

Case 1: $x_k^* = \sqrt{(A_k(\boldsymbol{x}^*) + \alpha_k)/(B_k(\boldsymbol{x}^*) + \beta_k)}$.

In this case, we have $\frac{\partial u}{\partial x_k}(\boldsymbol{x}^*) = 0$. Thus

$$\frac{\partial h}{\partial z_k}(\boldsymbol{z}^*) = \frac{\partial u}{\partial x_k}(\boldsymbol{x}^*)\frac{\partial x_k}{\partial z_k}(\boldsymbol{z}^*) = \frac{\partial u}{\partial x_k}(\boldsymbol{x}^*)\mathrm{e}^{z_k^*} = 0.$$

Case 2: $x_k^* = L_k$.

In this case, $L_k \geqslant \sqrt{(A_k(\boldsymbol{x}^*) + \alpha_k)/(B_k(\boldsymbol{x}^*) + \beta_k)}$. We have $\frac{\partial u}{\partial x_k}(\boldsymbol{x}^*) \geqslant 0$ and $z_k - z_k^* \geqslant 0, \forall \boldsymbol{z} \in \Omega_z$. Hence

$$\frac{\partial h}{\partial z_k}(\boldsymbol{z}^*)(z_k - z_k^*) = \frac{\partial u}{\partial x_k}(\boldsymbol{x}^*)\mathrm{e}^{z_k^*}(z_k - z_k^*) \geqslant 0, \quad \forall \boldsymbol{z} \in \Omega_z.$$

Case 3: $x_k^* = U_k$.

In this case, $U_k \leqslant \sqrt{(A_k(\boldsymbol{x}^*) + \alpha_k)/(B_k(\boldsymbol{x}^*) + \beta_k)}$. We have $\frac{\partial u}{\partial x_k}(\boldsymbol{x}^*) \leqslant 0$ and $z_k - z_k^* \leqslant 0, \forall \boldsymbol{z} \in \Omega_z$. Hence

$$\frac{\partial h}{\partial z_k}(\boldsymbol{z}^*)(z_k - z_k^*) = \frac{\partial u}{\partial x_k}(\boldsymbol{x}^*)\mathrm{e}^{z_k^*}(z_k - z_k^*) \geqslant 0, \quad \forall \boldsymbol{z} \in \Omega_z.$$

So $\frac{\partial h}{\partial z_k}(\boldsymbol{z}^*)(z_k - z_k^*) \geqslant 0$ for all k and for all $\boldsymbol{z} \in \Omega_z$. Thus for any feasible solution $\boldsymbol{x}$,

$$\begin{aligned} u(\boldsymbol{x}) - u(\boldsymbol{x}^*) &= h(\boldsymbol{z}) - h(\boldsymbol{z}^*) \\ &\geqslant \nabla h(\boldsymbol{z}^*)(\boldsymbol{z} - \boldsymbol{z}^*) \quad \text{as } h \text{ is convex} \\ &= \sum_{k=1}^{n} \frac{\partial h}{\partial z_k}(\boldsymbol{z}^*)(z_k - z_k^*) \\ &\geqslant 0. \end{aligned}$$

Therefore $\boldsymbol{x}^*$ is the global minimum point. □

Lemma 3. If $(x_1, \ldots, x_n) = (L_1, \ldots, L_n)$ is used as the starting solution, the greedy algorithm always converges.

Proof. For any two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, we use $\boldsymbol{x} \leqslant \boldsymbol{y}$ to denote that $x_i \leqslant y_i$ for all i.

Consider any two feasible solutions $\boldsymbol{x}$ and $\boldsymbol{y}$. Let $\boldsymbol{x}'$ and $\boldsymbol{y}'$ be the solutions after locally adjusting some variable x_k of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. If $\boldsymbol{x} \leqslant \boldsymbol{y}$, then $A_k(\boldsymbol{x}) \leqslant A_k(\boldsymbol{y})$ and $B_k(\boldsymbol{x}) \geqslant B_k(\boldsymbol{y})$. So

$$x_k' = \min\left\{U_k, \max\left\{L_k, \sqrt{\frac{A_k(\boldsymbol{x}) + \alpha_k}{B_k(\boldsymbol{x}) + \beta_k}}\right\}\right\}$$
$$\leqslant \min\left\{U_k, \max\left\{L_k, \sqrt{\frac{A_k(\boldsymbol{y}) + \alpha_k}{B_k(\boldsymbol{y}) + \beta_k}}\right\}\right\} = y_k'.$$

Also, $x_j' = x_j \leqslant y_j = y_j'$ for $j \neq k$. Hence $\boldsymbol{x} \leqslant \boldsymbol{y}$ implies $\boldsymbol{x}' \leqslant \boldsymbol{y}'$.

If we consider $\boldsymbol{x}$ and $\boldsymbol{y}$ be solutions before two consecutive optimal local adjustment operations, then $\boldsymbol{x}' = \boldsymbol{y}$. Therefore, $\boldsymbol{x} \leqslant \boldsymbol{x}' = \boldsymbol{y} \leqslant \boldsymbol{y}'$. Since the starting solution is $(L_1, \ldots, L_n)$, we can prove by mathematical induction that all variables are monotonically increasing after each optimal local adjustment operation.

If we consider $\boldsymbol{y}$ to be the optimal solution, then $\boldsymbol{y} = \boldsymbol{y}'$. Hence $\boldsymbol{x} \leqslant \boldsymbol{y}$ implies $\boldsymbol{x}' \leqslant \boldsymbol{y}$. Since the starting solution is $(L_1, \ldots, L_n)$, we can prove by mathematical induction that every x_i after each optimal local adjustment operation is upper bounded by the optimal value y_i.

As x_i is monotonically increasing and is upper bounded for all i, the greedy algorithm always converges. □

By lemmas 2 and 3, we have the following theorem.

Theorem 1. For any unary geometric program, if $(x_1, \ldots, x_n) = (L_1, \ldots, L_n)$ is used as the starting solution, the greedy algorithm always converges to the optimal solution.

3.3. Convergence rate of the greedy algorithm

In section 2, we show that many formulations of the circuit sizing problem can be reduced to a sequence of unary geometric programs by Lagrangian relaxation. We prove in section 3.2 the convergence of the greedy algorithm only for the special starting solution $\boldsymbol{x} = (L_1, \ldots, L_n)$. So in order to guarantee convergence, before solving each unary geometric program instance, all variables have to be reset to their lower bounds to form the starting solution for the greedy algorithm. However, since two consecutive unary geometric program instances by Lagrangian relaxation are almost the same (except that the Lagrange multipliers are changed by a little bit), the optimal solution of the first unary geometric program is close to the optimal solution of the second one, and hence a good starting solution to the second one. So if we can guarantee convergence to the optimal solution, it is better not to reset the solution before solving each unary

geometric program instance. We observe that not reseting can speed up the greedy algorithm by more than 50% in practice. In addition, even for the special starting solution, the convergence rate of the greedy algorithm is not known.

In this subsection, we consider unary geometry programs satisfying the condition that $\alpha_i \neq 0$ or $\beta_i \neq 0$ for all i. We point out in section 4 that for VLSI circuit component sizing problems, this condition is essentially always true. Under this condition, we prove that the greedy algorithm always converges to the optimal solution for any starting solution. Moreover, we prove that the convergence rate for any starting solution is always linear with convergence ratio upper bounded by the parameter σ defined as follows:

$$\sigma = \max_{1 \leqslant k \leqslant n} \left\{ \frac{\phi_k + \theta_k}{2} \right\},$$

where

$$\phi_k = 1 \Big/ \left(1 + \frac{\alpha_k}{A_k(U_1, \ldots, U_n)} \right) \quad \text{and} \quad \theta_k = 1 \Big/ \left(1 + \frac{\beta_k}{B_k(L_1, \ldots, L_n)} \right).$$

Note that for all k, at least one of α_k and β_k is positive. So at least one of ϕ_k and θ_k is less than 1. Therefore, it is clear that σ is a constant such that $0 < \sigma < 1$.

Lemma 4 gives bounds on the changes of the variables after each iteration of the greedy algorithm. Let $\boldsymbol{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})$ be the starting solution, and for $t \geqslant 1$, let $\boldsymbol{x}^{(t)} = (x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)})$ be the solution just after the tth iteration of the greedy algorithm. Let $\Delta = \max_{1 \leqslant i \leqslant n} \{(U_i - L_i)/L_i\}$.

Lemma 4. For any $t \geqslant 0$,

$$\frac{1}{1 + \Delta\sigma^t} \leqslant \frac{x_i^{(t+1)}}{x_i^{(t)}} \leqslant 1 + \Delta\sigma^t \quad \text{for all } i.$$

The proof of lemma 4 is given in the appendix.

Theorem 2. If $\alpha_i \neq 0$ or $\beta_i \neq 0$ for all i, then the greedy algorithm always converges to the optimal solution for any starting solution.

Proof. Since $0 < \sigma < 1$, $1 + \Delta\sigma^t \to 1$ as $t \to \infty$. So by lemma 4, it is obvious that the greedy algorithm always converges for any starting solution. Lemma 2 proves that if the greedy converges, then the solution is optimal. So the theorem follows. □

Let $\boldsymbol{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)$ be the optimal solution. The following lemma proves that the convergence rate of the greedy algorithm is linear with convergence ratio upper bounded by σ.

Lemma 5. For any $t \geqslant 0$,

$$\left|\frac{x_i^* - x_i^{(t)}}{x_i^*}\right| \leqslant \frac{(1+\Delta)\Delta\sigma^t}{1-\sigma}$$

for all i.

Proof. For any $t \geqslant 0$ and for any i,
Case 1: $(1+\Delta)\sigma^t/(1-\sigma) \geqslant 1$.
Then

$$\frac{x_i^{(t)}}{x_i^*} \leqslant \frac{U_i}{L_i} \leqslant 1 + \Delta \leqslant 1 + \Delta\frac{(1+\Delta)\sigma^t}{1-\sigma}.$$

Similarly, we can prove

$$\frac{x_i^{(t)}}{x_i^*} \geqslant \frac{1}{1+(1+\Delta)\Delta\sigma^t/(1-\sigma)}.$$

Case 2: $(1+\Delta)\sigma^t/(1-\sigma) < 1$.
Then

$$\frac{x_i^{(t)}}{x_i^*} = \prod_{k=t}^{\infty}\frac{x_i^{(k)}}{x_i^{(k+1)}}.$$

So by lemma 4, $1/P \leqslant x_i^{(t)}/x_i^* \leqslant P$ where $P = \prod_{k=t}^{\infty}(1+\Delta\sigma^k)$.

$$\begin{aligned}
\ln P &= \sum_{k=t}^{\infty}\ln\bigl(1+\Delta\sigma^k\bigr) \\
&= \sum_{k=t}^{\infty}\left(\Delta\sigma^k - \frac{1}{2}\Delta^2\sigma^{2k} + \frac{1}{3}\Delta^3\sigma^{3k} - \frac{1}{4}\Delta^4\sigma^{4k} + \cdots\right) \qquad (1)\\
&\leqslant \sum_{k=t}^{\infty}\left(\sum_{j=1}^{\infty}\frac{1}{j}\Delta^j\sigma^{jk}\right) \\
&= \sum_{j=1}^{\infty}\frac{\Delta^j}{j}\sum_{k=t}^{\infty}\bigl(\sigma^j\bigr)^k \\
&= \sum_{j=1}^{\infty}\frac{\Delta^j}{j}\frac{\sigma^{jt}}{1-\sigma^j} \\
&\leqslant \sum_{j=1}^{\infty}\frac{\Delta^j}{j}\frac{\sigma^{jt}}{(1-\sigma)^j} \qquad (2)\\
&= \ln\frac{1}{1-\Delta\sigma^t/(1-\sigma)}, \qquad (3)
\end{aligned}$$

where (1) is because $\ln(1+x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \cdots$, (2) is because $0 < \sigma < 1$, which implies $0 < (1-\sigma)^j < 1 - \sigma < 1 - \sigma^j$ for $j \geqslant 1$, and (3) is because $0 < \Delta\sigma^t/(1-\sigma) < (1+\Delta)\sigma^t/(1-\sigma) < 1$ and $\ln \frac{1}{1-x} = x + \frac{1}{2}x^2 + \frac{1}{3}x^3 + \cdots$ if $0 < x < 1$.

So

$$\begin{aligned} P &\leqslant \frac{1}{1 - \Delta\sigma^t/(1-\sigma)} \\ &= 1 + \frac{\Delta\sigma^t}{1-\sigma} \Big/ \left(1 - \frac{\Delta\sigma^t}{1-\sigma}\right) \\ &\leqslant 1 + \frac{\Delta\sigma^t}{1-\sigma} \Big/ \left(1 - \frac{\Delta}{1+\Delta}\right) \\ &= 1 + \frac{(1+\Delta)\Delta\sigma^t}{1-\sigma}. \end{aligned}$$

Hence

$$\frac{1}{1 + (1+\Delta)\Delta\sigma^t/(1-\sigma)} \leqslant \frac{x_i^{(t)}}{x_i^*} \leqslant 1 + \frac{(1+\Delta)\Delta\sigma^t}{1-\sigma}.$$

Therefore for both cases,

$$\frac{1}{1 + (1+\Delta)\Delta\sigma^t/(1-\sigma)} \leqslant \frac{x_i^{(t)}}{x_i^*} \leqslant 1 + \frac{(1+\Delta)\Delta\sigma^t}{1-\sigma}.$$

It is easy to see that

$$1 - \frac{(1+\Delta)\Delta\sigma^t}{1-\sigma} \leqslant \frac{1}{1 + (1+\Delta)\Delta\sigma^t/(1-\sigma)}.$$

So for any $t \geqslant 0$ and for all i,

$$\left|\frac{x_i^* - x_i^{(t)}}{x_i^*}\right| \leqslant \frac{(1+\Delta)\Delta\sigma^t}{1-\sigma}. \qquad \square$$

4. Analysis of the greedy algorithm when applied to VLSI circuits

In this section, we analyze the runtime of the greedy algorithm when applied to VLSI circuits. In section 4.1, we show that for VLSI circuits, each iteration of the greedy algorithm only takes time linear to the number of circuit components. In section 4.2, we show that for VLSI circuits, the condition $\alpha_i \neq 0$ or $\beta_i \neq 0$ for all i in section 3.3 is always true. So we conclude that for any circuit component sizing formulation, the Lagrangian relaxation subproblem can be solved optimally by the greedy algorithm in time linear to the size of the circuit. Notice that Chu and Wong [5] also showed that the runtime of a similar greedy algorithm for wiring sizing of a single interconnect tree is linear.

4.1. Linear time for each iteration

We first show that when applied to VLSI circuits, each iteration of the greedy algorithm only takes linear time. For each optimal local adjustment operation of x_k, we need to calculated

$$A_k(\boldsymbol{x}) = \sum_{1 \leqslant i \leqslant n,\, i \neq k} \gamma_{ik} x_i \quad \text{and} \quad B_k(\boldsymbol{x}) = \sum_{1 \leqslant j \leqslant n,\, j \neq k} \gamma_{kj} \frac{1}{x_j}.$$

Hence each optimal local adjustment operation takes $O(n)$ time and each iteration takes $O(n^2)$ time in general.

However, for VLSI circuits, $A_k(\boldsymbol{x})$'s and $B_k(\boldsymbol{x})$'s can be computed incrementally. The reason is for any component k, $A_k(\boldsymbol{x})$ is a weighted downstream capacitance and $B_k(\boldsymbol{x})$ is a weighted upstream resistance of the component. So $A_k(\boldsymbol{x})$ can be computed easily by finding a weighted sum of $A_j(\boldsymbol{x})$ over all component j at the output of component k. Similarly, $B_k(\boldsymbol{x})$ can be computed easily by finding a weighted sum of $B_j(\boldsymbol{x})$ over all component j at the input of component k. Note that the number of inputs and number of outputs for VLSI circuits are always bounded by a smaller constant in practice. If we perform the optimal local adjustment operations in a topological order, for each k, both $A_k(\boldsymbol{x})$ and $B_k(\boldsymbol{x})$ can be computed in constant time. Therefore, the optimal local adjustment of x_k can be done in constant time. As a result, each iteration of the greedy algorithm only takes linear time.

4.2. Convergence ratio of the greedy algorithm

In section 3.3, we prove that the greedy algorithm converges linearly with convergence ratio σ to the optimal solution from any starting solution. The convergence ratio σ is upper bounded by the maximum of $(\phi_k+\theta_k)/2$ among all k. So if both $\phi_k = 1$ and $\theta_k = 1$ for some k, then the proof cannot guarantee the convergence of the greedy algorithm. This situation occurs when $\alpha_k = 0$ and $\beta_k = 0$ for some k. On the other hand, if $\alpha_k \neq 0$ or $\beta_k \neq 0$ for all k, then σ is less than 1. The convergence of the greedy algorithm is guaranteed. Moreover, the larger the values of α_k's and β_k's, the faster the convergence of the greedy algorithm. For all k, α_k and β_k are, respectively, the coefficients of the terms $1/x_k$ and x_k in the objective function of the unary geometric program. For VLSI circuit component sizing, α_k's and β_k's are essentially always non-zero. Factors causing α_k's and β_k's to be greater than zero are listed below.

Wire fringing capacitance and gate perimeter capacitance. The Elmore delay for a component is equal to its resistance times its downstream capacitance. Notice that wire fringing capacitance and gate perimeter capacitance are independent of the component sizes, whereas the resistance of any component is inversely proportional to its size. So the wire fringing capacitance of all wire segments and the gate perimeter capacitance of all gates/loads at the downstream of component k contribute to the value of α_k.

Driver resistance and load capacitance. The Elmore delay for a driver equals to the driver resistance times the total capacitance of wire segments and gates driven by it. Since the driver resistance is independent of $x_1, \ldots, x_n$, and the total capacitance of wire segments and gates driven is a linear function of $x_1, \ldots, x_n$, the driver resistance contributes to the value of β_k for all component k driven by the driver. Similarly, if any component k is in the upstream of a load with capacitance C_L, then the term $C_L\widehat{r}_k/x_k$ will occur in the Elmore delay expression. Therefore, the load capacitance contributes to the value of α_k.

Component area. For any VLSI circuit sizing formulation involving the total component area, $\beta_k \neq 0$ for all k. Let the total component area be $\sum_{i=1}^{n} \omega_i x_i$ for some positive constants $\omega_1, \ldots, \omega_n$. If the total component area is the objective to minimize, then the objective function of the unary geometric program will contain the term $\sum_{i=1}^{n} \omega_i x_i$. If the total component area is constrained, then after Lagrangian relaxation, the objective function of the unary geometric program will contain the term $\lambda(\sum_{i=1}^{n} \omega_i x_i)$ where λ is the Lagrange multiplier. In both cases, $\beta_k \neq 0$ for all k.

Power dissipation. As stated in section 2.3, the power dissipation of a circuit is a linear function $\sum_{i=1}^{n} \omega_i x_i$ for some positive constants $\omega_1, \ldots, \omega_n$. So if power dissipation is considered either in the objective or as a constraint, $\beta_k \neq 0$ for all k.

In fact, the number of iterations of the greedy algorithm is a function of the convergence ratio σ. The value of σ depends on a lot of factors like the electrical parameters of the fabrication technology, the resistance of drivers and the capacitance of loads, and the upper and lower bounds on the component sizes. However, we observe that the actual convergence ratio is not very sensitive to those factors, and is usually much less than 0.1 in practice. In addition, the change in σ does not affect the number of iterations very much. For example, if σ changes from 0.05 to an unrealistically large value 0.5, the number of iterations is increased only by a factor of $\log 0.05/\log 0.5 = 4.3$.

Since the convergence rate of the greedy algorithm is linear and the runtime of each iteration is $O(n)$, we have the following theorem.

Theorem 3. When applied to VLSI circuit component sizing, the total runtime of the greedy algorithm for any starting solution is $O(n \log(1/\varepsilon))$, where ε specifies the precision of the final solution (i.e., for the optimal solution $\boldsymbol{x}^*$, the final solution $\boldsymbol{x}$ satisfies $|(x_i^* - x_i)/x_i^*| \leqslant \varepsilon$ for all i).

Proof. By lemma 5, for any $t \geqslant 0$ and for all i,

$$\left| \frac{x_i^* - x_i^{(t)}}{x_i^*} \right| \leqslant \frac{(1+\Delta)\Delta\sigma^t}{1-\sigma}.$$

In order to guarantee that $|(x_i^* - x_i^{(t)})/x_i^*| \leqslant \varepsilon$ for all i, the number of iterations t must satisfy

$$\frac{(1+\Delta)\Delta\sigma^t}{1-\sigma} \leqslant \varepsilon,$$

or equivalently,

$$t \geqslant \log_{1/\sigma} \frac{(1+\Delta)\Delta}{(1-\sigma)\varepsilon}.$$

In other words, at most

$$\left\lceil \log_{1/\sigma} \frac{(1+\Delta)\Delta}{(1-\sigma)\varepsilon} \right\rceil$$

iterations are enough. Since each iteration of the greedy algorithm takes $O(n)$ time, the total runtime is $O(n \log(1/\varepsilon))$. □

Therefore, to obtain a solution with any fixed precision, only a constant number of iterations of the greedy algorithm are needed. This implies that for Lagrangian relaxation subproblems of VLSI circuit component sizing, the runtime of the greedy algorithm is $O(n)$.

5. Experimental results

In this section, the runtime and storage requirements of our unary-geometric-program based approach to circuit component sizing are presented. We implemented a circuit component sizing program for minimizing area subject to maximum delay bound on a PC with a 333 MHz Pentium II processor. In the program, the Lagrangian relaxation technique is used to reduce the problem to unary geometric programs, which are then solved optimally by our greedy algorithm. The Lagrangian dual problem is solved by the classical subgradient optimization method. We test our circuit component sizing program on adders [14] of different sizes ranging from 8 bits to 1024 bits. Number of gates range from 120 to 15360. Number of wires range from 96 to 12288 (note that the number of wires here means the number of sizable wire segments). The total number of sizable components range from 216 to 21648. The lower bound and upper bound of the size of each gate are 1 and 100, respectively. The lower bound and upper bound of the width of each wire are 1 and 3 μm, respectively. The stopping criteria of our program is the solution is within 1% of the optimal solution.

In table 1, the runtime and storage requirements of our program are shown. Even for a circuit with 27648 sizable components, the runtime and storage requirements of our program are 11.53 minutes and about 23 MB only. As mentioned in section 1, the interior-point-method based approach in [21] is the best previous algorithm that can guarantee exact circuit sizing solutions. The largest test circuit in [21] has 832 transistors and the reported runtime and memory are 9 hours (on a Sun SPARCstation 1) and

Table 1
The runtime and storage requirements of our circuit component sizing program on test circuits of different sizes.

Circuit name	Circuit size			Runtime	Memory
	# Gates	# Wires	Total	(minutes)	(MB)
adder (8 bits)	120	96	216	0.00	0.48
adder (16 bits)	240	192	432	0.01	0.76
adder (32 bits)	480	384	864	0.02	1.15
adder (64 bits)	960	768	1728	0.09	1.75
adder (128 bits)	1920	1536	3456	0.28	2.82
adder (256 bits)	3840	3072	6912	0.85	5.37
adder (512 bits)	7680	6144	13824	2.75	11.83
adder (1024 bits)	15360	12288	27648	11.53	22.92

11 MB, respectively. Note that for a problem of similar size (864), our approach only needs 1.3 seconds of runtime (on a PC with a 333 MHz Pentium II processor) and 1.15 MB of memory. According to the SPEC benchmark results [23], our machine is roughly 40 times faster than the slowest model of Sun SPARCstation 1. Taking the speed difference of the machines into account, our program is about 600 times faster than the interior-point-method based approach for a small circuit. For larger circuits, we expect the speedup to be even more significant.

Figures 4 and 5 plot the runtime and storage requirements of our program. By performing a linear regression on the logarithm of the data in figure 4, we find that the empirical runtime of our program is about $O(n^{1.7})$. Figure 5 shows that the ratio of the storage versus the circuit size of our program is close to linear. The storage requirement for each sizable component is about 0.8 KB.

The basic idea of the subgradient optimization method is to repeatedly modify the vector of Lagrange multipliers according to the subgradient direction and then solve the corresponding Lagrangian relaxation subproblem until the solution converges. Figure 6 shows the convergence sequence of the subgradient optimization method for the Lagrangian dual problem on a 128-bit adder. It shows that our program converges smoothly to the optimal solution. The solid line represents the upper bound of the optimal solution and the dotted line represents the lower bound of it. The lower bound values come from the optimal value of the unary geometric program at current iteration. Note that the optimal solution is always inbetween the upper bound and the lower bound. So these curves provide useful information about the distance between the optimal solution and the current solution, and help users to decide when to stop the programs.

Figure 7 shows the area versus delay tradeoff curve of a 16-bit adder. In our experiment, we observe that to generate a new point in the area and delay tradeoff curve, the subgradient optimization method converges in only about 5 iterations. It is because the vector of Lagrange multipliers of the previous point is a good approximation for that of the new point and hence the convergence of the subgradient optimization method is fast. As a result, generating these tradeoff curves requires only a little bit more runtime but provides precious information.

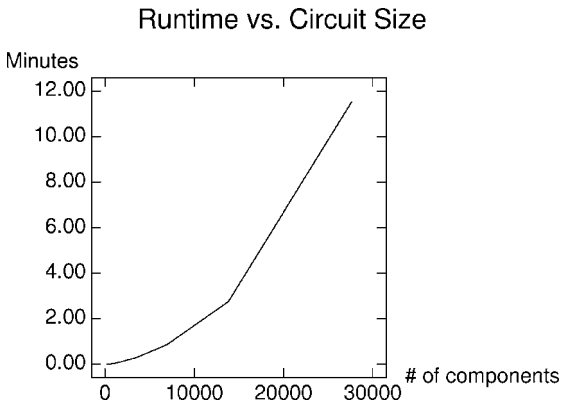

Figure 4. The runtime requirement of our program versus circuit size.

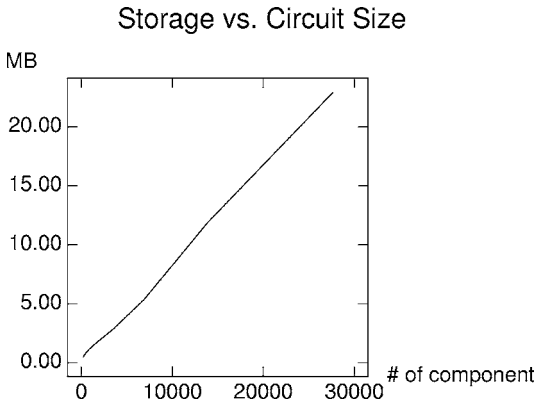

Figure 5. The storage requirement of our program versus circuit size.

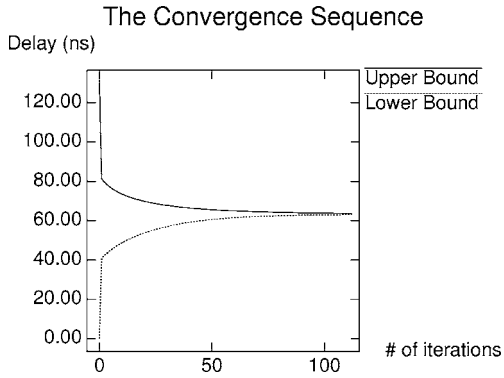

Figure 6. The convergence sequence for a 128-bit adder.

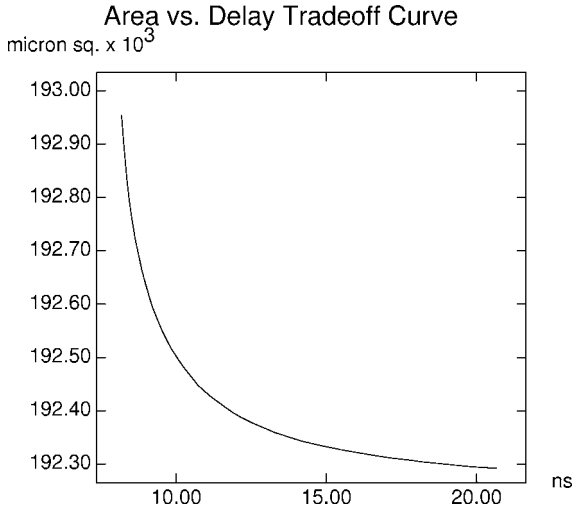

Figure 7. The area versus delay tradeoff curve for a 16-bit adder.

6. Conclusion

We have introduced a special type of geometric program called unary geometric program, which is of the following form:

$$\text{Minimize } u(x_1, \ldots, x_n) = \sum_{1 \leqslant i \leqslant n} \frac{\alpha_i}{x_i} + \sum_{1 \leqslant i \leqslant n} \beta_i x_i + \sum_{1 \leqslant i, j \leqslant n} \gamma_{ij} \frac{x_i}{x_j}$$
$$\text{subject to} \quad L_i \leqslant x_i \leqslant U_i \quad \text{for all } 1 \leqslant i \leqslant n,$$

where α_i, β_i, γ_{ij}, L_i and U_i for all i and j are non-negative constants. We have shown that unary geometric programs are very useful in VLSI circuit component sizing. Many formulations involving delay, area and power can be reduced by the Lagrangian relaxation technique to unary geometric programs.

We have presented a greedy algorithm to solve unary geometric programs optimally and very efficiently. We have proved that the algorithm converges to the optimal solution if x_i is set to L_i for all i in the starting solution. We have also proved that when applied to VLSI circuit component sizing, the algorithm always converges to the optimal solution from any starting solution in time linear to the number of gates or wire segments in the circuit.

Appendix: Proof of lemma 4

In order to prove lemma 4, we need to prove lemmas 6, 7, and 8 first.

For lemmas 6 and 7, we focus on variable x_k for some fixed k. Note that during the n optimal local adjustment operations just before the local adjustment of x_k at a particular iteration (except the first iteration), each variable is adjusted exactly once.

Intuitively, the following two lemmas show that during these n adjustment operations, if the changes in all variables are small, then the change in x_k during the local adjustment of x_k at that iteration will be even smaller.

For some $t \geqslant 1$, let $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{x}' = (x_1', \ldots, x_n')$ and $\boldsymbol{x}'' = (x_1'', \ldots, x_n'')$ be, respectively, the solutions just before the local adjustment of x_k at iteration t, $t+1$ and $t+2$ of the greedy algorithm. Let

$$q_k' = \sqrt{\frac{A_k(\boldsymbol{x}) + \alpha_k}{B_k(\boldsymbol{x}) + \beta_k}} \quad \text{and} \quad q_k'' = \sqrt{\frac{A_k(\boldsymbol{x}') + \alpha_k}{B_k(\boldsymbol{x}') + \beta_k}}.$$

So by lemma 1, $x_k' = \min\{U_k, \max\{L_k, q_k'\}\}$ and $x_k'' = \min\{U_k, \max\{L_k, q_k''\}\}$.

Lemma 6. For any $\rho > 0$, if

$$\frac{1}{1+\rho} \leqslant \frac{x_i'}{x_i} \leqslant 1 + \rho \quad \text{for all } i,$$

then

$$\frac{1}{1+\rho\sigma} \leqslant \frac{q_k''}{q_k'} \leqslant 1 + \rho\sigma.$$

Proof. If $x_i/(1+\rho) \leqslant x_i' \leqslant (1+\rho)x_i$ for all i, we have

$$\frac{1}{1+\rho} A_k(\boldsymbol{x}) \leqslant A_k(\boldsymbol{x}') \leqslant (1+\rho) A_k(\boldsymbol{x})$$

and

$$\frac{1}{1+\rho} B_k(\boldsymbol{x}) \leqslant B_k(\boldsymbol{x}') \leqslant (1+\rho) B_k(\boldsymbol{x}).$$

Since $\gamma_{ik} \geqslant 0$ and $x_i \leqslant U_i$ for all i and k, we have

$$A_k(\boldsymbol{x}) = \sum_{1 \leqslant i \leqslant n,\, i \neq k} \gamma_{ik} x_i \leqslant \sum_{1 \leqslant i \leqslant n,\, i \neq k} \gamma_{ik} U_i = A_k(U_1, \ldots, U_n).$$

So by the definition of ϕ_k, $\phi_k \geqslant 1/\left(1 + \alpha_k / A_k(\boldsymbol{x})\right)$, or equivalently,

$$A_k(\boldsymbol{x}) \leqslant \phi_k\big(A_k(\boldsymbol{x}) + \alpha_k\big).$$

Hence

$$\begin{aligned} A_k(\boldsymbol{x}') + \alpha_k &\leqslant (1+\rho) A_k(\boldsymbol{x}) + \alpha_k \\ &= \rho A_k(\boldsymbol{x}) + \big(A_k(\boldsymbol{x}) + \alpha_k\big) \\ &\leqslant \rho\phi_k\big(A_k(\boldsymbol{x}) + \alpha_k\big) + \big(A_k(\boldsymbol{x}) + \alpha_k\big) \\ &= (1 + \rho\phi_k)\big(A_k(\boldsymbol{x}) + \alpha_k\big) \end{aligned} \tag{4}$$

and

$$\begin{aligned} A_k(\boldsymbol{x}') + \alpha_k &\geqslant \frac{1}{1+\rho} A_k(\boldsymbol{x}) + \alpha_k \\ &= A_k(\boldsymbol{x}) + \alpha_k - \frac{\rho}{1+\rho} A_k(\boldsymbol{x}) \\ &\geqslant A_k(\boldsymbol{x}) + \alpha_k - \frac{\rho\phi_k}{1+\rho}\big(A_k(\boldsymbol{x}) + \alpha_k\big) \\ &= \left(1 - \frac{\rho\phi_k}{1+\rho}\right)\big(A_k(\boldsymbol{x}) + \alpha_k\big) \\ &> \frac{1}{1+\rho\phi_k}\big(A_k(\boldsymbol{x}) + \alpha_k\big) \\ &\quad \text{as } \rho > 0 \text{ and } 0 < \phi_k < 1. \end{aligned} \tag{5}$$

Similarly, since $\gamma_{kj} \geqslant 0$ and $x_j \geqslant L_j$ for all j and k, we have

$$B_k(\boldsymbol{x}) = \sum_{1 \leqslant j \leqslant n,\ j \neq k} \gamma_{kj} \frac{1}{x_j} \leqslant \sum_{1 \leqslant j \leqslant n,\ j \neq k} \gamma_{kj} \frac{1}{L_j} = B_k(L_1, \ldots, L_n).$$

So by the definition of θ_k, $\theta_k \geqslant 1/(1 + \beta_k / B_k(\boldsymbol{x}))$, or equivalently,

$$B_k(\boldsymbol{x}) \leqslant \theta_k\big(B_k(\boldsymbol{x}) + \beta_k\big).$$

Hence we can prove similarly that

$$B_k(\boldsymbol{x}') + \beta_k \leqslant (1 + \rho\theta_k)\big(B_k(\boldsymbol{x}) + \beta_k\big) \tag{6}$$

and

$$B_k(\boldsymbol{x}') + \beta_k > \frac{1}{1+\rho\theta_k}\big(B_k(\boldsymbol{x}) + \beta_k\big). \tag{7}$$

By definitions of q_k' and q_k'', and by (4) and (7), we have

$$\begin{aligned} q_k'' &= \sqrt{\frac{A_k(\boldsymbol{x}') + \alpha_k}{B_k(\boldsymbol{x}') + \beta_k}} \\ &\leqslant \sqrt{(1+\rho\phi_k)(1+\rho\theta_k)\frac{A_k(\boldsymbol{x}) + \alpha_k}{B_k(\boldsymbol{x}) + \beta_k}} \\ &\leqslant \left(1 + \rho\frac{\phi_k + \theta_k}{2}\right) q_k' \quad \text{(as geometric mean} \leqslant \text{arithmetic mean)} \\ &\leqslant (1 + \rho\sigma) q_k'. \end{aligned}$$

Similarly, by (5) and (6), we can prove that

$$q_k'' \geqslant \frac{1}{1 + \rho(\phi_k + \theta_k)/2} q_k' \geqslant \frac{1}{1+\rho\sigma} q_k'.$$

As a result, $1/(1+\rho\sigma) \leqslant q''_k/q'_k \leqslant 1+\rho\sigma$. □

Lemma 7. For any $\rho > 0$, if

$$\frac{1}{1+\rho} \leqslant \frac{x'_i}{x_i} \leqslant 1+\rho \quad \text{for all } i,$$

then

$$\frac{1}{1+\rho\sigma} \leqslant \frac{x''_k}{x'_k} \leqslant 1+\rho\sigma.$$

Proof. By lemma 6, if $x_i/(1+\rho) \leqslant x'_i \leqslant (1+\rho)x_i$ for all i, then $q'_k/(1+\rho\sigma) \leqslant q''_k \leqslant (1+\rho\sigma)q'_k$. By lemma 1, $x'_k = \min\{U_k, \max\{L_k, q'_k\}\}$ and $x''_k = \min\{U_k, \max\{L_k, q''_k\}\}$.

In order to prove $x'_k/(1+\rho\sigma) \leqslant x''_k$, we consider three cases:

Case 1: $q'_k < L_k$.

Then $x'_k = L_k$. So

$$\frac{1}{1+\rho\sigma}x'_k = \frac{1}{1+\rho\sigma}L_k < L_k \leqslant x''_k.$$

Case 2: $q''_k > U_k$.

Then $x''_k = U_k$. So

$$\frac{1}{1+\rho\sigma}x'_k \leqslant \frac{1}{1+\rho\sigma}U_k < U_k = x''_k.$$

Case 3: $q'_k \geqslant L_k$ and $q''_k \leqslant U_k$.

Then $q'_k \geqslant L_k \Rightarrow x'_k \leqslant q'_k$ and $q''_k \leqslant U_k \Rightarrow q''_k \leqslant x''_k$. So

$$\frac{1}{1+\rho\sigma}x'_k \leqslant \frac{1}{1+\rho\sigma}q'_k \leqslant q''_k \leqslant x''_k.$$

In order to prove $x''_k \leqslant (1+\rho\sigma)x'_k$, we consider another three cases:

Case 1: $q'_k > U_k$.

Then $x'_k = U_k$. So $x''_k \leqslant U_k < (1+\rho\sigma)U_k = (1+\rho\sigma)x'_k$.

Case 2: $q''_k < L_k$.

Then $x''_k = L_k$. So $x''_k = L_k < (1+\rho\sigma)L_k \leqslant (1+\rho\sigma)x'_k$.

Case 3: $q'_k \leqslant U_k$ and $q''_k \geqslant L_k$.

Then $q'_k \leqslant U_k \Rightarrow q'_k \leqslant x'_k$ and $q''_k \geqslant L_k \Rightarrow x''_k \leqslant q''_k$. So

$$x''_k \leqslant q''_k \leqslant (1+\rho\sigma)q'_k \leqslant (1+\rho\sigma)x'_k.$$

As a result, $1/(1+\rho\sigma) \leqslant x''_k/x'_k \leqslant 1+\rho\sigma$. □

Lemma 8 gives bounds on the changes of the variables after each iteration of the greedy algorithm. Be reminded that $\boldsymbol{x}^{(t)} = (x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)})$ is the solution just after the tth iteration of the greedy algorithm.

Lemma 8. For any $t \geqslant 0$ and $\rho > 0$, if

$$\frac{1}{1+\rho} \leqslant \frac{x_i^{(t+1)}}{x_i^{(t)}} \leqslant 1+\rho \quad \text{for all } i,$$

then

$$\frac{1}{1+\rho\sigma} \leqslant \frac{x_i^{(t+2)}}{x_i^{(t+1)}} \leqslant 1+\rho\sigma \quad \text{for all } i.$$

Proof. The lemma can be proved by induction on i.
Base case: Consider the variable x_1. Before the local adjustment of x_1, the solutions at iteration $t+1$ and $t+2$ are $(x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)})$ and $(x_1^{(t+1)}, x_2^{(t+1)}, \ldots, x_n^{(t+1)})$, respectively. Since $1/(1+\rho) \leqslant x_i^{(t+1)}/x_i^{(t)} \leqslant 1+\rho$ for all i, by lemma 7, we have $1/(1+\rho\sigma) \leqslant x_1^{(t+2)}/x_1^{(t+1)} \leqslant 1+\rho\sigma$.
Induction step: Assume that the induction hypothesis is true for $i = 1, \ldots, k-1$. Before the local adjustment of x_k, the solutions at iteration $t+1$ and $t+2$ are $(x_1^{(t+1)}, \ldots, x_{k-1}^{(t+1)}, x_k^{(t)}, \ldots, x_n^{(t)})$ and $(x_1^{(t+2)}, \ldots, x_{k-1}^{(t+2)}, x_k^{(t+1)}, \ldots, x_n^{(t+1)})$, respectively. By induction hypothesis,

$$\frac{1}{1+\rho\sigma} \leqslant \frac{x_i^{(t+2)}}{x_i^{(t+1)}} \leqslant 1+\rho\sigma \quad \text{for } i = 1, \ldots, k-1.$$

Hence

$$\frac{1}{1+\rho} \leqslant \frac{x_i^{(t+2)}}{x_i^{(t+1)}} \leqslant 1+\rho \quad \text{for } i = 1, \ldots, k-1,$$

as $\sigma < 1$. Also, it is given that

$$\frac{1}{1+\rho} \leqslant \frac{x_i^{(t+1)}}{x_i^{(t)}} \leqslant 1+\rho \quad \text{for } i = k, \ldots, n.$$

So by lemma 7,

$$\frac{1}{1+\rho\sigma} \leqslant \frac{x_k^{(t+2)}}{x_k^{(t+1)}} \leqslant 1+\rho\sigma.$$

Hence the lemma is proved. □

Proof of lemma 4. This can be proved by induction on t.
Base case: Consider $t = 0$. Note that for any solution $\boldsymbol{x} = (x_1, \ldots, x_n)$, $L_i \leqslant x_i \leqslant U_i$ for all i. For all i,

$$\frac{x_i^{(1)}}{x_i^{(0)}} \leqslant \frac{U_i}{L_i} \leqslant 1 + \frac{U_i - L_i}{L_i} \leqslant 1 + \Delta.$$

Similarly, we can prove that for all i, $x_i^{(1)}/x_i^{(0)} \geqslant 1/(1+\Delta)$.
Induction step: Assume that the induction hypothesis is true for t. Therefore,

$$\frac{1}{1+\Delta\sigma^t} \leqslant \frac{x_i^{(t+1)}}{x_i^{(t)}} \leqslant 1+\Delta\sigma^t \quad \text{for all } i.$$

By lemma 8,

$$\frac{1}{1+\Delta\sigma^{t+1}} \leqslant \frac{x_i^{(t+2)}}{x_i^{(t+1)}} \leqslant 1+\Delta\sigma^{t+1} \quad \text{for all } i.$$

Hence the lemma is proved. □

References

[1] M.S. Bazaraa, H.D. Sherali and C.M. Shetty, *Nonlinear Programming: Theory and Algorithms*, 2nd ed. (Wiley, 1993).
[2] C.-P. Chen, Y.-W. Chang and D.F. Wong, Fast performance-driven optimization for buffered clock trees based on Lagrangian relaxation, in: *Proc. ACM/IEEE Design Automation Conf.* (1996) pp. 405–408.
[3] C.-P. Chen, C.C.N. Chu and D.F. Wong, Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation, IEEE Trans. Computer-Aided Design 18(7) (1999) 1014–1025.
[4] C.-P. Chen, H. Zhou and D.F. Wong, Optimal non-uniform wire-sizing under the Elmore delay model, in: *Proc. IEEE Intl. Conf. on Computer-Aided Design* (1996) pp. 38–43.
[5] C.C.N. Chu and D.F. Wong, Greedy wire-sizing is linear time, IEEE Trans. Computer-Aided Design 18(4) (1999) 398–405.
[6] M.A. Cirit, Transistor sizing in CMOS circuits, in: *Proc. ACM/IEEE Design Automation Conf.* (1987) pp. 121–124.
[7] J. Cong and L. He, An efficient approach to simultaneous transistor and interconnect sizing, in: *Proc. IEEE Intl. Conf. on Computer-Aided Design* (1996) pp. 181–186.
[8] J. Cong and C.-K. Koh, Simultaneous driver and wire sizing for performance and power optimization, in: *Proc. IEEE Intl. Conf. on Computer-Aided Design* (1994) pp. 206–212.
[9] J. Cong and K.-S. Leung, Optimal wiresizing under the distributed Elmore delay model, IEEE Trans. Computer-Aided Design 14(3) (1995) 321–336.
[10] R.J. Duffin, E.L. Peterson and C. Zener, *Geometric Programming – Theory and Application* (Wiley, NY, 1967).
[11] W.C. Elmore, The transient response of damped linear network with particular regard to wideband amplifiers, J. Applied Physics 19 (1948) 55–63.
[12] J.P. Fishburn and A.E. Dunlop, TILOS: A posynominal programming approach to transistor sizing, in: *Proc. IEEE Intl. Conf. on Computer-Aided Design* (1985) pp. 326–328.
[13] D.G. Luenberger, *Linear and Nonlinear Programming*, 2nd ed. (Addison-Wesley, 1984).
[14] M.M. Mano, *Digital Logic and Computer Design* (Prentice-Hall, 1979).
[15] D.P. Marple, Performance optimization of digital VLSI circuits, Technical Report CSL-TR-86-308, Stanford University (October 1986).
[16] D.P. Marple, Transistor size optimization in the Tailor layout system, in: *Proc. ACM/IEEE Design Automation Conf.* (1989) pp. 43–48.
[17] N. Menezes, R. Baldick and L.T. Pileggi, A sequential quadratic programming approach to concurrent gate and wire sizing, in: *Proc. IEEE Intl. Conf. on Computer-Aided Design* (1995) pp. 144–151.

[18] N. Menezes, S. Pullela, F. Dartu and L.T. Pileggi, RC interconnect syntheses-a moment fitting approach, in: *Proc. IEEE Intl. Conf. on Computer-Aided Design* (1994) pp. 418–425.
[19] N. Menezes, S. Pullela and L.T. Pileggi, Simultaneous gate and interconnect sizing for circuit level delay optimization, in: *Proc. ACM/IEEE Design Automation Conf.* (1995) pp. 690–695.
[20] S.S. Sapatnekar, RC interconnect optimization under the Elmore delay model, in: *Proc. ACM/IEEE Design Automation Conf.* (1994) pp. 387–391.
[21] S.S. Sapatnekar, V.B. Rao, P.M. Vaidya and S.M. Kang, An exact solution to the transistor sizing problem for CMOS circuits using convex optimizaiton, IEEE Trans. Computer-Aided Design 12(11) (1993) 1621–1634.
[22] J. Shyu, J.P. Fishburn, A.E. Dunlop and A.L. Sangiovanni-Vincentelli, Optimization-based transistor sizing, IEEE J. Solid-State Circuits 23 (1988) 400–409.
[23] SPEC table, ftp://ftp.cdf.toronto.edu/pub/spectable.