

# An Efficient Method for Large-scale Gate Sizing

Siddharth Joshi and Stephen Boyd, *Fellow, IEEE*

**Abstract**—We consider the problem of choosing the gate sizes or scale factors in a combinational logic circuit in order to minimize the total area, subject to simple RC timing constraints, and a minimum allowed gate size. This problem is well known to be a geometric program (GP), and can be solved using standard interior-point methods for small and medium size problems with up to several thousand gates. In this paper we describe a new method for solving this problem that handles far larger circuits, up to a million gates, and is far faster. Numerical experiments show that our method can compute an adequately accurate solution within around 200 iterations; each iteration, in turn, consists of a few passes over the circuit. In particular, the complexity of our method, with a fixed number of iterations, is linear in the number of gates. A simple implementation of our algorithm can size a 10000 gate circuit in 25 seconds, a 100000 gate circuit in 4 minutes, and a million gate circuit in 40 minutes, approximately. For the million gate circuit, the associated GP has 3 million variables and more than 6 million monomial terms in its constraints; as far as we know, these are the largest GPs ever solved.

**Index Terms**—Gate sizing, geometric programming, large-scale optimization.

## I. INTRODUCTION

WE consider the gate sizing problem, *i.e.*, the problem of choosing scale factors for the gates in a combinational logic circuit in order to minimize area, or any other objective function that is a linear function of the gate scale factors (such as power), subject to some timing requirements. We are given the circuit topology and timing constraints; the variables to be chosen are the gate scale factors, which must exceed some minimum allowed value. The scale factor of a gate affects its delay, area, and input capacitance, and so affects the load capacitance (and therefore also delay) of any gate that drives it. The gate delays, in turn, affect the arrival times of signals, which are subject to given requirements.

In this paper we use a relatively simple, but standard, model for timing analysis, based on an RC model for each gate, and static timing analysis. The same (or equivalent) model is used, for example, in the logical effort method [1], [2]. This timing model is approximate, and even with careful tuning of the model parameters is unlikely to predict actual timing with an accuracy better than 5% or 10%. For this reason, there is no need to solve the gate sizing problem considered here to an accuracy better than 5% or so.

Several methods can be used to achieve higher accuracy (if it is needed). First, our method can be extended to more

complex (and accurate) models that account for differing rising and falling gate delays, the effects of signal slope, and better models of gate and wire load delay, as described in [3]. Another approach to obtaining higher accuracy is to use the method described in this paper to find an initial design, and then use a local optimization method, with accurate models, to fine tune this design. This general approach can also be used to deal with problems in which the gate scale factors are restricted to a finite set of values, instead of being continuously variable, as is assumed here. We first solve the (approximate) problem, using the method described in this paper, ignoring the discrete constraints. We then round the scale factors obtained to valid values. Finally, we use a local method, with accurate timing models, to fine tune the design. In this approach, the RC gate sizing method described in this paper is used as a fast method for getting a good initial condition for a local optimization method (see, *e.g.*, [3]).

The gate sizing problem and variations on it such as wire and device sizing have been studied in many papers, *e.g.*, [4], [5], [6], [7], [8], [9]. Many methods have been proposed to solve, or approximately solve, the gate sizing problem and its variations, *e.g.*, [10], [11], [12]. The most widely known is the logical effort method [1], [2], [13], which gives fast heuristics or design guidelines for approximately solving the gate sizing problem. (The main focus of the logical effort method is, however, giving design insight, and not solving the problem *per se.*) Another approach is based on formulating the gate sizing problem as a *geometric program* (GP), and using a standard interior-point GP solver (*e.g.*, [14], [15], [16]) to obtain the exact solution [17], [18], [19], [3]. These methods can solve the gate scaling problem for circuits with up to 10000 gates, but require effort that typically scales with problem size more than linearly, and so become impractical for larger problems. In this paper we describe a custom method for solving the gate sizing problem, to within a few percent accuracy, that is extremely fast, and handles circuits with up to 1000000 gates (or more). Numerical experiments show that our method can compute an adequately accurate solution within around 200 iterations; each iteration, in turn, consists of a few passes over the circuit. In particular, the complexity of our method, which uses a fixed number of iterations, is linear in the number of gates. A simple implementation of our algorithm can size a 10000 gate circuit in 25 seconds, a 100000 gate circuit in 4 minutes, and a million gate circuit in 40 minutes, approximately.

For the million gate circuit, the associated GP has 3 million variables and more than 6 million monomial terms in its constraints. As far as we know, these are the largest GPs ever solved to date (by any method, for any application). A custom method for solving the  $\ell_1$ -regularized logistic regression problem, which like the gate sizing problem can be cast as a

The authors are with the Department of Electrical Engineering, Stanford University (e-mail: {sidj, boyd}@stanford.edu).

Contact information for S. Joshi (corresponding author): Address: 350 Serra Mall, Packard 243, Stanford CA 94305, USA; Tel: (650) 723-9833; Fax: (650) 723-8473.

Copyright ©2008 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from IEEE by sending an email to pubs-permissions@ieee.org.

GP, is reported in [21]. While the general approach is similar to the method described here at the highest level, all of the critical details differ.

The outline of the paper is as follows. In Section II we give the gate sizing optimization problem. In Section III we consider a variation on the gate sizing optimization problem, when the arrival times are fixed, and give a very efficient method, which we call *nonlinear back substitution*, to solve it exactly. Using this method, we can reduce the general gate sizing problem to an unconstrained optimization problem, where the variables are the arrival times. This problem is not differentiable, however, so in Section IV we introduce a smooth (and convex) approximation. In Section V we describe our algorithm, which is a variation on a *truncated pseudo-Newton method*, which solves the gate sizing optimization problem efficiently. In Section VI we report some numerical results for our method applied to various examples. In Section VII we give a proof of convexity for the unconstrained problem and the approximation; these imply that the solutions found are in fact global.

## II. GATE SIZING

In this section we describe our timing model and the gate sizing problem, and give an overview of our method.

We consider a combinational logic circuit consisting of  $n$  gates, labeled  $1, \dots, n$ . Its topology will be described by a directed acyclic graph, where each node represents a gate and each edge represents an interconnection from the output of a gate to an input of another gate. We let  $m$  denote the number of edges (interconnections). We define the set  $\text{FI}(i)$ , the *fan-in* of gate  $i$ , as

$$\text{FI}(i) = \{j \mid \text{there is a connection from the output of gate } j \text{ to an input of gate } i\},$$

and the set  $\text{FO}(i)$ , the *fan-out* of gate  $i$ , as

$$\text{FO}(i) = \{j \mid \text{there is a connection from the output of gate } i \text{ to an input of gate } j\}.$$

Gate  $i$  is said to *drive* gate  $j$  if  $j \in \text{FO}(i)$ . Gates for which the fan-out is the empty set are called *primary output gates* or just *primary outputs*, and gates for which the fan-in is the empty set are called *primary input gates* or just *primary inputs*. We denote the set of primary output gates by  $\text{PO}$ . We order the gates in such a way that  $\text{FO}(i) \subseteq \{i+1, \dots, n\}$ , for  $i = 1, \dots, n$ . In other words, a gate can only drive another gate with higher index. (This is possible because the topology is a directed acyclic graph.) In addition, we list the primary output gates last:  $\text{PO} = \{\tilde{n}+1, \dots, n\}$ . Thus we have  $\tilde{n}$  gates that not primary outputs.

Each gate has a *size* or *scale factor*  $x_i \in \mathbf{R}$ , with  $x_i \geq 1$ , which denotes the size of the gate relative to the minimum size gate of the same type. A gate with  $x_i = 1$  is called a minimum size gate. The gate sizes  $x_1, \dots, x_n$  will be the design variables in our problem. We let  $x \in \mathbf{R}^n$  be the vector of gate scale factors.

Each gate has an area, which we take to be  $a_i x_i$ , where  $a_i$  is the area of the minimum size gate  $i$ . (In other words, we assume that gate area scales linearly with scale factor.) The

total area is then  $a^T x$ , where  $a$  is the (given, positive) vector of area of the minimum size gates.

Each gate has a *delay*  $D_i$  that depends on its size and the size of the gates in its fan-out. We will use the model

$$D_i = d_i^{\min} + \frac{g_i + \sum_{j \in \text{FO}(i)} F_{ij} x_j}{x_i}, \quad i = 1, \dots, n, \quad (1)$$

where  $d^{\min} \in \mathbf{R}_+^n$ ,  $g \in \mathbf{R}_+^n$ , and  $F \in \mathbf{R}^{n \times n}$  are given problem parameters. Here  $d_i^{\min}$  gives the minimum possible delay of gate  $i$ , obtained only in the limit as  $x_i \rightarrow \infty$ , with all other gates fixed. All entries of the matrix  $F$  are nonnegative; moreover, the sparsity pattern of  $F$  is given by the circuit topology:

$$F_{ij} > 0 \iff j \in \text{FO}(i).$$

As a result of our ordering of gates, the matrix  $F$  is strictly upper triangular, and the number of nonzero entries in  $F$  is the number of interconnections  $m$ . The set of equations (1) can be written in matrix form as

$$D = d^{\min} + \mathbf{diag}(x)^{-1}(Fx + g),$$

where  $\mathbf{diag}(x)$  denotes the diagonal matrix with entries  $x_1, \dots, x_n$ .

Throughout this paper we will use the timing model (1), parametrized by  $d^{\min}$ ,  $g$ , and  $F$ . But we briefly explain here how the standard RC delay model can be put in our form. In the RC model, gate  $i$  is modeled as an RC circuit, with resistance inversely proportional to scale factor, and a total capacitance that has contributions from the gate itself, a wire load, and the input capacitance of gates that it drives. The delay of gate  $i$  is then taken to be the product of the resistance and capacitance (times a constant, typically 0.69, that gives the 50% threshold delay of an RC circuit). We can write this as

$$D_i = \frac{r_i}{x_i} \left( c_i^{\text{int}} x_i + c_i^{\text{wire}} + \sum_{j \in \text{FO}(i)} c_j^{\text{in}} x_j \right), \quad (2)$$

where  $r_i$  is the driving resistance,  $c_i^{\text{int}}$  is the internal capacitance, and  $c_i^{\text{in}}$  is the input capacitance of a minimum size version of gate  $i$ , and  $c_i^{\text{wire}}$  is the wire load capacitance for gate  $i$ . (We have absorbed the constant 0.69 into the constants  $r_i$ .) Our model here assumes that the internal capacitance and input capacitance of a gate scale linearly with scale factor. Although the formula (2) does not distinguish between different input pins of a gate, we could account for this by adding another subscript, that denotes input pin (or driving gate) to  $c_j^{\text{in}}$ . The RC timing model (2) is readily mapped to our model:

$$d_i^{\min} = r_i c_i^{\text{int}}, \quad g_i = r_i c_i^{\text{wire}}, \quad i = 1, \dots, n,$$

$$F_{ij} = \begin{cases} r_i c_j^{\text{in}} & j \in \text{FO}(i), \\ 0 & \text{otherwise.} \end{cases}$$

We now describe the timing of the whole circuit. We define the *arrival time* for gate  $i$  as the latest time at which its signal becomes valid. This is defined recursively, as follows. We define the arrival time of a primary input gate as its delay. The arrival time of gate  $i$  is defined recursively as its own gate delay, plus the largest arrival time of its fan-in gates. Our timing constraint is that the arrival times for

all output gates should not exceed  $T$ , which is the (given) *timing specification* for the whole circuit. (This assumes, for simplicity, that the signals at the inputs of the primary input gates arrive at time 0, and that the timing requirement on all primary output gates is the same. We can readily extend our model to incorporate different arrival times for the primary inputs, and different arrival time requirements for the primary outputs.) The maximum arrival time at the primary output gates (with the signals at the inputs of the primary input gates arriving at time 0) is called the *delay of the circuit*.

It will be convenient for us to work with *upper bounds* on the gate delays and arrival times, instead of their actual values. We let  $d_i$  denote an upper bound on gate delay  $i$ , *i.e.*,  $d_i \geq D_i$ , and we let  $t_i$  denote an upper bound on the arrival time at gate  $i$ . The timing requirements can then be expressed as a set of linear inequalities and equalities:

$$\begin{aligned} t_i - t_j &\geq d_i, & j \in \text{FI}(i), & i = 1, \dots, n, & (3) \\ t_j &= T, & j \in \text{PO}. & \end{aligned}$$

The inequality (3) is interpreted as  $t_i \geq d_i$  if  $\text{FI}(i) = \emptyset$ . If  $t$  and  $d$  satisfy these conditions, and  $d_i \geq D_i$ , then  $t_i$  will be an upper bound on the arrival time at gate  $i$ , and the overall circuit timing constraints will be met. In the sequel we will call  $d_i$  the delay of gate  $i$ , and  $t_i$  the arrival time of gate  $i$ , even though they are really only upper bounds on these quantities. The vector of arrival times  $t$  is also called the *timing assignment*.

Now we can form the gate sizing optimization problem. The objective is to minimize the area of the circuit subject to meeting the timing constraints. Instead of the variables  $d_i$ , we will use variables  $u_i = d_i - d_i^{\min}$ , which give the *additional gate delay*, above the minimum value. (Our motivation for formulating the problem in the variables  $u_i$ , instead of  $d_i$ , will be explained in Section IV-B.) The optimization problem is

$$\begin{aligned} &\text{minimize} && a^T x \\ &\text{subject to} && u_i \geq (g_i + \sum_{j \in \text{FO}(i)} F_{ij} x_j) / x_i, \quad i = 1, \dots, n \\ &&& t_i - t_j - d_i^{\min} \geq u_i, \quad j \in \text{FI}(i), \quad i = 1, \dots, n \\ &&& t_j = T, \quad j \in \text{PO} \\ &&& x_i \geq 1, \quad i = 1, \dots, n, \end{aligned} \quad (4)$$

where the variables are  $x \in \mathbf{R}^n$ ,  $u \in \mathbf{R}^n$ , and  $t \in \mathbf{R}^n$ . The domain of the problem, denoted  $\mathcal{P}$ , is

$$\mathcal{P} = \{(x, u, t) \mid x_i > 0, u_i > 0, t_i \in \mathbf{R}, i = 1, \dots, n\}. \quad (5)$$

The problem parameters are  $a$ ,  $g$ ,  $F$ ,  $d^{\min}$ , and  $T$ . (The circuit topology can be extracted from the matrix  $F$ .) The second inequality in the problem (4) is interpreted as  $t_i - d_i^{\min} \geq u_i$  if  $\text{FI}(i) = \emptyset$ .

The problem (4) has  $3n$  variables,  $3n$  inequality constraints, and  $n - \tilde{n}$  linear equality constraints. The equality constraints are easily eliminated, of course, since each one constrains one variable to a specific value. We can simply consider the variables to be  $x$ ,  $u$ , and  $\tilde{t} = (t_1, \dots, t_{\tilde{n}})$ , and replace  $t_{\tilde{n}+1}, \dots, t_n$  with the (constant)  $T$ .

We note that we can just as well minimize *any* linear function of  $x$ , with positive coefficients, such as power, or some linear combination of area and power.

The problem (4) is a GP (see [19], [3]), and so can be solved (globally) by standard methods. For geometric programming see [25], [26].

#### A. Overview of our approach

Our approach to solve the gate sizing optimization problem consists of several steps. In this section we give a brief overview of the steps involved.

In Section III we describe an efficient method, called *nonlinear back substitution*, to eliminate the variables  $x$  and  $u$ , thereby reducing the problem to an unconstrained optimization problem in the unspecified arrival time  $\tilde{t}$ . Thus, if we are given optimal (nearly optimal) arrival times, we can very efficiently compute the optimal (nearly optimal)  $x$  and  $u$ .

The objective function of the reduced problem is not differentiable because the gate sizing optimization problem (4) has implicit max and min functions. In Section IV, we construct a smooth approximation of the original problem (4), by replacing the max and min functions with soft-max and soft-min functions. When nonlinear back substitution is applied to this approximation, we reduce it a smooth unconstrained optimization problem in the unspecified arrival times. At this point we have a very large, but smooth and convex, unconstrained optimization problem.

To solve this optimization problem efficiently we develop a customized method, which is of the *truncated pseudo-Newton* type, described in Section V. We define an appropriate surrogate for the Hessian of the objective function, and then compute our search direction using an iterative method (pre-conditioned conjugate gradients) to approximately solve the pseudo-Newton equation.

The overall algorithm proceeds as follows. We first initialize the variables using a method described in Section V-A. We use our truncated pseudo-Newton method to approximately minimize the smoothed objective. Finally, we use nonlinear back substitution to obtain the final design. Efficiency depends on two issues: first, we must be able to compute the search directions fast, and second, the search directions must be good enough that the overall problem can be solved in a reasonable number of iterations.

The solution found using this method (or more accurately, would be found if the method were carried out to high accuracy) is in fact a globally optimal solution. This follows from convexity of the original and the smoothed reduced problems, which is shown in Section VII.

### III. NONLINEAR BACK SUBSTITUTION

In this section we show how the gate sizing optimization problem (4) can be transformed to an unconstrained optimization problem in the unspecified arrival times.

The optimization problem (4) can be written as

$$\begin{aligned} &\text{minimize} && a^T x \\ &\text{subject to} && x_i \geq \max \left( (g_i + \sum_{j \in \text{FO}(i)} F_{ij} x_j) / u_i, 1 \right), \\ &&& && i = 1, \dots, n \\ &&& u_i \leq \min_{j \in \text{FI}(i)} (t_i - t_j - d_i^{\min}), \quad i = 1, \dots, n \\ &&& t_j = T, \quad j \in \text{PO}, \end{aligned} \quad (6)$$

with the variables  $x \in \mathbf{R}^n$ ,  $u \in \mathbf{R}^n$ , and  $t \in \mathbf{R}^n$ .

We consider the problem (6) for a fixed timing assignment  $t$ , which is

$$\begin{aligned} & \text{minimize} && a^T x \\ & \text{subject to} && x_i \geq \max \left( (g_i + \sum_{j \in \text{FO}(i)} F_{ij} x_j) / u_i, 1 \right), \\ & && \hspace{15em} i = 1, \dots, n \\ & && u_i \leq \min_{j \in \text{FI}(i)} (t_i - t_j - d_i^{\min}), \quad i = 1, \dots, n, \end{aligned} \quad (7)$$

with the variables  $x$  and  $u$ . Here, the timing assignment  $t$  is considered given problem data, along with  $g$ ,  $F$ ,  $d^{\min}$ , and  $a$ .

The solution to the problem (7) ( $x^*$ ,  $u^*$ ) can be computed efficiently as follows. First,  $u^*$  is computed as

$$u_i^* = \min_{j \in \text{FI}(i)} (t_i - t_j - d_i^{\min}), \quad i = 1, \dots, n. \quad (8)$$

If any  $u_i^*$  is found to be non-positive, we simply terminate as the timing assignment  $t$  is infeasible.

The optimal sizes  $x_i^*$  are given by the backward recursion: for  $i = n, \dots, 1$ ,

$$x_i^* = \max \left( \frac{g_i + \sum_{j \in \text{FO}(i)} F_{ij} x_j^*}{u_i^*}, 1 \right). \quad (9)$$

Note that for any  $i$  the sizes  $x_j^*$  required to calculate  $x_i^*$  have been calculated in the previous steps.

We will now show that the point  $(x^*, u^*)$ , computed by the above procedure, is optimal. Suppose that  $(x^*, u^*)$  is not optimal, and let the optimal point be  $(\tilde{x}, \tilde{u})$ . The point  $(\tilde{x}, \tilde{u})$  has lower objective value, *i.e.*,  $a^T x^* > a^T \tilde{x}$ . Let  $k$  be the largest index such that  $x_k^* \neq \tilde{x}_k$ . If  $\tilde{x}_k < x_k^*$ , then

$$\tilde{u}_k > u_k^* = \min_{j \in \text{FI}(k)} (t_k - t_j - d_k^{\min}),$$

which means there is a  $j \in \text{FI}(k)$  such that  $t_k - t_j - d_k^{\min} < u_k$ , thereby making  $u_k$  infeasible. If  $\tilde{x}_k > x_k^*$ , consider the point  $(\tilde{x}, \tilde{u})$  with  $(\tilde{x}_k, \tilde{u}_k)$  replaced by  $(x_k^*, u_k^*)$ . This point is feasible, since the delay of any other gate can only decrease by decreasing  $x_k$ . The objective value of this point is lower than that of  $(\tilde{x}, \tilde{u})$ . Thus  $(\tilde{x}, \tilde{u})$  cannot be optimal. Therefore  $(x^*, u^*)$  is the optimal point.

The optimal sizes  $x_i^*$  are calculated starting from back of the circuit at the primary output gates, and moving to the primary input gates. We call this procedure *nonlinear back substitution*, because if we consider the problem (7) without the constraints on the minimum gate size, then the solution  $u^*$  is given by (8), and  $x^*$  is given by the upper triangular matrix equation

$$(U - F)x^* = g,$$

where  $U = \text{diag}(u_1^*, \dots, u_n^*)$ . The solution  $x^*$  can be efficiently found by *back substitution*, the standard algorithm for solving a set of upper triangular linear equations: for  $i = n, \dots, 1$ ,

$$x_i^* = \frac{g_i + \sum_{j \in \text{FO}(i)} F_{ij} x_j^*}{u_i^*}.$$

Thus the recursion (9) can be considered a generalization of the standard back substitution algorithm.

### A. Reduced problem

Let the optimal value of the optimization problem (7), as a function of  $t$ , be denoted by  $\psi(t)$ . The domain of the function  $\psi$ ,  $\text{dom } \psi$ , is given by

$$\text{dom } \psi = \mathcal{Q} = \{t \mid t_i - t_j > d_i^{\min}, j \in \text{FI}(i), i = 1, \dots, n\}. \quad (10)$$

If  $\text{FI}(i) = \emptyset$ , the inequality is interpreted as  $t_i > d_i^{\min}$ .

The function  $\psi$  can be viewed as a composition:

$$\psi(t) = a^T x^*(u^*(t)). \quad (11)$$

The function  $u^*$  is given by (8). The function  $x^*$  is a function of  $u^*$ , and is implicitly defined as a backward recursion by (9). The optimization problem (6) (and therefore (4)) reduces to the optimization problem

$$\begin{aligned} & \text{minimize} && \psi(t) \\ & \text{subject to} && t_j = T, \quad j \in \text{PO}, \end{aligned} \quad (12)$$

and further to the unconstrained optimization problem of minimizing  $\psi$  over the unspecified arrival time  $\tilde{t}$ .

The function  $\psi$  is not differentiable because the min and the max functions, that appear in  $u^*$  and  $x^*$ , respectively, are not differentiable. In Section VII we will show that  $\psi$  is a convex function of  $t$ , and therefore a method that finds a local minimum of  $\psi$ , in fact finds a global minimum of  $\psi$ , and thus finds a globally optimal solution of the gate sizing problem.

## IV. SMOOTH APPROXIMATION

In this section we show how to construct a smooth approximation to  $\psi$ , by substituting the max and min functions in (6) with soft-max and soft-min functions, respectively.

### A. Soft-max and soft-min functions

Consider the soft-max function

$$\text{smax}_p(\alpha_1, \dots, \alpha_n) = (\alpha_1^p + \dots + \alpha_n^p)^{(1/p)},$$

where  $p \geq 1$ , and  $\alpha \in \mathbf{R}^n$ . The domain of the soft-max function is

$$\text{dom } \text{smax}_p = \{\alpha \mid \alpha_i \geq 0, i = 1, \dots, n\}.$$

The soft-max function satisfies the inequality

$$\text{smax}_p(\alpha_1, \dots, \alpha_n) \geq \max(\alpha_1, \dots, \alpha_n).$$

We call this property the *conservative* nature of the soft-max function. Further, for any  $\alpha$ , as  $p \rightarrow \infty$  the value of the soft-max function decreases monotonically to the value of the max function. The soft-min function is constructed by applying the soft-max function to inverse of  $\alpha_i$ , *i.e.*,

$$\text{smin}_p(\alpha_1, \dots, \alpha_n) = (\alpha_1^{-p} + \dots + \alpha_n^{-p})^{(-1/p)},$$

where  $p \geq 1$ , and  $\alpha \in \mathbf{R}^n$ . The domain of the soft-max function is

$$\text{dom } \text{smin}_p = \{\alpha \mid \alpha_i > 0, i = 1, \dots, n\}.$$

The soft-min function is also conservative, *i.e.*,

$$\text{smin}_p(\alpha_1, \dots, \alpha_n) \leq \min(\alpha_1, \dots, \alpha_n).$$

We also use the following notation for the soft-min function

$$\text{smin}_p(\alpha_j) = \text{smin}_p(\alpha_1, \dots, \alpha_n),$$

$$j \in \{1, \dots, n\}$$

A soft-max (soft-min) function is characterized by the weight  $p$ .

### B. Smooth approximation

To obtain an approximation to the problem (6) we replace the max and the min functions in the problem (6) with the soft-max (with weight  $p_1 \geq 1$ ) and the soft-min (with weight  $p_2 \geq 1$ ) functions, respectively. The problem so obtained is

$$\begin{aligned} & \text{minimize} && a^T x \\ & \text{subject to} && x_i \geq \text{smax}_{p_1} \left( (g_i + \sum_{j \in \text{FO}(i)} F_{ij} x_j) / u_i, 1 \right), \\ & && \quad \quad \quad i = 1, \dots, n \\ & && u_i \leq \text{smin}_{p_2} (t_i - t_j - d_i^{\min}), \quad i = 1, \dots, n \\ & && \quad \quad \quad j \in \text{FI}(i) \\ & && t_j = T, \quad j \in \text{PO}. \end{aligned} \tag{13}$$

The variables are  $x \in \mathbf{R}^n$ ,  $u \in \mathbf{R}^n$ , and  $t \in \mathbf{R}^n$ . The domain of the problem is  $\mathcal{P}$ , as given in (5).

In Section III, the problem (7) was obtained by considering the problem (6) for fixed arrival times. Similarly, we consider the problem (13) for fixed arrival times. The solution to this problem ( $x^*$ ,  $u^*$ ) can be efficiently obtained by nonlinear back substitution with a slight modification, which is, in (8) the min function is replaced by the soft-min function  $\text{smin}_{p_2}$ , and in (9) the max function is replaced by the soft-max function  $\text{smax}_{p_1}$ . Let the optimal value of the optimization problem (13) for a fixed  $t$  be denoted by  $\psi^s(t)$ . The domain of the function  $\psi^s$ ,  $\text{dom } \psi^s = \mathcal{Q}$  (given in (10)). Like in Section III-A, this reduces the problem (13) to the unconstrained optimization problem of minimizing  $\psi^s$  over the unspecified arrival time  $\tilde{t} = (t_1, \dots, t_n)$ .

Like the function  $\psi$  as shown in (11), the function  $\psi^s$  can be viewed as the composition:  $\psi^s(t) = a^T x^*(u^*(t))$ , where  $u^*$  is defined using the soft-min function, and  $x^*$  is defined using the soft-max function. The function  $\psi^s$  is differentiable because the soft-min and soft-max functions, that appear in the composition, are differentiable.

The function  $\psi^s$  is an approximation to  $\psi$ , and as  $p_1 \rightarrow \infty$ ,  $p_2 \rightarrow \infty$ ,  $\psi^s(t) \rightarrow \psi(t)$  for any  $t$ . Further, for any  $t$ ,  $\psi^s(t) \geq \psi(t)$ . This is due to the conservative nature of the soft-min and the soft-max functions. The  $u_i$  obtained by the soft-min function are smaller than the  $u_i$  obtained by the min function. Decrease in the delays, coupled with the conservative nature of the soft-max function, yields  $x_i$  larger than the one found using the max function and the larger  $u_i$ . Thus for any  $t$  the optimal objective value of the problem (13) for a fixed  $t$ ,  $\psi^s(t)$ , is always greater than or equal to the optimal objective value of the problem (7),  $\psi(t)$ .

In Section VII we will show that  $\psi^s$  is a convex function of  $t$ , and therefore a method that finds a local minimum of  $\psi^s$ , in fact finds a global minimum of  $\psi^s$ , and therefore finds a globally optimal solution of the problem (13).

We can now explain the reason for formulating the gate sizing optimization problem in the variables  $u_i$ , the additional

gate delays, instead of the variables  $d_i$ , the actual gate delays. Formulating the problem (4) in the variables  $d_i$ , and following the same process by which the problem (13) was obtained, will lead to approximating the function  $\min_{j \in \text{FI}(i)} (t_i - t_j)$  by  $\text{smin}_{p_2, j \in \text{FI}(i)} (t_i - t_j)$ ; whereas, in the problem (13), we approximate  $\min_{j \in \text{FI}(i)} (t_i - t_j - d_i^{\min})$  with  $\text{smin}_{p_2, j \in \text{FI}(i)} (t_i - t_j - d_i^{\min})$ . It can be shown that

$$\min_{j \in \text{FI}(i)} (t_i - t_j) \geq \text{smin}_{p_2, j \in \text{FI}(i)} (t_i - t_j - d_i^{\min}) + d_i^{\min} \geq \text{smin}_{p_2, j \in \text{FI}(i)} (t_i - t_j),$$

which means that  $\text{smin}_{p_2, j \in \text{FI}(i)} (t_i - t_j - d_i^{\min}) + d_i^{\min}$  is a better approximation than  $\text{smin}_{p_2, j \in \text{FI}(i)} (t_i - t_j)$  to  $\min_{j \in \text{FI}(i)} (t_i - t_j)$ . If  $d_i^{\min} > 0$  and the number of gates in  $\text{FI}(i)$  is greater than 1, the inequalities can be shown to be strict. Thus, the advantage of formulating the problem in the additional gate delays  $u_i$ , instead of the gate delays  $d_i$ , is that we obtain a better approximation to  $\psi(t)$ .

## V. THE METHOD

We considered several candidate methods for minimizing  $\psi^s$  over the unspecified arrival time  $\tilde{t} = (t_1, \dots, t_n)$ . Simple methods such as gradient or diagonally scaled gradient required far too many iterations to converge; quasi-Newton methods required too much memory.

We developed a custom method of the *truncated pseudo-Newton* type. (See [27, Chapter 6], [28, Chapter 9], for related optimization methods.) The method consists of the following steps:

1. Compute a feasible initial point  $\tilde{t}$ .

### Repeat

- 2a. Compute the gradient  $g$  of the function  $\psi^s$  with respect to  $\tilde{t}$ .
- 2b. Compute a search direction  $\Delta \tilde{t}$  as an approximate solution to the system  $\tilde{H} \Delta \tilde{t} = -g$  by diagonally preconditioned conjugate gradient method.
- 2c. Compute the step size  $s$  by backtracking line search.
- 2d. Update.  $\tilde{t} := \tilde{t} + s \Delta \tilde{t}$ .

**Until** the stopping criteria is satisfied.

3. Use  $\tilde{t}$  to compute the area  $\psi(\tilde{t})$  and gate size  $x$  by nonlinear back substitution.

In step 2b, the matrix  $\tilde{H}$  is a suitable approximation of or surrogate for the Hessian of  $\psi^s$ . We describe each step of the method in detail in the following sections, and, in particular, the surrogate  $\tilde{H}$  for the Hessian of  $\psi^s$ , the preconditioned conjugate gradient (PCG) algorithm implementation to find a search direction  $\Delta \tilde{t}$ .

The method is called *pseudo-Newton* because the matrix  $\tilde{H}$  (in step 2b) is not the Hessian of  $\psi^s$ , but only an approximation; it is called *truncated* because the search direction  $\Delta \tilde{t}$  (in step 2b) is computed by terminating the PCG algorithm after only a few iterations, well before an accurate solution of the system has been computed.

### A. Initial point

In this section we will find a timing assignment  $t \in \mathbf{dom} \psi^s$  such that  $t_{\tilde{n}+1} = T, \dots, t_n = T$ . In the process we will find additional delays  $u_1, \dots, u_n$  such that  $u$  and  $t$  are feasible for the problem (4).

We first consider the circuit with all the gates operating at their minimum delays, *i.e.*, the additional delays  $u_i = 0$ ,  $i = 1, \dots, n$ . The arrival times  $t_i$  are calculated by a forward recursion: for  $i = 1, \dots, n$ ,

$$t_i = \max_{j \in \text{FI}(i)} t_j + d_i^{\min}.$$

The delay of the circuit, with all the gates operating at their minimum delay,  $T^{\min}$  is

$$T^{\min} = \max_{j \in \text{PO}} t_j.$$

A timing constraint  $T$  on the circuit is feasible if and only if  $T > T^{\min}$ , therefore  $T^{\min}$  is called the *minimum circuit delay*. The minimum circuit delay is achieved in the limit as  $x_i/x_{i+1} \rightarrow \infty$ ,  $i = 1, \dots, n-1$  and  $x_n \rightarrow \infty$ . (Since we order the gates so that  $j \in \text{FO}(i) \Rightarrow j > i$ , the above conditions imply that  $x_i/x_j \rightarrow \infty$ ,  $j \in \text{FO}(i)$ ,  $i = 1, \dots, n$ , and  $x_k \rightarrow \infty$ ,  $k \in \text{PO}$ .)

The critical slack of gate  $i$ , denoted  $s_i$ , is the maximum additional delay of gate  $i$ ,  $u_i$ , such that when gate  $i$  has delay  $d_i^{\min} + u_i$  and all other gates have their minimum delays, the delay of the circuit is less than or equal to  $T$ . For a feasible timing  $T$ , the critical slack of gate  $i$  satisfies  $s_i \geq (T - T^{\min}) > 0$ ,  $i = 1, \dots, n$ .

A *path* in the circuit is a sequence of gates for which each gate in the sequence is in the fan-in of the next gate in the sequence. The *length* of a path is the number of gates in the sequence. A *longest path* through a gate is a path among all the paths that contain the gate for which the length of the path is greater than or equal to any other path that contains the gate. Let the length of a longest path through gate  $i$  be  $l_i$ ,  $i = 1, \dots, n$ .

We set  $u_i$  to be

$$u_i = s_i/l_i, \quad i = 1, \dots, n.$$

The arrival times are calculated by the forward recursion: for  $i = 1, \dots, n$ ,

$$t_i = \max_{j \in \text{FI}(i)} t_j + u_i + d_i^{\min}. \quad (14)$$

As the critical slacks are positive, the additional delays are positive, and the arrival time satisfies  $t \in \mathbf{dom} \psi^s$ . Further, the delay along any path is less than or equal to  $T$  because the additional delay of any gate along the path is less than or equal to  $T - T^{\min}$  divided by the length of the path.

Finally, to obtain a feasible point for the problem (4), we set the arrival times for all primary output gates to be  $T$ . This only can increase the arrival times of the primary output gates, and thus  $t \in \mathbf{dom} \psi^s$  with  $t_{\tilde{n}+1} = T, \dots, t_n = T$ . Therefore we have a feasible initial point  $\tilde{t}$  for the unconstrained optimization problem.

We refer to a recursion, *e.g.*, (14), as a *pass over the circuit*, in particular, a forward recursion of the type (14) as forward

pass, and a backward recursion of the type (9) as a backward pass. The critical slack and the length of a longest path through the gate, can be found for all gates by carrying out two passes over the circuit (one forward and one backward), each requiring a very modest number of computations per interconnection.

### B. Gradient

In this section we will show how to compute the gradient of the function  $\psi^s$  efficiently. To simplify notation we write  $x^*$ ,  $u^*$  as  $x$ ,  $u$ , respectively in this section.

Like the function  $\psi$  as given in (11), the function  $\psi^s$  is a composition:  $\psi^s(t) = a^T x(u(t))$ . The function  $u$  is given by (8) with the min replaced by  $\text{smin}_{p_2}$ , which is

$$u_i^{-p_2} = \sum_{j \in \text{FI}(i)} (t_i - t_j - d_i^{\min})^{-p_2}, \quad i = 1, \dots, n. \quad (15)$$

The function  $x$  is implicitly defined by the recursion (9), with the max replaced by  $\text{smax}_{p_1}$ . With a little rearrangement of terms we can write the function  $x$  as

$$u_i^{p_1} = \frac{(g_i + \sum_{j \in \text{FO}(i)} F_{ij} x_j)^{p_1}}{x_i^{p_1} - 1}, \quad i = 1, \dots, n. \quad (16)$$

The set of equations (16) show that  $u$  can be written explicitly as a function of  $x$ .

Let

$$\phi = \psi^s(t) = a^T x(u(t)).$$

To find the gradient of  $\nabla \psi^s = \nabla_t \phi$ , we apply the chain rule. Applying the chain rule to  $u$  as a function of  $t$  gives

$$\nabla_t \phi = \begin{bmatrix} \frac{\partial u_1}{\partial t_1} & \dots & \frac{\partial u_n}{\partial t_1} \\ \vdots & & \vdots \\ \frac{\partial u_1}{\partial t_n} & \dots & \frac{\partial u_n}{\partial t_n} \end{bmatrix} \nabla_u \phi. \quad (17)$$

The partial derivatives, using (15), are

$$\frac{\partial u_i}{\partial t_j} = \begin{cases} u_i^{p_2+1} \sum_{j \in \text{FI}(i)} (t_i - t_j - d_i^{\min})^{-p_2-1} & i = j \\ -u_i^{p_2+1} (t_i - t_j - d_i^{\min})^{-p_2-1} & j \in \text{FI}(i) \\ 0 & \text{otherwise.} \end{cases}$$

Since we can write  $u$  as an explicit function of  $x$ , see (16), we can compute  $\partial u_i / \partial x_j$ . Applying the chain rule to  $u$  as a function of  $x$  gives

$$\nabla_x \phi = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \dots & \frac{\partial u_n}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial u_1}{\partial x_n} & \dots & \frac{\partial u_n}{\partial x_n} \end{bmatrix} \nabla_u \phi, \quad (18)$$

where  $\nabla_x \phi = a$ . The partial derivatives, using (16), are

$$\frac{\partial u_i}{\partial x_j} = \begin{cases} -u_i / (x_i (1 - x_i^{-p_1})) & i = j \\ u_i F_{ij} / (g_i + \sum_{j \in \text{FO}(i)} F_{ij} x_j) & j \in \text{FO}(i) \\ 0 & \text{otherwise.} \end{cases}$$

To calculate  $\nabla \psi^s = \nabla_t \phi$  we start with  $\nabla_x \phi = a$ . Then we solve (18) to obtain  $\nabla_u \phi$ . The Jacobian matrix in (18) is lower triangular, and has  $n + m$  nonzero entries. Therefore  $\nabla_u \phi$  can be calculated efficiently. Finally,  $\nabla_t \phi$  is given by (17).

The gradient of the function  $\psi^s$  with respect to  $\tilde{t}$ ,  $g$ , is the vector consisting of the first  $\tilde{n}$  components of  $\nabla_t \phi$ .

### C. Search direction

For a large-scale problem solving a Newton like system (e.g., (19)) accurately is not computationally practical, and is not needed. We need to find a search direction  $\Delta\tilde{t}$  which is good enough in terms of the trade-off of the computational complexity versus the accelerated convergence it provides. In our method, the search direction  $\Delta\tilde{t}$  is an approximate solution of

$$\tilde{H}\Delta\tilde{t} = -g, \quad (19)$$

where the matrix  $\tilde{H} \in \mathbf{R}^{\tilde{n} \times \tilde{n}}$  is *not* the Hessian of the  $\psi^s$  but a matrix that captures the critical curvature of  $\psi^s$ .

Consider the function

$$\varphi(t) = \sum_{i=1}^n \sum_{j \in \text{FI}(i)} \frac{a_i}{t_i - t_j - d_i^{\min}}. \quad (20)$$

The domain of the function  $\varphi$ ,  $\text{dom } \varphi = \mathcal{Q}$  (given in (10)). The matrix  $\tilde{H}$ , instead of being the Hessian of  $\psi^s$ , is the Hessian of the function  $\varphi$  (with respect to  $\tilde{t}$ ). In (20), if  $\text{FI}(i) = \emptyset$  the term  $\sum_{j \in \text{FI}(i)} a_i / (t_i - t_j - d_i^{\min})$  should be interpreted as  $a_i / (t_i - d_i^{\min})$ .

The Hessian  $\tilde{H} \in \mathbf{R}^{\tilde{n} \times \tilde{n}}$  of the function  $\varphi$  (with respect to  $t$ ) can be written compactly in the following notation. Let the interconnect wires of the circuit be labeled  $1, \dots, m$ , where each wire connects the output of a particular gate to an input of a particular gate. We also label wires  $m+1, \dots, \tilde{m}$ , one for each primary input gate, where each of these wires connects to any one input of a different primary input gate. These wires can be thought as connections from the output of a pseudo-gate to an input of every primary input gate. Thus we have  $\tilde{m} - m$  primary input gates, and  $\tilde{m}$  wires. We define  $\beta \in \mathbf{R}^{\tilde{m}}$  as follows. For  $k = 1, \dots, m$ , let

$$\beta_k = \frac{2a_i}{(t_i - t_j - d_i^{\min})^3},$$

where wire  $k$  connects the output of gate  $j$  to an input of gate  $i$ ; and for  $k = m+1, \dots, \tilde{m}$ , let

$$\beta_k = \frac{2a_i}{(t_i - d_i^{\min})^3},$$

where wire  $k$  connects to an input of the (primary input) gate  $i$ . Let the matrix  $A \in \mathbf{R}^{n \times \tilde{m}}$  be

$$A_{ij} = \begin{cases} 1 & \text{wire } j \text{ connects to an input of gate } i \\ -1 & \text{wire } j \text{ connects to the output of gate } i \\ 0 & \text{otherwise.} \end{cases}$$

The Hessian of  $\varphi$  (with respect to  $t$ ) is

$$H = A \text{diag}(\beta_1, \dots, \beta_{\tilde{m}}) A^T.$$

The Hessian of  $\varphi$  with respect to  $\tilde{t}$ ,  $\tilde{H}$ , is the  $\tilde{n} \times \tilde{n}$  top left sub-block of the matrix  $H$ . The matrix  $\tilde{H}$  is given by

$$\tilde{H} = \tilde{A} \text{diag}(\beta_1, \dots, \beta_{\tilde{m}}) \tilde{A}^T,$$

where  $\tilde{A} \in \mathbf{R}^{\tilde{n} \times \tilde{m}}$  is the matrix formed by the first  $\tilde{n}$  rows of the matrix  $A$ .

The matrix  $H$  is the Laplacian matrix of the graph with weights  $\beta_i$  on the edges  $i = 1, \dots, \tilde{m}$ . (See [29, Section 2.5] for the Laplacian matrix of a graph.) Therefore the matrix

$H$  (and also  $\tilde{H}$ ) is diagonally dominant. Since the diagonal entries of  $H$  are positive and off-diagonal entries of  $H$  are non-positive, diagonal dominance for  $H$  means  $\sum_{j=1}^n H_{ij} \geq 0$ ,  $i = 1, \dots, n$ . We note this fact for future use.

To find a search direction  $\Delta\tilde{t}$  we approximately solve the system (19) using a diagonally preconditioned conjugate gradient (PCG) algorithm. We will not go into the details of the PCG algorithm, and refer the reader to [30], [31, Section 6.7], [27, Chapter 5]. It is well known that the PCG algorithm with diagonal preconditioning performs well when the matrix is diagonally dominant, which is indeed the case for the matrix  $\tilde{H}$ . (See, e.g., [32, Section 6.1] and [28, Section 8.5] for more details.) Within a small number of iterations of the PCG algorithm, we expect to get a good enough search direction  $\Delta\tilde{t}$ .

We now describe the important points needed in an implementation of the PCG algorithm, which are, the initialization rule, the truncation rule, and a couple of operations described below. First, we need to multiply a given vector  $w \in \mathbf{R}^{\tilde{n}}$  with the Hessian  $\tilde{H}$ , which is implemented as follows. To compute  $\tilde{H}w$ , we start by multiplying the vector  $w$  with the matrix  $\tilde{A}^T$ , then we multiply the vector  $\tilde{A}^T w$  with the matrix  $\text{diag}(\beta_1, \dots, \beta_{\tilde{m}})$ , and the result  $\text{diag}(\beta_1, \dots, \beta_{\tilde{m}}) \tilde{A}^T w$  with the matrix  $\tilde{A}$  to obtain  $\tilde{H}w = \tilde{A} \text{diag}(\beta_1, \dots, \beta_{\tilde{m}}) \tilde{A}^T w$ . Note that we never need to form the matrix  $\tilde{H}$ . Second, given  $b \in \mathbf{R}^{\tilde{n}}$  and the preconditioning matrix  $M$ , we need to solve the system of equations  $Mw = b$ , for  $w$ . Since our preconditioning matrix  $M$  is a diagonal matrix, solving the system  $Mw = b$  is trivial. The diagonal of  $M$  is the diagonal of  $\tilde{H}$ , which has positive entries and is given by  $\tilde{A}^{(2)}\beta$ , where the matrix  $\tilde{A}^{(2)} \in \mathbf{R}^{\tilde{n} \times \tilde{m}}$  is

$$\tilde{A}_{ij}^{(2)} = (\tilde{A}_{ij})^2, \quad i = 1, \dots, \tilde{n}, \quad j = 1, \dots, \tilde{m}.$$

Note that  $\beta$  depends on  $\tilde{t}$  but the matrix  $\tilde{A}^{(2)}$  does not. The matrix  $\tilde{A}^{(2)}$  depends on the circuit topology and therefore needs to be computed only once.

Now we address the issue of initialization and truncation of the PCG algorithm. A good initial search direction requires on average fewer iterations of the PCG algorithm, and therefore can accelerate the method. There are many choices for the initial search direction, e.g., 0, the search direction found in the previous step of the method. Truncation rule for the algorithm gives the number of PCG iterations to be carried out before terminating the algorithm. Among the various schemes we tried out for the choice of the initial search direction and the truncation rule the following worked quite well.

In our implementation, the PCG algorithm is initialized with 0 if the decrement in the objective value in the previous step is less than 0.05 times the decrement in the objective two steps earlier. This check indicates that the search direction found in the previous step is not good enough for the current step. Otherwise the search direction found in the previous step is used for initialization.

The truncation rule in our implementation is simple. We perform 2 PCG iterations when the PCG algorithm is initialized with the search direction found in the previous step of the method, and 4 PCG iterations when the PCG algorithm is initialized with 0. This means we carry out at most 4 PCG

iterations irrespective of the size of the matrix  $\tilde{H}$ . Each PCG iteration, in turn, involves a small number of passes over the circuit.

We should make a few comments about our choice of the function  $\varphi$  for the surrogate Hessian. The function  $\psi^s$  is a composition of functions, so applying the chain rule to get the Hessian will involve Tensor products, which is cumbersome; the complexity of computing the Hessian of  $\psi^s$  (let alone computing the search direction) will be at least  $O(n^2)$ , which will defeat our aim of having a scalable method. Second, the function  $\varphi$  can be obtained by substituting  $\sum_{j \in \text{FI}(i)} 1/(t_i - t_j - d_i^{\text{min}})$  for  $x_i$ ,  $i = 1, \dots, n$ , in the objective function  $\sum_{i=1}^n a_i x_i$ . This captures, approximately, the inverse relationship of the gate size  $x_i$  and the additional gate delay  $u_i$ , where  $u_i$  is given by (15). Also, the Hessian of the function  $\varphi$  can be easily adapted for applying the PCG algorithm as shown earlier in this section.

#### D. Line search

Given the search direction  $\Delta\tilde{t}$ , the new point is  $\tilde{t} + s\Delta\tilde{t}$ , where  $s \in \mathbf{R}_+$ , the step size, is to be computed. A backtracking line search is implemented to find the step size  $s$ . First, to ensure that the new point is feasible we compute the maximum step size  $s_0$  such that the point  $\tilde{t} + s_0\Delta\tilde{t}$  is feasible, *i.e.*,

$$t_i + s_0\Delta t_i - t_j - s_0\Delta t_j \geq d_i^{\text{min}}, \quad j \in \text{FI}(i), \quad i = 1, \dots, n.$$

In the backtracking line search, taking the initial value of  $s = 0.9s_0$ , the following condition is checked:

$$\psi^s(\tilde{t} + s\Delta\tilde{t}) < \psi^s(\tilde{t}) + \alpha_{\text{ls}} s g^T \Delta\tilde{t}, \quad (21)$$

with  $\alpha_{\text{ls}} = 0.01$ . If the condition (21) holds then  $s$  is taken to be the step size, else the step size is set to be  $s := \beta_{\text{ls}} s$ , with  $\beta_{\text{ls}} = 0.5$ , and the procedure is repeated. The current point is updated, *i.e.*,  $\tilde{t} := \tilde{t} + s\Delta\tilde{t}$ .

Note that an iteration of the backtracking line search is expensive. Each iteration of the line search, *i.e.*, checking the condition (21) requires one evaluation of the function  $\psi^s$ , which is one nonlinear back substitution.

#### E. Complexity

The computational complexity of the method, for a fixed number of iterations, is linear in the number of interconnections (or edges)  $m$ . To see this we analyze the complexity of each step of the method.

We analyze the step that needs to be carried out only one time, which is to find an initial feasible point. To compute the initial point we need to compute the lengths of a longest path and the critical slacks for all the gates. These quantities can be computed by two recursions which take a small number of operations per interconnection  $m$ .

Now we analyze the operations that are performed for every step of the method. First, we need to find the gradient  $g$ . The complexity of computing  $g$  or  $\nabla\psi^s$  is the number of nonzero entries in the Jacobian matrices in (18) and (17). Each of the matrices has  $n + m$  nonzero entries, so the complexity of computing the gradient is linear in  $m$ . Second,

TABLE I  
RC MODEL PARAMETER VALUES.

Inputs	Gate type	$a$	$r$	$c^{\text{in}}$	$c^{\text{int}}$
1	inv	3	0.333	3	3
2	nand2	8	0.333	4	6
2	nor2	10	0.333	5	6
3	aoi21	17	0.333	6	7
3	oai21	16	0.333	6	7

we need to computing the search direction, *i.e.*, an approximate solution to the system  $\tilde{H}\Delta\tilde{t} = g$  by the PCG algorithm. The computationally expensive operation for a PCG iteration is the matrix-vector multiplication, which primarily depends on the number of nonzeros in the matrix  $\tilde{A}$ . The number of nonzeros in  $\tilde{A}$  is linear in  $m$ . Since every step requires at most 4 PCG iterations, computing a search direction has a complexity linear in  $m$ . To compute the step size we require a small number of backtracking line search iterations. For each iteration we need to evaluate  $\psi(\tilde{t})$  which is one nonlinear back substitution. The complexity of nonlinear back substitution is linear in the number of edges because it requires a traversal of the circuit once from the primary outputs to the primary inputs, considering each edge exactly once. Thus the complexity of nonlinear back substitution, and therefore the complexity of computing the step size is linear in  $m$ .

Finally, suppose the stopping criteria does not depend on the circuit size ( $m$  or  $n$ ). Then the complexity of the method is linear in the number of interconnections  $m$  in the circuit. Suppose that the average fan-out of a gate is constant, and does not depend on  $n$  (which usually is the case). Then  $m$  is depends linearly on  $n$  and therefore the complexity of the method is linear in the number of gates of the circuit  $n$ .

The numerical experiments in Section VI-B suggest that a few hundred steps are sufficient to obtain a good enough solution, even for very large circuits. Thus it seems to be, for practical purposes, that the complexity of the method is linear in the number of gates in the circuit.

## VI. EXAMPLES

In this section we apply our method to 105 circuits. The examples consist of 35 different circuit topologies, and for each topology we solve the gate sizing optimization problem for 3 timing specifications: loose, medium, and tight. The 35 circuit topologies consist of 11 ISCAS-85 benchmark circuits [33], with numbers of gates ranging from 6 to 3512. The other 24 topologies are randomly generated, with number of gates ranging from 100 to 1000000.

We use the standard RC model for the gate delays as shown in (2), with model parameter values given in Table I. These model parameter values come from the logical effort model (see [1]). The values are chosen so that the delay of unit size inverter with no load is  $0.333 \times 3 \approx 1$ . Thus, the unit of time is taken to be the delay of an unloaded unit size inverter.

The randomly generated circuits include 5 types of gates, given in Table I. Depending on the number of inputs of a gate, the RC model parameter values for the gate are chosen from

Table I. If there are two or more models with the same number of inputs, we choose among them with equal probability. The value of wire/fixed load capacitance  $c^{\text{wire}}$  for a gate is chosen randomly from a uniform distribution on  $[0, 10]$ . For a gate whose output is an output of the circuit, a capacitance of 20 is added to the wire/fixed load capacitance. For the ISCAS-85 circuits, the RC model parameter values for gates are chosen similarly, according to the number of inputs of the gate. If the number of inputs of a gate  $\eta$  exceeds 3 then the parameters values are given by

$$a = 5\eta, \quad c^{\text{in}} = 2.3\eta, \quad c^{\text{int}} = 3\eta.$$

The resistance  $r$  is taken to be 0.333.

The timing specification  $T$  for a circuit will be given in terms of the minimum circuit delay  $T^{\text{min}}$ . For the randomly generated circuits, the loose timing specification is  $T^{\text{loose}} = 3.0T^{\text{min}}$ , the medium timing specification is  $T^{\text{medium}} = 2.7T^{\text{min}}$ , and the tight timing specification is  $T^{\text{tight}} = 2.4T^{\text{min}}$ . For the ISCAS-85 circuits,  $T^{\text{loose}} = 2.7T^{\text{min}}$ ,  $T^{\text{medium}} = 2.4T^{\text{min}}$ , and  $T^{\text{tight}} = 2.1T^{\text{min}}$ . These timing specifications are chosen so that the distribution of the optimal gate sizes is reasonable. For the loose timing specification, around half of the optimal gate sizes are of minimum size, and the largest gates have a size on the order of 16. For the medium timing specification around 35% of the optimal sizes were of minimum size, and the maximum gate size was 64. For the tight timing specification, around a quarter of the optimal gates have minimum size, and the largest gates have a size on the order of 128.

We will judge convergence by  $\epsilon(t)$ , the *suboptimality* of a timing assignment  $t$ , defined as

$$\epsilon(t) = (\psi(t) - f^*)/f^*, \quad (22)$$

where  $f^*$  is the optimal area. The domain of the function  $\epsilon$  is  $\text{dom } \epsilon = \mathcal{Q}$  (given in (10)). To observe the performance of our method we will look at the suboptimality  $\epsilon(t)$  versus the cumulative number of PCG iterations.

To calculate the suboptimality we need to know the exact optimal area  $f^*$ , which can be found by solving the problem by a standard technique. As mentioned earlier the gate sizing optimization problem (4) is a geometric program, and a standard technique is to transform the problem into a convex optimization problem and use an interior-point method to solve it. The advantage of the interior-point based standard method is that we can solve the problem to high degree of accuracy, which is guaranteed by the method since it produces a certificate of optimality for the solution. For all but the three largest circuits (with 100000 and more gates), we used a customized GP solver, using a primal-dual interior-point method to solve the convex optimization problem, with each Newton step solved approximately by the PCG algorithm. For the large circuits, computing the exact value of  $f^*$  was expensive (tens of hours), but we did this only to reliably judge convergence of our method (which was much faster). For the three largest circuits, even our custom GP solver failed. For these circuits, we estimated  $f^*$  using our own algorithm run many iterations (1000). For these circuits, then, we cannot be absolutely certify that our value of  $f^*$  is correct, but we have

very high confidence in our estimates. (This will become clear later.)

### A. Randomly generated circuit topologies

To generate a circuit topology we decide on a number of levels  $L$ , and the number of gates in each level  $N$ , so the total number of gates in the circuit is  $n = NL$ . Level 1 gates are primary input gates and level  $L$  gates are primary output gates. For each gate in the circuit the number of inputs of the gate is chosen independently according to the following probability distribution.

Inputs	1	2	3
Probability	0.2	0.4	0.4

Similarly, the tentative number of gates in the fan-out of a gate is chosen independent of its number of inputs and other gates, according to the following probability distribution.

Outputs	1	2	3	4	5
Probability	0.25	0.35	0.30	0.025	0.025
Outputs	6	7	8	9	10
Probability	0.01	0.01	0.01	0.01	0.01

The actual number of gates in the fan-out of a gate can be less than the randomly generated number due to nonavailability of gate inputs at a higher level. In such a case the gate's output serves as an output of the circuit.

The output of a gate will be connected to an input of a gate at a higher level, or will serve as an output of the circuit. Therefore the topology of the circuit will be a directed acyclic graph. For each gate, for each output, a level is chosen independently according to the following probability distribution.

Level	+1	+2	+3	Circuit output
Probability	0.75	0.1875	0.0469	0.0156

Once a level is chosen, then among all the unconnected inputs of all the gates in that level, one unconnected input is chosen randomly. This input, and the output of the gate (at a lower level) for which this input has been selected, are then connected. After connecting the outputs of all the gates, some gates may have some of their inputs unconnected. Such unconnected inputs are considered as the inputs of the circuit.

We generate 24 topologies, ranging from 100 gates (rand1), up to 1000000 gates (rand24). These topologies are listed in Table II.

### B. Numerical results

We start by showing the performance of the method for a typical circuit, rand21, with 100000 gates, and the medium timing specification. Fig. 1 shows the suboptimality  $\epsilon(t)$  versus the cumulative number of PCG iterations. The soft-max weight is  $p_1 = 5$ , and the soft-min weight is  $p_2 = 55$ . The dotted line shows the value of  $(\psi^s(t) - f^*)/f^*$ , and the solid line displays the suboptimality  $\epsilon(t)$ . Since  $\psi^s(t) > \psi(t)$  for any  $t$ , the solid line is below the dotted line. The two circles indicate the first iteration to achieve 10% and 5% suboptimality, *i.e.*, 50 and 72 iterations, respectively. To give a rough idea of the speed of our

algorithm, we note that the custom GP solver required around 20 hours to compute  $f^*$ , whereas our algorithm required around 4 minutes to compute a 5% suboptimal point. The time taken by MOSEK [14] (a standard GP solver) to compute  $f^*$  for a much smaller circuit, rand8 with 1200 gates, is around 20 minutes.

We solve the gate sizing optimization problem (4) for the randomly generated circuits with the loose, medium and tight timing specifications. We use soft-max weight  $p_1 = 5$ , and soft-min weight  $p_2 = 55$ . The cumulative number of PCG iterations required to achieve 10% and 5% suboptimality is shown in Table II. The corresponding numbers for the ISCAS-85 circuits are shown in Table III. For all the circuits, the plot of the suboptimality versus cumulative number of PCG iterations looks similar to Fig. 1. In general, by 500 cumulative number of PCG iterations (or less) the method has converged to the optimal solution, *i.e.*, the suboptimality is less than 1%.

Increasing the number of PCG iterations in each step, to compute an approximate search direction, does not help. Consider, for example, the rand21 circuit with medium timing specification. If 10 PCG iterations are used in each step, the cumulative PCG iterations required to achieve 10% and 5% suboptimality are 120 and 220, respectively, as compared to 50 and 72. Similar results are observed for the other circuits, which indicates that increasing the number of PCG iterations for each step, is, in fact, wasteful.

The values for weights of the soft-max and soft-min functions,  $p_1$  and  $p_2$  respectively, are chosen so the approximation  $\psi^s$  is good enough, as judged by the final performance. For very small values of  $p_1$  and  $p_2$  the approximation poor is enough to affect final performance; for very large values of  $p_1$  and  $p_2$  numerical problems are sometimes encountered. However, the method works for a wide range of values of  $p_1$  and  $p_2$ .

For various circuits, the average number of line search iterations to find the step size in step 2c. of the algorithm, is 3 or sometimes even less. Each line search iteration, *i.e.*, checking condition (21), requires 1 nonlinear back substitution, which is one pass over the circuit. The computational effort of the line search amortized over the PCG iterations is small. Thus measuring the computational effort of the method in terms of the cumulative PCG iterations is a good criterion.

As mentioned earlier, it is not practical to solve the gate sizing optimization problem for circuits rand22, rand23, and rand24 (which contain more than 100000 gates), using the customized GP solver. For these large-scale circuits the optimal area  $f^*$  is taken to be the area obtained by our method after running for 1000 cumulative PCG iterations. This is indicated by a break in Table II. For these three large circuits, the loose, medium and tight timing specification were changed to  $T^{\text{loose}} = 3.2T^{\text{min}}$ ,  $T^{\text{medium}} = 2.9T^{\text{min}}$ , and  $T^{\text{tight}} = 2.6T^{\text{min}}$ , respectively, so as to achieve reasonable distributions of optimal gate sizes. (For the same reason, we use  $T^{\text{loose}} = 3.0T^{\text{min}}$ ,  $T^{\text{medium}} = 2.7T^{\text{min}}$ , and  $T^{\text{tight}} = 2.4T^{\text{min}}$ , for c6288 ISCAS-85 circuit.)

The values in Table II and Table III indicate that the cumulative number of PCG iterations required to achieve points with suboptimality less than 10% or 5% does not depend on

TABLE III  
PCG ITERATIONS REQUIRED TO ACHIEVE 10% AND 5% SUBOPTIMALITY  
FOR ISCAS-85 BENCHMARK CIRCUITS.

Circuit	Total gates	Inter-connections	PCG iterations					
			Loose		Medium		Tight	
			10%	5%	10%	5%	10%	5%
c17	6	6	0	4	0	4	0	4
c432	160	255	10	22	4	24	22	30
c499	202	296	16	38	18	36	22	32
c880	383	507	8	30	10	34	30	40
c1355	546	856	30	56	40	48	60	108
c1908	880	1419	62	76	98	152	114	148
c2670	1193	1850	12	22	26	42	166	274
c3540	1669	2630	32	56	54	72	90	116
c5315	2307	3878	4	16	18	24	42	56
c6288	2416	4288	6	42	42	124	170	256
c7552	3512	5836	6	20	22	50	62	102

the problem size, *i.e.*, the number of gates  $n$ . (The ratio of number of interconnections to the number of gates is around 2 for the randomly generated topologies, and around 1.5 for the ISCAS-85 circuits. Therefore we will use number of gates as our comparison criterion.) In fact, a couple of hundred of cumulative PCG iterations are good enough to achieve a point with suboptimality less than 10%. The cumulative number of PCG iterations do depend on the timing specification. One reason is that the suboptimality of the initial point found by the method described in Section V-A, increases as the timing specification  $T$  decreases. The suboptimality of the initial point for the circuits with loose timing specification is around 20%; with medium timing specification the suboptimality is around 50%; and with tight timing specification, the suboptimality is typically around 150%. In some cases the initial points are within 10% of optimality, before any PCG steps have been carried out. (This is seen as the entries marked 0 in Table III.) This is the case for the ISCAS-85 c17 circuit, for example, even with the tight timing specification.

Of course, in practice we do not know the value of  $f^*$ . The numerical results shown (and many others that are not shown) suggest that a very safe stopping criterion is to simply run the algorithm for some fixed total PCG iterations, such as 200 or 300. We plot the number of circuits versus the suboptimality obtained after 200 and 300 iterations, in Fig. 2. We see that at 300 PCG iterations the suboptimality is less than 10% for all circuits and less than 5% for most of the circuits.

Finally, we report the time required to obtain a point with less than 10% suboptimality for various circuits. A scatter plot of time taken versus the number of gates in the circuit is shown in Fig. 3 on a log-log scale. The dashed line is the least-squares fit of the function  $\mathcal{C}(n) = o \log n + \kappa$  to the log of the observed times. The parameters  $o$  and  $\kappa$  are found to be 1.11 and  $-9.36$ , respectively. This shows that the time required to achieve 10% suboptimality is nearly linear in the number of gates (or interconnect wires).

Our implementation, `lsgs`, written in MATLAB and C (using the MEX interface), is available online [34]. The method takes approximately 20 seconds for rand14 (9000

TABLE II  
PCG ITERATIONS REQUIRED TO ACHIEVE 10% AND 5% SUBOPTIMALITY FOR RANDOMLY GENERATED CIRCUITS.

Circuit	Levels	Gates	Total gates	Interconnections	PCG iterations					
					Loose		Medium		Tight	
					10%	5%	10%	5%	10%	5%
rand1	10	10	100	165	6	12	8	12	8	20
rand2	10	15	150	272	6	10	10	18	14	26
rand3	10	20	200	381	8	10	8	16	14	30
rand4	10	30	300	575	6	12	10	16	16	20
rand5	20	20	400	800	8	12	18	30	36	62
rand6	20	30	600	1183	12	20	16	36	48	86
rand7	20	40	800	1626	10	14	14	32	48	96
rand8	20	60	1200	2462	8	14	14	28	40	76
rand9	20	80	1600	3316	24	42	50	64	68	162
rand10	20	120	2400	4911	14	26	42	56	60	124
rand11	20	160	3200	6548	32	44	18	36	52	100
rand12	20	225	4500	9269	10	34	18	36	48	88
rand13	20	315	6300	12962	36	64	38	60	90	122
rand14	20	450	9000	18594	10	26	56	74	50	92
rand15	20	630	12600	25994	18	44	38	52	78	122
rand16	20	900	18000	37183	10	42	42	62	74	138
rand17	20	1250	25000	51764	18	50	48	64	96	142
rand18	20	1775	35500	73500	8	42	32	58	90	140
rand19	20	2500	50000	103423	10	24	86	106	146	216
rand20	20	3500	70000	144890	10	44	50	68	160	220
rand21	20	5000	100000	206673	10	24	50	72	152	198
rand22	40	5000	200000	426671	6	34	84	108	156	186
rand23	40	12500	500000	1066850	10	24	76	126	272	340
rand24	40	25000	1000000	2135860	12	52	88	138	232	322

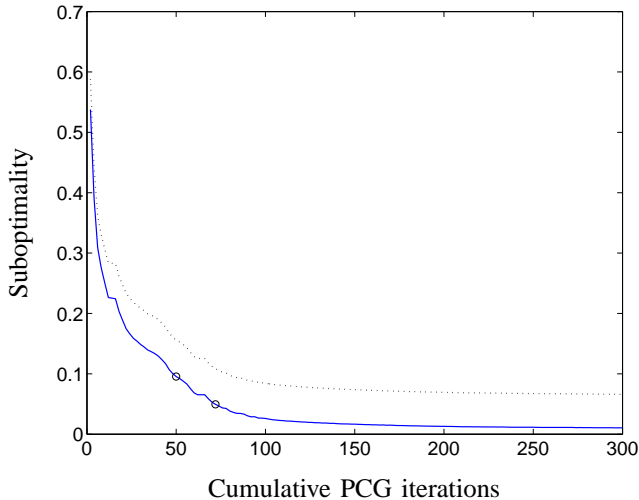


Fig. 1. Suboptimality versus cumulative PCG iterations for rand21 circuit (consisting of 100000 gates), with medium timing specification.

gate) circuit, 4 minutes for rand21 (100000 gate) circuit, and 40 minutes for rand24 (1000000 gate) circuit, to achieve a point with less than 10% suboptimality. Since the method is implemented in MATLAB, the measured times should not be taken too seriously; a complete C implementation is expected to be substantially faster.

Of course, these experiments do not *prove* that our method

will converge to a point with 5% suboptimality or less within, say, 300 PCG iterations. However, these results (and many others, not reported here) strongly suggest that this is the case.

## VII. GLOBAL OPTIMALITY

In this section we will show that the functions  $\psi$  and  $\psi^s$  are convex in  $t$ .

Before proceeding, we would like to mention an important difference between the traditional approach to solve the gate sizing optimization problem as a *geometric program*, and our approach. One standard way to solve the problem (4) is based on recognizing that the problem is a geometric program in the variables  $x_i, d_i, t_i, i = 1, \dots, n$ . This means that the problem (4) is a convex optimization problem in the variables  $\log x_i, \log d_i, \log t_i, i = 1, \dots, n$ . Our approach is different. We show that the gate sizing optimization problem (4) can be formulated as a convex optimization problem in the variables  $\log x_i, \log u_i, t_i, i = 1, \dots, n$ . This will lead us to show that function  $\psi$  is convex in  $t$ . The function  $\psi$  is not differentiable, and we construct a smooth approximation  $\psi^s$ , by choosing the soft-min and soft-max functions so that  $\psi^s$  is also convex in  $t$ .

### A. Convexity of $\psi$

Consider the transformation

$$y_i = \log x_i, \quad z_i = \log u_i, \quad i = 1, \dots, n. \quad (23)$$



tions explicitly, the optimization problem obtained is

$$\begin{aligned}
& \text{minimize} && \sum_{i=1}^n a_i \exp(y_i) \\
& \text{subject to} && \exp(p_1 y_i) \geq (g_i \exp(-z_i) \\
& && \quad + \sum_{j \in \text{FO}(i)} F_{ij} \exp(y_j - z_i))^{p_1} + 1, \\
& && \quad i = 1, \dots, n \\
& && \exp(-p_2 z_i) \geq \sum_{j \in \text{FI}(i)} (t_i - t_j - d_i^{\min})^{-p_2}, \\
& && \quad i = 1, \dots, n \\
& && t_j = T, \quad j \in \text{PO},
\end{aligned} \tag{26}$$

with the variables are  $y \in \mathbf{R}^n$ ,  $z \in \mathbf{R}^n$ , and  $t \in \mathbf{R}^n$ . The domain of the problem is  $\mathcal{S}$  (given in (25)). Rearranging the terms and taking the log of the first set of inequalities, the optimization problem (26) can be written as

$$\begin{aligned}
& \text{minimize} && \sum_{i=1}^n a_i \exp(y_i) \\
& \text{subject to} && \log(\exp(p_1 y_i) - 1) \geq p_1 \log(g_i \exp(-z_i) \\
& && \quad + \sum_{j \in \text{FO}(i)} F_{ij} \exp(y_j - z_i)), \\
& && \quad i = 1, \dots, n \\
& && 1 \geq \exp(p_2 z_i) \sum_{j \in \text{FI}(i)} (t_i - t_j - d_i^{\min})^{-p_2}, \\
& && \quad i = 1, \dots, n \\
& && t_j = T, \quad j \in \text{PO}.
\end{aligned} \tag{27}$$

We now show that the problem (27) is a convex optimization problem in the variables  $y$ ,  $z$ , and  $t$ . The objective function is convex in  $y$ , as shown in Section VII-A. The first set of inequalities can be written as

$$\begin{aligned}
\log(\exp(p_1 y_i) - 1) &\geq p_1 \log\left(\exp(-z_i + \log g_i) \right. \\
&\quad \left. + \sum_{j \in \text{FO}(i)} \exp(y_j - z_i + \log F_{ij})\right), \\
&\quad i = 1, \dots, n,
\end{aligned}$$

since  $g_i > 0$ ,  $F_{ij} > 0$ ,  $j \in \text{FI}(i)$ ,  $i = 1, \dots, n$ . The function on the right side is the log-sum-exp function, and the function  $\log(\exp(p_1 y_i) - 1)$  is concave in  $y_i$  (see (28) in Appendix ). Thus the first set of inequalities represents a convex set. The second set of inequalities is

$$1 \geq \exp(p_2 z_i) \sum_{j \in \text{FI}(i)} (v_{ij})^{-p_2}, \quad i = 1, \dots, n,$$

where  $v_{ij} = t_i - t_j - d_i^{\min}$ ,  $j \in \text{FI}(i)$ ,  $i = 1, \dots, n$ . Using (30) in Appendix , and since  $v_{ij}$  is an affine transformation of  $(t_i, t_j)$ , the function on the right hand side is a convex function of  $z_i$ ,  $t_i$ , and  $t_j$ ,  $j \in \text{FI}(i)$ . Thus the second set of inequalities also represents a convex set. Therefore the problem (27) is convex in  $y$ ,  $z$ , and  $t$ .

The optimal objective value of the approximation (13) as a function  $t$ , *i.e.*,  $\psi^s(t)$ , is the optimal objective value of the problem (13) or (27) for the given (fixed)  $t$ . Thus eliminating  $y$  and  $z$  in the problem (27), which effectively is eliminating  $x$  and  $u$  in the problem (13), gives the optimization problem of minimizing  $\psi^s(t)$  subject to  $t_j = T$ ,  $j \in \text{PO}$ . Therefore  $\psi^s$  is convex in  $t$ .

### VIII. CONCLUSION

In this paper we have described a new custom method for solving the gate sizing problem with an RC timing model. Numerical experiments show that the method reliably finds

solutions accurate to 5% (or better) with a computational effort that scales linearly with problem size, up to 1000000 gates. Our method can size a 1000000-gate circuit in around 40 minutes. For a circuit of this size, the associated geometric program has 3000000 variables, and more than 6000000 monomial terms in the constraints. As far as we know, these are among the largest GPs ever solved.

The same approach can be generalized to handle a variety of extensions. For example, the nonlinear back substitution method can be used with any delay model in which the gate delay is monotone decreasing in the gate size, for a given load capacitance. More complex timing models, such as distinguishing between rising and falling gate delays and signal arrival times, can also be used.

Finally, we mention a variation on the problem formulation. In this paper we focus on the problem of minimizing area (or power), given a timing constraint. (As part of our solution, we determine, very efficiently, whether the timing constraint is feasible.) Our method can be used to trace out the entire optimal trade-off curve between area and circuit delay (*i.e.*,  $T$ ). This is done simply by minimizing the area, for each of several values of  $T$ . Once we have this curve, we can readily solve the problem of minimizing circuit delay, subject to a limit on area.

### APPENDIX SOME CONVEX FUNCTIONS

In this section we show the convexity of some of the functions used in Section VII-B.

Consider the function

$$h(y) = -\log(\exp(py) - 1), \tag{28}$$

where  $y \in \mathbf{R}$ , and  $p$  is a positive constant. The domain of  $h$  is  $\text{dom } h = \{y \mid y > 0\}$ . The function  $h$  is convex in  $y$ . This can be seen by calculating the second derivative of  $h$

$$\frac{d^2 h}{dy^2} = \frac{p^2 \exp(py)}{(\exp(py) - 1)^2},$$

which is positive for any  $y$ .

Consider the function

$$g(w, v) = \exp(wp)v^{-p}, \tag{29}$$

where  $w \in \mathbf{R}$ ,  $v \in \mathbf{R}$ , and  $p$  is a positive constant. The domain of  $g$  is  $\text{dom } g = \{(w, v) \mid w \geq 0, v > 0\}$ . The function  $g$  is convex in  $(w, v)$ . To prove this we will show that the Hessian of  $g$  is positive definite. The Hessian of the function  $g$  is

$$\nabla^2 g = p \exp(wp) \begin{bmatrix} pv^{-p} & -pv^{-p-1} \\ -pv^{-p-1} & (p+1)v^{-p-2} \end{bmatrix}.$$

The diagonal elements of the  $2 \times 2$  Hessian matrix are positive. The determinant of the Hessian is

$$\det(\nabla^2 g) = p^2 \exp(wp)v^{-2p-2} > 0.$$

Therefore the function  $g$  is convex.

Consider the function

$$f(w, v) = \exp(wp) \sum_{i=1}^n v_i^{-p}, \tag{30}$$

where  $w \in \mathbf{R}$ ,  $v \in \mathbf{R}^n$ , and  $p$  is a positive constant. The domain of  $f$  is

$$\text{dom } f = \{(w, v) \mid w \geq 0, v_i > 0, i = 1, \dots, n\}.$$

The function  $f$  is convex in  $(w, v)$ , as it is the sum of convex functions, each function of the form  $g$  given in (29).

#### ACKNOWLEDGMENTS

We thank the anonymous reviewers for constructive comments and helpful suggestions. We thank Kwangmoo Koh and Seung-Jean Kim for help with the customized large-scale GP solver that we used to calculate the exact optimal solutions for our test circuits. We thank Phiroze Parakh, Benigne Deprey, and Ismail Bustany of Sierra Design Automation for many helpful suggestions. We are grateful to Mark Horowitz and Dinesh Patil for helpful suggestions, especially in formulating the random circuit topology generator used in the examples. We are grateful to Jo Ebergen of Sun Microsystems, and Kyle Kelley for extremely useful feedback on the `lsgs` toolbox.

This work was funded in part by a Stanford Graduate Fellowship, the MARCO Focus Center for Circuit & System Solutions (C2S2, [www.c2s2.org](http://www.c2s2.org)), under contract 2003-CT-888, by AFOSR grant AF F49620-01-1-0365, by NSF grant ECS-0423905, by NSF grant 0529426, by DARPA/MIT grant 5710001848, by AFOSR grant FA9550-06-1-0514, DARPA/Lockheed contract N66001-06-C-2021, and by AFOSR/Vanderbilt grant FA9550-06-1-0312.

#### REFERENCES

- [1] I. Sutherland, B. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*. San Francisco, CA: Morgan Kaufmann Publishers, 1999.
- [2] P. Rezvani and M. Pedram, "A fanout optimization algorithm based on the effort delay model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 12, pp. 1671–1678, 2003.
- [3] S. Boyd, S.-J. Kim, D. Patil, and M. Horowitz, "Digital circuit optimization via geometric programming," *Operation Research*, vol. 53, no. 6, pp. 899–932, 2005.
- [4] W. Chuang, S. Sapatnekar, and I. Hajj, "Delay and area optimization for discrete gate sizes under double-sided timing constraints," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1993, pp. 9.4.1–9.4.4.
- [5] H. Sathiyamurthy, S. Sapatnekar, and J. Fishburn, "Speeding up pipelined circuits through a combination of gate sizing and clock skew optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 2, pp. 173–182, 1998.
- [6] I. Jiang, Y. Chang, and J. Jou, "Crosstalk-driven interconnect optimization by simultaneous gate and wire sizing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 9, pp. 999–1010, 2000.
- [7] M. Ketkar and S. Sapatnekar, "Standby power optimization via transistor sizing and dual threshold voltage assignment," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Nov. 2002, pp. 375–378.
- [8] S.-J. Kim, S. Boyd, S. Yun, D. Patil, and M. Horowitz, "A heuristic for optimizing stochastic activity networks with applications to statistical digital circuit sizing," *Optimization and Engineering*, vol. 8, no. 4, pp. 397–430, 2007.
- [9] P. Pant, M. Roy, and A. Chatterjee, "Dual-threshold voltage assignment with transistor sizing for low power CMOS circuits," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 9, no. 2, pp. 390–394, 2001.
- [10] C.-P. Chen, C. Chu, and D. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 7, pp. 1014–1025, 1999.
- [11] O. Coudert, R. Haddad, and S. Manne, "New algorithms for gate sizing: A comparative study," in *Proceedings of 33rd IEEE/ACM Design Automation Conference*, 1996, pp. 734–739.
- [12] O. Coudert, "Gate sizing for constrained delay/power/area optimization," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 5, no. 4, pp. 465–472, 1997.
- [13] J. Ebergen, J. Gainsley, and P. Cunningham, "Transistor sizing: How to control the speed and energy consumption of a circuit," in *Proceedings of the 10th International Symposium on Asynchronous Circuits and Systems*, 2004, pp. 51–61.
- [14] E. Andersen and K. Andersen, *The MOSEK Optimization Tools Version 3.2. User's Manual and Reference*, 2002, available from [www.mosek.com](http://www.mosek.com).
- [15] A. Mutapcic, K. Koh, S. Kim, L. Vandenbergh, and S. Boyd, *GGPLAB: A Simple Matlab Toolbox for Geometric Programming*, 2006, available from [www.stanford.edu/~boyd/ggplab/](http://www.stanford.edu/~boyd/ggplab/).
- [16] J. Dahl and L. Vandenbergh, *CVXOPT: A Python Package for Convex Optimization*, available from [www.ee.ucla.edu/~vandenbe/cvxopt/](http://www.ee.ucla.edu/~vandenbe/cvxopt/).
- [17] J. Fishburn and A. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," in *IEEE International Conference on Computer-Aided Design: ICCAD-85. Digest of Technical Papers*. IEEE Computer Society Press, 1985, pp. 326–328.
- [18] K. Kasamsetty, M. Ketkar, and S. Sapatnekar, "A new class of convex functions for delay modeling and its application to the transistor sizing problem," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 7, pp. 779–788, July 2000.
- [19] S. Sapatnekar, V. Rao, P. Vaidya, and S. Kang, "An exact solution to the transistor sizing problem for CMOS circuits using convex optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 11, pp. 1621–1634, Nov. 1993.
- [20] S. Sapatnekar and S. Kang, *Design Automation for Timing-driven Layout Synthesis*. Kulwer Academic Publishers, 1993.
- [21] K. Koh, S.-J. Kim, and S. Boyd, "An interior-point method for large-scale  $\ell_1$ -regularized logistic regression," *The Journal of Machine Learning Research*, vol. 8, pp. 1519–1555, 2007.
- [22] F. Beeffink, P. Kudva, D. Kung, and L. Stok, "Gate-size selection for standard cell libraries," in *Digest of Technical Papers the IEEE/ACM International Conference on Computer-Aided Design*, Nov. 1998, pp. 545–550.
- [23] J. Cong and K.-S. Leung, "Optimal wiresizing under Elmore delay model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 3, pp. 321–336, 1995.
- [24] J. Cong and H. He, "Theory and algorithm of local-refinement-based optimization with application to device and interconnect sizing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 4, pp. 406–420, 1999.
- [25] S. Boyd and L. Vandenbergh, *Convex Optimization*. Cambridge University Press, 2004.
- [26] S. Boyd, S.-J. Kim, L. Vandenbergh, and A. Hassibi, "A tutorial on geometric programming," *Optimization and Engineering*, vol. 8, no. 1, pp. 67–127, 2007.
- [27] J. Nocedal and S. Wright, *Numerical Optimization*, ser. Springer Series in Operations Research. New York: Springer, 1999.
- [28] D. Luenberger, *Linear and Nonlinear Programming*, 2nd ed. Addison-Wesley, 1984.
- [29] R. Brualdi and H. Ryser, *Combinatorial Matrix Theory*. Cambridge University Press, 1991.
- [30] C. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, ser. Frontiers in Applied Mathematics. Philadelphia: SIAM, 1995, vol. 16.
- [31] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia: SIAM, 2003.
- [32] R. A. Horn and C. A. Johnson, *Matrix Analysis*. Cambridge University Press, 1985.
- [33] *ISCAS-85 benchmark circuits*, available from [www.fm.vslib.cz/~kes/asic/iscas/](http://www.fm.vslib.cz/~kes/asic/iscas/).
- [34] R. A. Joshi and S. Boyd, *lsgs: large-scale gate sizing MATLAB toolbox, version 0.25*, Mar. 2007, available from [www.stanford.edu/~boyd/lsgs/](http://www.stanford.edu/~boyd/lsgs/).



**Siddharth Joshi** received the B. Tech. (with Honors) degree in electrical engineering from the Indian Institute of Technology, Kharagpur, India, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, in 2002 and 2004, respectively. He is currently pursuing the Ph.D. degree in electrical engineering at Stanford University.

His current interests include application of convex optimization to various engineering applications.

Siddharth is a recipient of the Stanford Graduate Fellowship.



**Stephen Boyd** (S'82–M'85–SM'97–F'99) received the A.B. degree in mathematics from Harvard University, Cambridge, MA, in 1980, and the Ph.D. in electrical engineering and computer science from the University of California, Berkeley, in 1985.

He is the Samsung Professor of Engineering and Professor of Electrical Engineering in the Information Systems Laboratory at Stanford University, Stanford University. His current research focus is on convex optimization applications in control, signal processing, and circuit design.