Gate Sizing Using Lagrangian Relaxation Combined with a Fast Gradient-Based Pre-Processing Step

Hiran Tennakoon and Carl Sechen
Department of Electrical Engineering
University of Washington
Seattle, WA 98195-2500
hiran@ee.washington.edu, sechen@ee.washington.edu

Abstract — *In this paper, we present Forge, an optimal algorithm for gate sizing using the Elmore delay model. The algorithm utilizes Lagrangian relaxation with a fast gradient-based pre-processing step that provides an effective set of initial Lagrange multipliers. Compared to the previous Lagrangian-based approach, Forge is considerably faster and does not have the inefficiencies due to difficult-to-determine initial conditions and constant factors. We compared the two algorithms on 30 benchmark designs, on a Sun UltraSparc-60 workstation. On average Forge is 200 times faster than the previously published algorithm. We then improved Forge by incorporating a slew-rate-based convex delay model, which handles distinct rise and fall gate delays. We show that Forge is 15 times faster, on average, than the AMPS transistor-sizing tool from Synopsys, while achieving the same delay targets and using similar total transistor area.*

1 Introduction

The increasing use of fluid standard cell libraries demands efficient algorithms for gate sizing. For some time researchers have been exploiting the convexity of the Elmore [1] or Penfield-Rubinstein [2] delay model for optimizing transistor, gate and wire sizes. One of the first algorithms was TILOS [4], which models transistors as RC equivalent circuits and uses the Elmore model for gate delays. TILOS makes incremental changes to the transistors using a delay sensitivity criterion.

Sapatnekar, *et al*., presented the first exact solution to the transistor-sizing problem, minimizing a convex function over a convex set of constraints [3]. The formulation of the problem minimized circuit area subject to the constraint that the maximum delay for all paths meets a delay target. An interior point method was used to find the global minimum area. The algorithm proceeds by binding the optimal solution in a multi-dimensional volume. The size of this volume is progressively halved (approximately) while ascertaining that the half that is retained binds the optimal solution. This is accomplished by finding a point near the center of the current volume. This is an iterative process with time complexity per iteration of $O(n^{2.5})$, where n is the number of transistors in the design. Once a point close to the center is found, a hyper-plane is generated to halve the volume. When the volume is sufficiently small the algorithm terminates. This method may not be suitable for large problems, as the cost per iteration is high.

In 1999, Chen, *et al*. [5], presented an elegant solution by defining constraints on the circuit components rather than on the signal paths. Instead of an exponential number of signal paths, now only a linear number of timing constraints over all circuit components needs to be considered. Thus, a single iteration of this algorithm has a time complexity of $O(n)$, where n is the number of components. This approach was also used in [6]. Lagrangian relaxation then solves the problem of minimizing total area subject to a delay requirement [7]. This work was the first to prove convergence of the algorithm and to guarantee optimality. However, certain practical implementation details expose the weaknesses of this approach and these will be discussed in the subsequent sections. In Lagrangian relaxation, constraints are "relaxed" by incorporating them into the objective function. In our case the objective function is the total area of the circuit, or a weighted sum of the component sizes. Multiplying each constraint by a constant called the *Lagrange multiplier*, and adding it to the objective function facilitates this inclusion. Thus, for a fixed vector **e** of Lagrange multipliers, we have a new optimization problem that is free of constraints. Of course, the challenge now is to find suitable values for **e**. In [5], the approach is to start from an arbitrary point in the solution space. Then the Lagrange multipliers are iteratively adjusted based on the difference between the timing of the current solution and the desired timing. We propose an approach that estimates the optimal vector **e** using fast gradient-based minimization. Then the relaxation process is invoked with this vector **e**, guaranteeing minimum area. We call our implementation *Forge*, since our algorithmic approach "forges" fast delay minimization through a gradient-based search and optimal area minimization through the relaxation method.

The rest of the paper is organized as follows. Section 2 will briefly go over the Elmore delay model. Section 3 will give details of Lagrangian relaxation applied to minimizing area given a maximum delay bound. We are using the Elmore model, as it is simple enough for compact equations leading to clearer derivations. Motivation for the *Forge* algorithm will be given in Section 4. We will consider practical implementation issues of Lagrangian relaxation, as proposed in [5], in this section. *Forge* is discussed in Section 5. Section 6 will show results comparing *Forge* with the stand-alone Lagrangian method (*e.g.* in [5]). Results with *Forge* using a more complex delay model and comparisons with the AMPS tool from Synopsys are in Section 7. Conclusions will be given in Section 8.

2 Elmore Delay Model

The Elmore delay model is applicable to distributed networks of resistors and capacitors [1][2]. Thus, each component in the design is decomposed into an equivalent output resistance, and each input to a component is represented by a capacitance. Let $\hat{r}_i$ and $\hat{c}_i$, be the unit size output resistance and input capacitance for component i, respectively. The size of component i is given by x_i. Then the output resistance and input capacitance for a component i are: $r_i = \hat{r}_i / x_i$ and $c_i = \hat{c}_i \cdot x_i$. The delay for the design is then the sum of the resistor delays. The delay for a resistor is its resistance multiplied by the downstream capacitance.

Lagrangian relaxation has been successfully applied to simultaneous optimization of interconnect and gate sizes considering cross-talk [8] with the Elmore delay model. Next we look at area minimization given a maximum delay constraint with Lagrangian relaxation based on the Elmore model.

3 Lagrangian Relaxation

We review the algorithm presented in [5]. For simplicity we will only consider gate sizing using the Elmore delay model. It has been shown that the algorithm applies to simultaneous gate and wire sizing with a delay model that accounts for input/output slew rates and diffusion capacitances [5]. The main optimization problem, termed the Primal Problem (PP), is first reduced to a simpler form, using characteristics of the optimal solution. Then by considering the Lagrangian Dual Problem (LDP) the optimal solution to PP is obtained. LDP can be seen as an optimization problem having variables corresponding to the constraints of the PP [7][10]. We begin by looking at the Primal Problem.

3.1 The Primal Problem (PP)

Consider a design with s input drivers, t output loads and n resizable components. An output resistance R_i^D represents each primary input $i, 1 \le i \le s$, and a load capacitance C_j^L is at each primary output j, $1 \le j \le t$. Input drivers and gates are termed components. A node is the output of a component that connects components together or connects primary output drivers to an output load. Thus, there are $n+s$ components and nodes. Two additional artificial components are added, an output and input component. The output component connects all the t output loads, and the input component is connected to all the s inputs. A circuit with the artificial components included is shown in Figure 1. There are now $n+s+2$ components and nodes. Let $m = n+s+1$. The components and nodes are labeled in reverse topological order.

Topological sorting is achieved in linear run-time via PERT [9]. Thus, we have the output loads in the range $1 \le i \le t$, the resizable components are in the range $1 \le i \le n$, and the input drivers are in the range

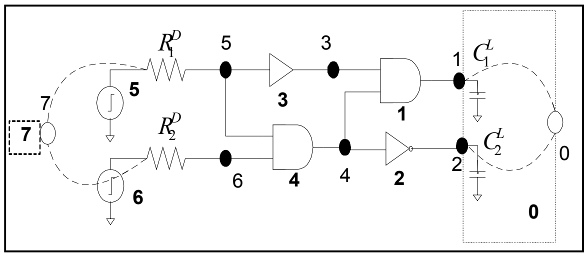

Fig. 1. Circuit representation showing two artificial components. Nodes and components are indexed. The filled circles represent nodes.

$n+1 \le i \le n+s$. Let $input(i)$ be the set of indices of components connected to the inputs of component i for $0 \le i \le m-1$. Let $output(i)$ be the set of indices of components connected to the output of component i. Then the Primal Problem (PP) is formulated as follows:

x_i represents the size of the component, a_i and D_i are the arrival time at the output, and the delay of component i, re-

$$
\boxed{
\begin{array}{lll}
PP: \text{Minimize} \sum_{i=1}^{n} \alpha_i x_i & & \\
\quad \text{subject to } a_j \le A_0 & \text{j}=1,..,t: \text{outputs} & \\
\quad\quad a_j + D_i \le a_i & i = 1,...,n \text{ and } \forall j \in input(i) & \\
\quad\quad D_i \le a_i & i = n+1,...,n+s: \text{inputs} & \\
\quad\quad L_i \le x_i \le U_i & i = 1,..,n &
\end{array}
}
$$

Fig. 2. The Primal Problem: minimize area given timing constraints for each gate.

spectively. α_i is the weighting for a component size. The delay of a component is its output $D_i = (\hat{r}_i / x_i) \cdot C_{downstream}$ resistance multiplied by the downstream capacitance:. L_i and U_i are the lower and upper bounds on the size of component *i*. The problem is formulated with arrival time constraints on the components, and primary outputs, and not on the worst-case path. By introducing one Lagrange multiplier per arrival time constraint in **a** , we get

$$
\begin{aligned}
L_\lambda(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{n} \alpha_i x_i + \sum_{j \in input\,(0)} \lambda_{j0} \left(a_j - A_0\right) \\
+ \sum_{i=1}^{n} \sum_{j \in input(i)} \lambda_{ji} \left(a_j + D_i - a_i\right) \\
+ \sum_{i=n+1}^{n+s} \lambda_{mi} \left(D_i - a_i\right)
\end{aligned}
\quad (1)
$$

Then the Lagrangian relaxation sub problem (LRS/λ) is:

$$
\begin{aligned}
LRS/\lambda: \ \text{Minimize} \ L_\lambda(\mathbf{x},\mathbf{a}) \\
\text{subject to} \ L_i \le x_i \le U_i \ \ \text{i} = 1,...,\text{n}
\end{aligned}
\quad (2)
$$

There are three sets of unknowns: the set of component sizes **x**, the set of arrival times **a**, and the set of Lagrange multipliers **e**. The Kuhn-Tucker conditions require $\partial L_\lambda / \partial a_i = 0$ at the optimal solution for $1 \le i \le n+s$. Rearranging LRS/λ, we get:

$$\begin{aligned} L_\lambda(\mathbf{x},\mathbf{a}) = & \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{t}\left(\lambda_{i0} - \sum_{j\in input(i)} \lambda_{ji}\right) a_i \\ & + \sum_{i=1}^{n}\left(\sum_{k:i\in input(k)} \lambda_{ik} - \sum_{j\in input(i)} \lambda_{ji}\right) a_i \\ & + \sum_{i=n+1}^{n+s}\left(\sum_{k:i\in input(k)} \lambda_{ik} - \lambda_{mi}\right) a_i \\ & + \sum_{i=1}^{n}\left(\sum_{j\in input(i)} \lambda_{ji}\right) D_i - \sum_{j\in input(0)} \lambda_{j0} A_0 + \sum_{i=n+1}^{n+s} \lambda_{mi} D_i \end{aligned} \qquad \textbf{(3)}$$

Applying $\partial L_\lambda / \partial a_i = 0$, we get the following optimality conditions on the set of Lagrange multipliers,

$$\sum_{k\in output(i)} \lambda_{ik} = \sum_{j\in input(i)} \lambda_{ji} \quad \text{for } 1 \le i \le n+s \qquad \textbf{(4)}$$

That is, for all resizable components and primary input drivers, the sum of the Lagrange multipliers at the outputs must equal the sum of the Lagrange multipliers at the inputs.

Let $\boldsymbol{\Omega}_\lambda = \{\mathbf{e} \ge 0\text{: } \mathbf{e} \text{ satisfies optimality conditions}\}$. Using the above, and ignoring any constants, the problem is further reduced to

$$L_\lambda(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n+s} \sum_{j\in input(i)} \lambda_{ji} D_i$$

$$\begin{aligned} LRS/\lambda:\ & \text{Minimize } L_\lambda(\mathbf{x}) \\ & \text{subject to } L_i \le x_i \le U_i \quad i = 1,\dots,n \end{aligned} \qquad \textbf{(5)}$$

Next we will see how the above is solved.

3.2 Solving LRS/λ

For a given vector $\mathbf{e} \ge 0$, the problem is solved optimally by a greedy algorithm. The algorithm performs optimal local resizing until convergence. Local resizing involves solving for the optimal size of a component that minimizes $L_\lambda(\mathbf{x})$, while keeping the other sizes fixed. Let μ_i be the sum of Lagrange multipliers assigned to the inputs of component *i*.

$$\mu_i = \sum_{j\in input(i)} \lambda_{ji}$$

Let R_i and C_i be the upstream weighted resistance and the downstream capacitance of component *i*, respectively.

$$R_i = \sum_{j\in\ input(i)} \mu_j \frac{\hat{r}_j}{x_j} \qquad C_i = \sum_{k\in\ output(i)} \hat{c}_k x_k$$

Then we can rewrite $L_\lambda(\mathbf{x})$, focusing on a particular gate *i*:

$$L_\lambda(\mathbf{x}) = (R_i \hat{c}_i + \alpha_i) x_i + \mu_i \frac{\hat{r}_i}{x_i} C_i + \text{terms independent of } x_i$$

The local optimum for x_i is obtained at:

$$x_i^* = \min\left(U_i, \max\left(L_i, \sqrt{\frac{\mu_i \hat{r}_i C_i}{R_i \hat{c}_i + \alpha_i}}\right)\right) \qquad \textbf{(6)}$$

Local optimizations are carried out on components in topologically sorted order until no further improvement is possible. Note that this process is carried out for a given set of Lagrange multipliers. To achieve the minimum area for a given target delay, the optimal set of multipliers must be found. The next section will discuss the Lagrangian Dual Problem (LDP), which will enable us to iteratively solve for the optimal set of multipliers.

3.3 The Lagrangian Dual Problem (LDP)

Consider the Primal Problem given in Figure 2. The LDP of the PP is:

$$\boxed{\begin{aligned} & LDP:\ \text{Maximize } Q(\boldsymbol{\lambda}) \\ & \qquad\qquad \text{subject to } \boldsymbol{\lambda} \in \boldsymbol{\Omega}_\lambda \\ & \text{where } Q(\lambda) = \inf\left\{\sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n+s}\left(\sum_{j\in input(i)} \lambda_{ji}\right) D_i : L_i \le x_i \le U_i\right\} \end{aligned}}$$

Fig. 3. LDP: maximize the infimum (greatest lower bound) of *LRS/λ*.

For a formal discussion on the Dual, the reader is directed to Bazaraa, *et al.* [10]. For a vector **e**, *Q*(**e**) is a concave function over $\mathbf{e} \ge 0$, and is non-differentiable [10]. Thus, the authors of [5] utilize subgradient optimization [10], to solve LDP. In subgradient optimization, the gradient direction is substituted with a subgradient direction. In our application, starting from an arbitrary point $\mathbf{e_{init}}$ in Ω_λ, the solution to LRS/λ is obtained. Then for each arrival time, the subgradient is defined as the arrival time constraint (slack) evaluated at the current solution. Next the subgradient is multiplied by a step size ρ_k. The next point is obtained by adding the subgradient to each λ. Finally the new λ's are projected to the closest point in Ω_λ. Given that the step size has the following properties: $\lim_{k\to\infty} \rho_k = 0$ and $\sum_{k=1}^{\infty} \rho_k = \infty$, the procedure will converge to the optimum. The procedure is summarized in Figure 4. Note that the updating procedure for λ (for resizable components) does the following: for any input on the current critical path, λ for that input is unchanged; for any input on a non-critical path, λ for that input is decreased. Thus λ values dictate the criticality of paths. Note also from (6) that the size of component *i* is directly proportional to $\mu_i = \sum_{j\in input(i)} \lambda_{ji}$. Thus, components on a non-critical path are downsized.

```
Algorithm Solve LDP:
1. $k := 1$ /* step counter */
   $\boldsymbol{\lambda}$ := arbitrary initial vector in $\Omega_\lambda$
2. Solve $LRS/\lambda$
3. For the current set of arrival times $\mathbf{a}$
      for $i := 0$ to $n+s$ do
         for all $j \in$ input of $i$ do
            $\lambda_{ji} = \begin{cases} \lambda_{ji} + \rho_k (a_j - A_o) & \text{if } i = 0 \\ \lambda_{ji} + \rho_k (a_j + D_i - a_i) & \text{if } 1 \le i \le n \\ \lambda_{ji} + \rho_k (D_i - a_i) & \text{if } n+1 \le i \le n+s \end{cases}$
4. Project $\boldsymbol{\lambda}$ onto the nearest point in $\Omega_\lambda$
5. $k := k + 1$
6. Repeat steps 2 - 5 until convergence
```

Fig. 4. Algorithm *SolveLDP*.

For the initial λ vector **e** we assigned an $\mathbf{e_{init}}$ to primary outputs and applied the procedure in Figure 5, assuming equal distribution of μ_{out} over the input λ's. The mapping to Ω_λ (step 4, Figure. 4) requires that the sum of the input λ's equal the sum of the output λ's. We first get the current sum of λ's, or μ, of a component, and then compute how the total is distributed among each input. Then we apply the same distribution using the sum of the downstream λ's.

```
Project into $\Omega_\lambda$:
    for $i := 0$ to $n$ do
        $\mu_i = \sum_{j \in input(i)} \lambda_{ji}$
        $\mu_{out} = \sum_{q \in output(i)} \lambda_{iq}$
        for all $j \in input(i)$ do
            $percentage_j = \lambda_{ji} / \mu_i$
        for all $j \in input(i)$ do
            $\lambda_{ji} = percentage_j * \mu_{out}$
```

Fig. 5. Mapping of λ to the closest point in Ω_λ.

The final algorithm that solves *PP* is:

```
Algorithm Lagrange:
1. Call Solve LDP to obtain optimal $\lambda$
2. Solve $LRS/\lambda$
```

Fig. 6. Algorithm Lagrange utilizing subgradient optimization.

Next we will look at key motivations that led to the formulization of *Forge*.

4 Motivation

The delay model we are using is convex. Thus, for a given delay, other than the absolute minimum, there exist contours of equal delays. A contour means that the same delay can be achieved with different sets of sizes.

This is illustrated in Figure 7. Consider the path "tracking" the minimum area solution at each contour. This path mimics the operation of the *Lagrange* algorithm. The inner loop of the *SolveLDP* algorithm operates by incrementally changing the set of Lagrange multipliers (which reduces the delay) and then finding the minimum area for that given set of multipliers. Thus, extra iterations are being utilized to optimize intermediate steps for area. We anticipate great savings in run time if we could search the solution space as represented by the path from the right in Figure 7. Employing a steepest descent method, the delay target can be reached very quickly. The first solid arrow terminating at the delay contour represents this. The second arrow terminating at the minimum area point represents area minimization on the delay contour.

A method to facilitate minimizing area on the delay contour is explored by considering the inverter chain given in Figure 8. From the optimal conditions on the Lagrange multipliers (4), we have $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda$. Also, as we have single input gates, $\mu_i = \mu_j = \lambda$. Using $R_i = \lambda(\hat{r}_j / x_j)$, if we examine (6) (which solves *LRS/λ*) disregarding the boundary values we get,

$$x_i = \sqrt{\frac{\lambda \hat{r}_i C_i}{\lambda(\hat{r}_j / x_j) \cdot \hat{c}_i + \alpha_i}} \quad \textbf{(7)}$$

Then solving for λ:

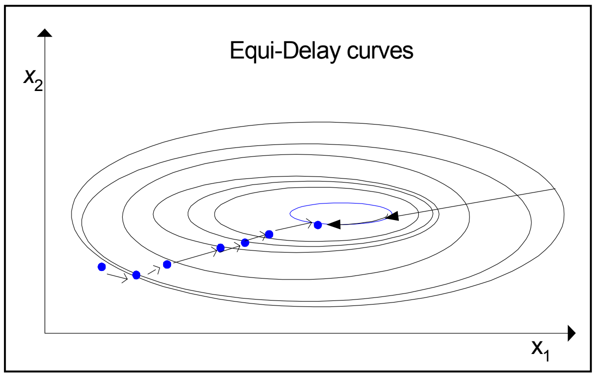

Fig. 7. Equal delay contours in gate-size space (x_1 and x_2 represent certain gate sizes). The solid dots are minimum area solutions for a given delay. The path from the left tracking the solid dots represents *Lagrange*, taking multiple stops on the delay contours, while the other path from the right represents *Forge*, which takes fewer stops at contours.

$$x_i^2 = \frac{\lambda \hat{r}_i C_i}{\lambda(\hat{r}_j / x_j)\hat{c}_i + \alpha_i} \quad \Rightarrow x_i^2 \lambda(\hat{r}_j / x_j)\hat{c}_i + \alpha_i x_i^2 = \lambda \hat{r}_i C_i$$

Dividing by x_i

$$\lambda(\hat{r}_j / x_j)\hat{c}_i x_i + \alpha_i x_i - \lambda(\hat{r}_i / x_i) C_i = 0$$

$$\lambda((\hat{r}_i / x_i) C_i - (\hat{r}_j / x_j)\hat{c}_i x_i) = \alpha_i x_i$$

$$\lambda = \frac{\alpha_i x_i}{(\hat{r}_i / x_i) C_i - (\hat{r}_j / x_j)\hat{c}_i x_i} \quad \textbf{(8)}$$

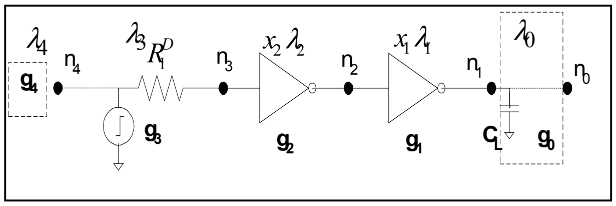

Fig. 8. Example with back-to-back inverters.

Thus, it can be seen that the sizes x_i, x_j imply a value for λ. Consider the point at which the steepest descent has found the target delay. Then using (8) we compute values for all λ's. This provides a very good initial vector $\mathbf{e_{init}}$.

Another motivation is that though subgradient optimization theoretically converges to the optimum solution, practically it has been seen that this is not the case [10][11]. The authors of [10] state that careful fine-tuning of step size choices is required for good performance. The authors of [11] state "although of simple convergence conditions, the convergence of subgradient methods can consume a long time for some instances. The subgradient optimization is very sensitive to the initial values for the multipliers and the rules applied for step size controlling". We illustrate this in Table 1. For each design we used two initial vectors $\mathbf{e_{init}}$. The values shown are the λ values for the primary outputs (all other λ are derived using (4)). We report the runtime for each step size multiplier used. (*Inf* denotes divergence). We used several step size multipliers, multipliers ρ_k that meet the requirements for convergence, and multipliers ρ that do not meet convergence requirements (k is the iteration count).

Design	e_{init}	ρ_k		ρ	
		$1/1e7k$	$1/5e7k$	1/1e7	1/5e7
C17	1e-4	0.03	Inf	Inf	0.01
	1e-5	0.00	Inf	Inf	0.00
b1	1e-4	Inf	Inf	0.28	1.10
	1e-5	Inf	Inf	0.30	Inf.
cc	1e-4	Inf	Inf	0.44	1.82
	1e-5	Inf	Inf	Inf	2.16
alu2	1e-4	Inf	Inf	Inf	Inf
	1e-5	Inf	Inf	Inf	Inf
des	1e-4	362.76	Inf	Inf	Inf
	1e-5	Inf	Inf	Inf	Inf

Table 1. Lagrange run times, illustrating convergence problems due to the strong sensitivity to the step size multipliers and initial points. Reported run time for each multiplier step size and initial point pair are in seconds.

It can be readily seen from Table 1 that subgradient optimization is very sensitive to the start point and the choice of step sizes, and the best set of parameters differs from design to design. As the best, or even a workable pair of start point and multiplier choices are not known *a priori*, in our implementation we try a range of initial points and step size multipliers. These are in the two forms $\rho = 1/k\beta, \rho = 1/\beta$, k is the current iteration. Each start point, multiplier pair is tried out for a time period of 30 minutes. If algorithm convergence is not seen within this period, then we restart with another pair.

In the next section we present *Forge,* which exploits the relationship between the sizes and the Lagrange multipliers, and is void of difficult-to-predict initial conditions and constant factors.

5 Forge

As seen in the previous section, the subgradient search used to solve *LDP* is very crucial for performance. Many approaches have been suggested to improve subgradient search [10]. Exploiting the convex nature of the problem at hand, we will present a method that provides an initial $\mathbf{e}_{\mathbf{init}}$ that will be close to the optimal $\mathbf{e}^*$. Then using this $\mathbf{e}_{\mathbf{init}}$ as a starting point we

```
Algorithm : Forge
    SteepestDescentSearch
    ContourSearch
    LagrangeM (Modified Lagrange Algorithm)
```

Fig. 9. Algorithm *Forge.*

will invoke Lagrangian relaxation with a modified subgradient approach, *LagrangeM*. Our approach is independent of constant factors, and uses the delay slack at each gate in a more intuitive manner than that given in Figure 4.

The algorithm is outlined in Figure 9. The algorithm consists of three steps. The first uses steepest descent to arrive as close as possible to the desired delay contour. Then the algorithm attempts to search along the delay contour for the best area solution. Using the set of sizes from the end of this process, at the next step we estimate values for the initial Lagrange multipliers, and then invoke the relaxation algorithm. In practice these estimates were close to the optimal values, resulting in fast convergence of the relaxation, which is the last step.

5.1 Steepest Descent Search

The first step in *Forge* is to arrive at the desired delay contour. We use a steepest-descent, gradient-based approach. The drawback of using such an approach is that steepest-descent is not directed toward a particular delay target, but simply minimizes the delay. To address this problem, when we apply the steepest-descent in topological order, we will attempt to size the primary output drivers to meet the current delay target at the output. If this is not feasible, we size the drivers optimally. Finally, the steepest descent is stopped when the delay is less than the target delay.

Consider (6), which sizes x_i to its local optimum value. If we disregard the upper and lower bounds on x_i (only to simplify the presentation in this paper), the area α_i, and the Lagrange multiplier μ_i (as we are only interested in minimizing delay at this step), we arrive at the value of x_i that minimizes delay.

$$x_i^* = \sqrt{\hat{r}_i C_i / R_i \hat{c}_i} \quad \textbf{(9)}$$

We apply (9) to each gate in topological order, except for the primary output drivers. For the primary output drivers we attempt to arrive as close to the target delays by sizing the drivers to meet the delay target. Consider the simple inverter chain in Figure 8. The total delay is given by,

$$D = R_1 \hat{c}_2 x_2 + (\hat{r}_2 / x_2) \cdot \hat{c}_1 x_1 + (\hat{r}_1 / x_1) \cdot C_L \quad \textbf{(10)}$$

Let a_3 be the arrival time at node n_3. Then the delay is written as,

$$D = a_3 + (\hat{r}_2 / x_2) \cdot \hat{c}_1 x_1 + (\hat{r}_1 / x_1) \cdot C_L \quad \textbf{(11)}$$

For a given target delay, $D = A_0$, we can compute the value of x_1 that will result in the desired delay:

$$(\hat{r}_2 / x_2) \cdot \hat{c}_1 x_1 + (\hat{r}_1 / x_1) \cdot C_L = A_0 - a_3$$

Multiplying by x_1 and, rearranging we get the quadratic equation

$$(\hat{r}_2 / x_2) \cdot \hat{c}_1 x_1^2 - (A_0 - a_3) x_1 + \hat{r}_1 C_L = 0 \quad \textbf{(12)}$$

(12) can be solved if: $(A_0 - a_3)^2 - 4(\hat{r}_2 / x_2)\hat{c}_1 x_1^2 \hat{r}_1 C_1 \geq 0$. If this is not the case then we use (9) to optimally size the driver. The algorithm is shown in Figure 10.

5.2 Contour Search

Using (12) and (8) we can formulate a routine that explores the contour and converges to the optimal solution. The routine for the simple inverter chain in Figure 8 is as follows. The chain is traversed in reverse topological order. From the inverter that drives the load we compute $\lambda_0 = \lambda_1$ using (8). Then we assume that $\lambda_2 = \lambda_1$ from the optimality criteria, and com-

```
Algorithm SteepestDescentSearch :
1 : while (delay > target_delay)
      for $i := t+1$ to $n$ do   /* all resizable components */
          for $j \in output(i)$
              if $1 \le j \le t$        /* primary outputs */
                  size to meet the target delay if possible, equation (14)
                  else minimize delay, equation (10)
              else
                  minimize delay, equation (10)
```

Fig. 10. Algorithm steepest-descent.

pute x_2 using (6). Arrival times are updated, and the last stage is resized to meet the target delay using (12), thus remaining on the delay contour. The above procedure is repeated until convergence.

For general circuit topologies it is not possible to get such a simple relationship between the Lagrange multipliers as for an inverter chain. But from the above we can see that sizes imply values for the multipliers. In the next section we will expand the algorithm to include general topologies. Using the fact that sizes imply values for the Lagrange multipliers, we predict reliable values for the Lagrange multipliers that will be the starting point of the relaxation algorithm (*LagrangeM*).

5.2.1 General networks of gates

For networks of gates, it is not possible to form a simple relationship between the Lagrange multipliers. To facilitate explo-

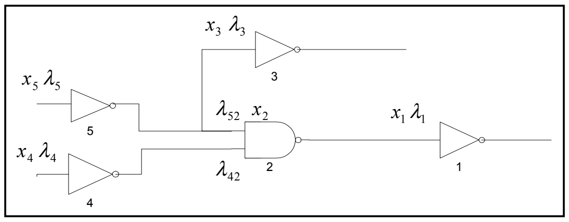

Fig. 11. General network.

ration of the contour in order to estimate values for $\mathbf{e}_{\mathbf{init}}$, we make the following assumption for an n-input gate i :

$$\text{if } \mu_i = \sum_{j \in input(i)} \lambda_{ji}, \ \text{ then } \lambda_{ji} = \mu_i / n$$

We assume that the total μ for a component is equally distributed across the $\lambda's$ related to each input of that component. For example, consider component 2 in Figure 11. From the optimality criteria we get: $\lambda_{52} + \lambda_{42} = \mu_1 = \lambda_1$. Then assuming a uniform distribution we get $\lambda_{52} = \lambda_{42} = \mu_1 / 2$.

The Lagrange multipliers are now computed as follows. Consider component 2 of the design in Figure 11. From (5), considering x_2 :

$$L_\lambda(\mathbf{x}) = \alpha_2 x_2 + \left(\lambda_5 \frac{\hat{r}_5}{x_5} + \lambda_4 \frac{\hat{r}_4}{x_4} \right) \hat{c}_2 x_2 + \mu_2 \frac{\hat{r}_2}{x_2} C_2$$
$$+ \text{ terms independent of } x_2$$

Using our assumption of equal distribution:

$\lambda_{42} = \lambda_{52} = \mu_2 / 2$ and that $\lambda_4 = \lambda_{42}$ and $\lambda_5 = \lambda_{52}$ we get:

$$L_\lambda(\mathbf{x}) = \alpha_2 x_2 + \left(\frac{\mu_2}{2} \frac{\hat{r}_5}{x_5} + \frac{\mu_2}{2} \frac{\hat{r}_4}{x_4} \right) \hat{c}_2 x_2 + \mu_2 \frac{\hat{r}_2}{x_2} C_2$$
$$+ \text{ terms independent of } x_2$$

Setting $\partial L / \partial x_2 = 0$, multiplying by $-x_2$ and solving for μ_2 :

$$\alpha_2 x_2 + \frac{\mu_2}{2} \left(\frac{\hat{r}_5}{x_5} + \frac{\hat{r}_4}{x_4} \right) \hat{c}_2 x_2 - \mu_2 \frac{\hat{r}_2}{x_2} C_2$$

$$\mu_2 = \frac{\alpha_2 x_2}{\left(\frac{\hat{r}_2}{x_2} C_2 - \frac{1}{2} \left(\frac{\hat{r}_5}{x_5} + \frac{\hat{r}_4}{x_4} \right) \hat{c}_2 x_2 \right)} \qquad \textbf{(13)}$$

This has the same form as (8). For an n-input component i we have:

$$\mu_i = \frac{\alpha_i x_i}{\left(\frac{\hat{r}_i}{x_i} C_i - \frac{1}{n} Rs_i \hat{c}_i x_i \right)} \qquad \textbf{(14)}$$

Rs_i is the sum of the upstream resistances. Then for each input gate we assume that this value is evenly distributed. That is, $\lambda_{ji} = \mu_i / n_i = \lambda_i$, where $j \in input(i)$ and n_i is the number of inputs for gate i

By rearranging (14) and solving for x we get:

$$x_j = \sqrt{\frac{\lambda_i \hat{r}_j C_j}{\left(\alpha_j + \lambda_i Rs_j \hat{c}_j \right)}} \quad j \in input(i) \qquad \textbf{(15)}$$

The final algorithm that estimates λ values is given in Figure 12.

The assumption of equal distribution assigns equal importance to paths that fan-in to that component. This could lead to sizing components such that the load drivers cannot be sized to meet the required delay. For example, the estimation may size the upstream components too much or too little, ultimately causing the delay to deviate too far from the target. Finally, the assumption of equal distribution may not lead to the minimum area solution since we are disregarding the criticality of timing paths. However, by following the procedure in Figure 12 we get a good estimate of an initial vector $\mathbf{e}_{\mathbf{init}}$ for the Lagrangian Relaxation.

In summary, first we reach a delay contour as close as

```
Algorithm ContourSearch :
1 :  for $i := t+1$ to $n$ do
         for $k \in output(i)$   /* $i$ is not an output load driver */
             compute $\mu_k$, using (14)
             $\mu_i = \sum_{k \in output(i)} \lambda_{ik}$
         for $j \in input(i)$ and $j \le n$   /* $j$ is not a primary input driver */
             $\lambda_{ji} = \mu_i / N_i$   $N$ # of inputs for $i$
             compute $x_j$, using (15)
2 : /* now try to place the solution on the desired delay contour */
       for $i := 1$ to $t$ do
           for $j \in input(i)$ do
               size output load driver, $x_j$, to meet delay target, equation (12)
3 : repeat steps 1 and 2 until convergence or the current
    solution is no longer on the contour
```

Fig. 12. Contour search.

possible to the target delay using gradient-based minimization. Then we explore this contour (seeking the minimum area solution) by enforcing a uniform distribution of the $\lambda's$. Finally, with the set of sizes and the Lagrange multipliers they imply, we call Lagrangian Relaxation with a modified subgradient search, which is the subject of the next section.

5.3 Modified Subgradient Search (*LagrangeM*)

In section 4 we showed that Lagrangian Relaxation coupled with subgradient optimization suffered from two problems: 1) not knowing what a good value for $\mathbf{e}_{\mathbf{init}}$ is, and 2) the great difficulty in determining the proper factors used to control the step size adjustments to the multipliers. In the previous sections we have addressed the initial value problem by a method used to estimate $\mathbf{e}_{\mathbf{init}}$. However, we are again faced with the task of step-wise adjustments to the multipliers to arrive at the optimum solution. One of the main drawbacks in the method used in Figure 4 is that the dampening factor β is fixed throughout the optimization process. We propose the following procedure, given in Figure 13, for the incremental improvement of the λ's.

$$\lambda_{ji} = \begin{cases} \lambda_{ji} \times (a_i / A_0) & \text{if } i = 0, j \in input(i) \quad /\text{*outputs*}/ \\ \lambda_{ji} \times (a_j / (a_i - D_i)) & \text{if } 1 \le i \le n, j \in input(i) \\ \lambda_{ji} \times (D_i / a_i) & \text{if } n+1 \le i \le n+s, j \in input(i) \quad /\text{*inputs*}/ \end{cases}$$

Fig. 13. Step-wise improvement procedure for the λ's.

This procedure is more sensitive to local delay information, by providing a dampening factor that is dependent on the local delay information.

The $\mathbf{a}_i$'s are the current set of arrival times at the outputs of each component, the $\mathbf{a}_j$'s are the arrival times at the inputs to each component, $\mathbf{D}_i$ the delay through each component, and A_0 is the target delay. We ran *Lagrange* and *Forge* on 30 benchmark circuits. In the next section we will see how the two algorithms compared to each other.

6 Results

As the *Lagrange* algorithm requires tuning to get the best runtime, to make an equitable comparison, we tried a range of β values and initial points, for intervals of 30 minutes. If algorithm convergence was not seen, then we switched to another β, initial point pair. Table 2 summarizes the results. The numbers reported as area are sum of the gate widths normalized by the result from *Lagrange*.

		Lagrange		Forge		Impr.
Design	cells	area	CPU(s)	Area	CPU(s)	runtime
C17	6	1	0.310	0.973	0.010	31
b1	9	1	1.360	0.998	0.010	136
cm138a	23	1	2.670	0.999	0.020	134
cm42a	20	1	4.160	1	0.020	227
cm151a	38	1	12.860	1.002	0.030	429
cm150a	52	1	6.990	0.998	0.060	140
cu	57	1	16.020	0.985	0.060	267
cc	60	1	8.600	0.999	0.060	143
B9	131	1	16.040	1	0.170	94
apex7	257	1	524.180	0.996	0.650	806
C880	467	1	83.860	0.989	9.930	90
alu2	516	n/a	Inf	n/a	1.390	n/a
too_large	800	n/a	Inf	n/a	2.100	n/a
apex6	703	n/a	Inf	n/a	1.850	n/a
C1355	791	1	1697.77	0.992	4.870	349
rot	753	n/A	Inf	n/a	13.540	n/a
vda	935	n/a	Inf	n/a	4.880	n/a
frg2	1070	n/a	Inf	n/a	8.210	n/a
C2670	1310	n/a	Inf	n/a	3.130	n/a
t481	1439	1	124.690	0.9685	2.910	43
C3540	1424	n/a	Inf	n/a	7.300	n/a
apex5	1707	n/a	Inf	n/a	9.640	n/a
dalu	1921	1	1261.14	0.9914	6.030	209
k2	2215	n/a	Inf	n/a	33.250	n/a
seq	2724	n/a	Inf	N/a	56.890	n/a
C5315	2529	n/a	Inf	n/a	33.250	n/a
i10	3280	n/a	Inf	n/a	39.030	n/a
C6288	3798	n/a	Inf	n/a	45.760	n/a
C7552	3994	1	300.94	0.9789	11.520	26
des	6160	1	1305.39	0.993	86.13	15
Average						196

Table 2. Runtime comparisons.

From Table 2 it can be seen that *Forge* converges at a much faster rate than *Lagrange*. The final values for the x 's (the total area) are within numerical "noise" of *Lagrange* for all benchmarks. Also note that of the 30 benchmarks, *Lagrange* did not converge for 14 of the designs. On average, *Forge* is nearly 200 times faster than *Lagrange* and has no convergence problems.

7 Delay Model

To demonstrate the versatility of our algorithm, we applied a more accurate delay model, and took rise and fall delays into consideration. We then compared the quality of our sizing results with AMPS, a commercially available sizing tool from Synopsys. All cells in our library were inverting. The slope-based delay and output slew models are:

$$Delay_{fall} = (a_f \cdot x_p + b_f \cdot C_{load} + c_f \cdot \tau_{input,rise}) / x_n$$
$$Delay_{rise} = (a_r \cdot x_n + b_r \cdot C_{load} + c_r \cdot \tau_{input,fall}) / x_p$$
$$Slew_{fall} = (a_{ft} \cdot x_p + b_{ft} \cdot C_{load} + c_{ft} \cdot \tau_{input,rise}) / x_n$$
$$Slew_{rise} = (a_{rt} \cdot x_n + b_{rt} \cdot C_{load} + c_{rt} \cdot \tau_{input,fall}) / x_p$$

The nMOS and pMOS widths are x_n and x_p , respectively, and we assumed a uniform size for all transistors in a pull-up or pull-down network. τ is the input slew rate. The constants $a_f, b_f, c_f, a_r, b_r, c_r, a_{ft}, b_{ft}, c_{tf}, a_{rt}, b_{rt}, c_{rt}$ were determined using the non-linear curve fitting software NLREG [12]. The characterization was over transistor widths from 0.5um to 12um, input slew rates from 20 to 200 ps and output loads up to 500fF. Similar approaches have been successfully applied in [13]. Table 3 summaries the average error and the standard deviation of the error, for the gates in our library. The average error for delay and slew-rate includes error averages for rise and fall transitions. Note that delay models are in posynomial form, which has been shown to be convex. We present results on 5 benchmarks and compare with the results from AMPS.

Gate	Delay		Slew	
	$\overline{\text{error}}$	σ_{n-1}	$\overline{\text{error}}$	σ_{n-1}
INV	4.89%	3.75%	5.13%	3.47%
NAND2	6.57%	5.64%	6.59%	4.87%
NOR2	5.19%	4.42%	5.12%	3.78%
NAND3	7.64%	7.99%	7.02%	6.56%
NOR3	6.87%	6.35%	5.43%	4.55%
NAND4	9.12%	10.13%	4.55%	4.93%
NOR4	8.85%	8.76%	5.73%	4.32%

Table 3. Accuracy of Delay model.

The gate sizing constraint that we employed with *Forge* and AMPS was to require uniform sizing of all pMOS (nMOS) transistors in the pull-up (pull-down) network.

The results for sizing are shown in Table 4. Note that *Forge* has a significant run time advantage over AMPS, averaging 15X faster, while achieving the same delay targets and using similar total transistor area, approximated by the sum of device sizes. The delays reported were obtained from PathMill static timing analysis runs. Note that *Forge* is optimal with respect to the delay model used for the gates. As was shown in Table 3, the delay model we used leaves room for improvement. This explains the reasonably small (percentage wise) area differences between *Forge* and AMPS.

Design	cells	Amps			Forge			run
		Delay (ns)	Area (um)	CPU (s)	Delay (ns)	Area (um)	CPU (s)	time Impr
b9	129	1.548	219		1.548	219		
		1.092	221.06		1.095	220.65		
		0.762	259.8	22.17	0.748	311.34	0.420	53
C880	482	3.136	815.00		3.136	815.00		
		2.416	847.00		2.416	874.97		
		2.224	924.75	42.68	2.221	958.79	4.520	9
alu2	496	3.301	795.00		3.301	795.00		
		2.859	824.64		2.873	868.21		
		2.846	811.65	42.26	2.828	919.62	10.460	4
Apex6	1029	1.767	1652.00		1.767	1652.00		
		1.486	1656.50		1.552	1660.34		
		1.427	1660.77	50.70	1.427	1698.59	17.990	3
C5315	2379	3.1781	3813.00		3.1781	3813.00		
		2.506	3841.01		2.541	3841.49		
		2.191	4069.36	128.76	2.24	4200.27	16.380	8
C6288	5494	7.387	8818.00		7.387	8818.00		
		5.992	8863.55		6.010	8873.17		
		5.734	8936.59	2672.78	5.751	9421.65	207.260	13
Average								15

Table 4. *Forge* sizing solutions compared with AMPS. Each design was given two delay targets. Reported CPU times are the total for both targets.

8 Conclusions

In this paper we presented *Forge*, an algorithm that enhances the Lagrangian relaxation method, by very quickly predicting reasonable values for the Lagrange multipliers required in the optimization. This was achieved by a gradient-based minimization to obtain the desired delay target, followed by a search on the equal delay contour. We have shown that the sizes of the components imply a value for a Lagrange multiplier, thus enabling the estimation. In the previous formulation of the Relaxation algorithm, values for the multipliers were obtained using a subgradient method. However, this approach is highly dependent on an initial value and constant factors. These factors are shown to contribute to erratic behavior in the subgradient search. Unfortunately, acceptable values for these factors are highly specific to the design and target-delay. On average there is about a 200X improvement in the run time for *Forge* over *Lagrange*.

Finally, we also showed that *Forge* provided a significant run time advantage over the AMPS tool from Synopsys, averaging 15 times faster.

9 Acknowledgements

We are grateful for the financial support provided by NSF, the SRC, MARCO, CDADIC, Boeing/DARPA, and Intel Corporation. The authors are grateful to Larry McMurchie and Miodrag Vujkovic for helpful discussions.

10 References

[1] W.C. Elmore, “The transient analysis of damped linear networks with particular regard to wideband amplifiers.” *Journal of Applied Physics*, vol. 19, no. 1, pp. 55-63, 1948.

[2] J. Rubinstein, P. Penfield, and Mark Horowitz, “Signal delay in RC tree networks,” *IEEE Trans. on CAD*, vol. CAD-2, no. 3, pp. 202-210, July 1983.

[3] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S. M. Kang, “An exact solution of the transistor sizing problem for CMOS circuits using convex optimization,” *IEEE Trans. on CAD of IC’s and Systems*, vol. CAD-12, pp. 1621-1634, Nov 1993.

[4] J. P. Fishburn and A. E. Dunlop, “TILOS: A posynomial programming approach to transistor sizing,” *IEEE Trans on CAD*, pp. 326-328, Nov 1985.

[5] C. P. Chen, C. C. N. Chu, and D.F. Wong, “Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation,” *Proceedings, ICCAD*, pp. 617-624, Nov 1998.

[6] D. Marple, “Transistor size optimization in the Tailor layout system”, *Proceedings, DAC,* pp. 43-48, June 1989.

[7] Ronald E. Miller, *Optimization*, John Wiley & Sons, Inc, 2000.

[8] Iris Hui-Ru Jiang, Yao-Wen Chang, and Jing-Yang Jou, “Crosstalk-Driven Interconnect Optimization by Simultaneous Gate and Wire Sizing,” *IEEE Trans. on CAD of IC’s and Systems*, vol. 19, no. 9, pp. 999-1010, September 2000.

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

[10] M. S. Bazaraa, H. D. Sherali, C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, Second Edition, John Wiley and Sons, 1993.

[11] L. A. N. Lorena, E. L. F. Senne, “Improving traditional subgradient scheme for Lagrangian relaxation: application to location problems,” *International Journal of Mathematical Algorithms*, pp. 133-151, 1, 1999.

[12] J.E. Dennis, D. M. Gay, and R. E. Welsch, “An adaptive nonlinear least-squares algorithm,” ACM Transactions on Mathematical Software 7,3 Sept. 1981.

[13] K. Kasamsetty, M. Ketkar, and S. S. Sapatnekar, “A New Class of Convex Functions for Delay Modeling and their Application to the Transistor Sizing Problem,” IEEE Trans. on CAD of Integrated Circuits and Systems, Vol. 19, No. 7, pp. 779 - 788, July 2000.