Lagrangian Relaxation Based Gate Sizing With Clock Skew Scheduling - A Fast and Effective Approach

Ankur Sharma
ankur_sharma2@mentor.com
Mentor Graphics, Fremont, USA

David Chinnery
david_chinnery@mentor.com
Mentor Graphics, Fremont, USA

Chris Chu
cnchu@iastate.edu
Iowa State University, Ames, USA

ABSTRACT

Recent work has established Lagrangian relaxation (LR) based gate sizing as state-of-the-art providing the best power reduction with low run time. Gate sizing has limited potential to reduce the power when the timing constraints are tight. By adjusting the arrival times of clock signals (clock skew scheduling), the timing constraints can be relaxed facilitating more power reduction.

Previous LR attempts at simultaneous gate sizing and skew scheduling solved a minimum-cost network flow problem for updating the Lagrange multipliers in each LR iteration, and for optimality assumed continuous sizes with convex delay models. We propose an alternative approach, modifying a LR discrete gate sizing formulation with table lookup non-convex delay models, which are more accurate for modern process technologies. For the Lagrange multiplier update, we use a projection heuristic that is much faster than solving the minimum cost network flow problem.

On the ISPD 2012 gate sizing contest benchmark suite, our proposed approach outperforms the previous min-cost flow based approach by saving 5.3% more power and is 70x faster. Compared to sizing alone with the state-of-the-art LR gate sizer, skew scheduling with sizing saves 19.7% more power with a small runtime penalty.

CCS CONCEPTS

• **Hardware** → **Combinational synthesis**; **Circuit optimization**.

KEYWORDS

Lagrangian relaxation; discrete gate sizing; Vt assignment; clock skew; multi-threading; gate sizing contest

ACM Reference Format:
Ankur Sharma, David Chinnery, and Chris Chu. 2019. Lagrangian Relaxation Based Gate Sizing With Clock Skew Scheduling - A Fast and Effective Approach. In *2019 International Symposium on Physical Design (ISPD '19), April 14–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3299902.3309746

1 INTRODUCTION

In modern designs, power consumption has increased substantially as larger circuits are being integrated on a single chip while the technology continues to shrink. That results in high power density causing reliability challenges, large cooling cost in data centers, and quicker discharge of the batteries in mobile devices. Circuit performance is also limited by power due to higher power densities. Thus, reducing the power has become a major concern.

In physical design, gate sizing is one of the most frequently used circuit optimizations. Each gate can be implemented by several possible cell options, with different sizes and threshold voltages (Vth). Different cell options trade off area or power for delay. A gate-sizer has to choose a suitable cell for every gate to minimize the objective cost while meeting the design timing constraints.

Synchronous circuits have combinational gates forming *data paths* and sequential gates that receive the clock signal. The setup timing constraint requires the data path signal to reach the sequential gate before the arrival of the clock signal. If a data path violates such a constraint, a gate sizing tool would resize gates on the path to speed it up. An alternative is to delay (*skew*) the arrival of the clock at the sequential gate.

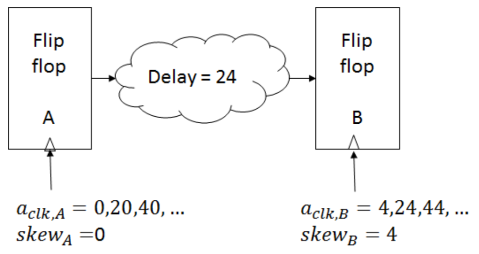

Figure 1: Example of clock skew to meet the timing constraint.

An example is shown in Figure 1. There are two flip-flops, A and B. They receive the same clock edge every 20 time units. There is a combinational block between them with a worst path delay of 24. The data signal is required to arrive at flip-flop B in ≤20. By delaying the arrival at the clock pin of B by 4 units, the timing constraint can be satisfied. Assigning clock skews to improve performance of the design is *clock skew optimization* [1]. A gate sizer with only cell sizes to vary is limited in how much power/area it can recover. Varying the clock skew provides an additional degree of freedom that gate sizers can utilize to improve the results as demonstrated previously [2–4]. These previous works make simplifying assumptions like continuity in sizes and convexity of delay models, and their approaches are very slow on larger designs.

In this paper, we propose a new flow for effective simultaneous gate sizing and clock skew scheduling that seamlessly integrates with the state-of-the-art gate sizing tool. For comparison, we also extend the previous approach from Wang et al. [3]. Wang et al. transformed the original timing graph to eliminate the skew variables

ISPD '19, April 14–17, 2019, San Francisco, CA, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6253-5/19/04…$15.00
https://doi.org/10.1145/3299902.3309746

and in the process introduced loops in the graph. The Lagrangian dual problem on the transformed graph was modeled as a min-cost network flow problem. Assuming continuity in the cell sizes and convexity of the delay models, their algorithm maximized the dual cost to realize primal optimality, if the primal problem was feasible. We detail how to apply their algorithm with more realistic constraints, i.e., discrete cell sizes and non-convex delay models, and discuss several limitations of this approach in the presence of these realistic constraints. We refer to this approach as *NetFlow*.

Our proposed approach for simultaneous gate sizing and skew scheduling is built upon a state-of-the-art LR gate sizing algorithm. We add skews variables to the LR formulation while keeping the timing graph directed acyclic. The timing graph must be acyclic to apply the projection based Lagrange multiplier update heuristic which is crucial to the performance of state-of-the-art gate sizing tools. We propose a new flow to simultaneously update skew along with the cell sizes and the Lagrange multipliers. We discuss our skew update strategy and propose modifications to the Lagrange multiplier update strategy in order to reflect the modified timing constraints caused by skewed clocks. We refer to this approach as *EGSS* which stands for *E*ffective *G*ate sizing with *S*kew *S*cheduling. For benchmarking, we use the ISPD 2012 gate sizing contest benchmark suite [5]. Compared to NetFlow, EGSS achieves an average of 5.3% additional power savings, and is 70x faster in total run time.

Our main contributions are as follows:

- We derive an LR formulation for the simultaneous gate sizing and clock skew scheduling problem while preserving the directed acyclic structure of the underlying timing graph.
- We propose a modified Lagrange multiplier update heuristic accounting for the skew.
- We present a simultaneous gate sizing and clock skew scheduling approach that seamlessly integrates with the state-of-the-art gate sizing tool. We empirically verify it's efficiency.
- In the context of discrete gate sizing with non-convex delays, we identify and empirically demonstrate several limitations of achieving primal optimality via dual maximization.

The rest of the paper is organized as follows. Section 2 summarizes previous work on LR gate sizing and sizing with skew scheduling. Section 3 formulates the problem. Sections 4 and 5 present NetFlow then EGSS. Section 6 discusses limitations of achieving primal optimality via dual optimality. Section 7 briefly describes the greedy refinement strategies that we use in this work. We present the experimental results in Section 8 and conclude in Section 9.

2 PREVIOUS WORK

The gate sizing problem has been researched for several decades. Earlier, most of the approaches assumed continuity in the gate sizes, convex delay models and experimented on relatively smaller designs. The gate sizing contests organized by Intel in ISPD 2012 [5] and ISPD 2013 [6] gave a fresh momentum to research in this area. The objective in the contests was to minimize the leakage power under the delay constraints. Contests were based on realistic constraints, discrete cell options and table lookup based non-linear delay models, and provided a suite of small to large designs having up to a million gates for benchmarking. Most of the post-contest publications in gate sizing utilized the contest framework for benchmarking and thus, greatly pushed ahead the state-of-the-art.

Some of the post-contest publications like [7] used sensitivity guided greedy metaheuristics to reduce timing violations and then reduce the power. Daboul et al. [8] modeled the gate sizing problem as a resource sharing problem. However, most of the gate sizers that were published used LR formulation [9–13]. Li et al. [9] first achieved minimum clock period and then recovered power using the min-cost network flow formulation. Ren et al. [14] also used network flow for discrete cell sizing, but neither of their formulations considered skew. Flach et al. [11] improvised on the projection based Lagrange multiplier update heuristic that was originally proposed in [15]. They demonstrated the least power results on the ISPD 2012 gate sizing contest designs. Sharma et al. [12] proposed a multi-threaded LR gate sizer and reported the least runtime, which they further improved in [13]. They proposed a simple and tunable framework for projection based Lagrange multiplier update which significantly improved the convergence. LR gate sizing using the projection heuristic has been demonstrated to yield designs with lower power and much smaller runtime compared to the other approaches. The original LR gate sizing idea is credited to [16].

Some of the previous works on simultaneous gate sizing and skew scheduling include [2–4, 17, 18]. Chuang et al. [2] directly solved the primal problem by formulating it as a linear programming problem using the piecewise-linear (PWL) approximations of the convex delays. Roy et al. [18] assumed continuous sizes and convex delays, and thus were able to minimize the Lagrangian subproblem simultaneously over size and skew variables using a bound constrained optimization solver. Without giving much details they claim to use the projection heuristic of [15] for updating Lagrange multipliers. Wang et al. [3] eliminated the skew variables from the primal problem in order for the Hessian of the primal objective to be positive definite so that optimality of their algorithm can be guaranteed. As a result, the timing graph could no longer be acyclic. Wang et al. maximized the dual cost by solving the min-cost network flow formulation of the Lagrangian dual problem. While they prove primal optimality under the assumptions that sizes are continuous and delay models are convex, these assumptions are not valid in modern design methodologies which would limit the effectiveness of their approach. Shklover et al. [4] accounted for the cost of implementing the clock skew via clock tree. Although they formulate a simultaneous discrete gate sizing and skew scheduling problem using LR, they mainly focus on clock tree optimization via dynamic programming. For sizing of the datapath gates and Lagrange multiplier update, they simply refer to the previous works [3, 16, 18]. The multiplier update strategies used in these previous works [3, 16, 18] are either too slow, especially on large designs, or have problem converging to a good solution [15].

3 PROBLEM FORMULATION

In order to formally define the problem, we use the notation tabulated in Table 1. The objective is to minimize the leakage power subject to the delay constraints, maximum load constraints, and maximum slew (transition time) constraints. Skew variables are bounded. Only combinational gates can change size, sequential gates are a fixed size. The primal problem is formally defined as:

Table 1: Commonly used notations.

Notation	Meaning
T	Target clock period
$\mathcal{G}$	Set of gates in the design
$\mathcal{X}_g$	Discrete set of cells for gate g
$x_g \in \mathcal{X}_g$	Current cell for gate g
$\mathcal{FF}$	Set of flip-flops in the design
$\mathcal{PO}$	Set of primary outputs in the design
w_k	Skew at flip-flop $k \in \mathcal{FF}$
$\mathcal{N}$	Set of nodes in the timing graph
$\mathcal{E}$	Set of timing arcs in the timing graph
$d_{ij}(\boldsymbol{x})$	Delay function of the timing arc (i, j)
λ_{ij}	Lagrange multiplier for the timing arc (i, j)
a_i	Arrival time at node i
a_{d_k}, a_{q_k}	Arrival times at D and Q pins of flip-flop k
$setup_k, d_{clk2q_k}(\boldsymbol{x})$	Setup and clock to Q delay of flip-flop k
$\lambda_{d_k}, \lambda_{q_k}$	Lagrange multipliers associated with the setup and the clock to Q delay timing arcs of flip-flop k
$gate\_power(x)$	Power of cell x
$p(\boldsymbol{x})$	Total power of the design
$max\_load(x)$	Maximum load capacity of cell x
$\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{a}, \boldsymbol{\lambda}$	Respective set of variables x, w, a and λ
$load_g(\boldsymbol{x})$	Capacitive load at the output of gate g
$slew_i(\boldsymbol{x})$	Slew at node i
max_slew	Maximum slew defined in the cell library

$$
\begin{array}{lllr}
\underset{\boldsymbol{x},\boldsymbol{a},\boldsymbol{w}}{\text{minimize}} & p(\boldsymbol{x}) & & \\
\text{subject to} & a_i + d_{ij}(\boldsymbol{x}) \le a_j & \forall (i,j) \in \mathcal{E} & \\
 & a_{po} \le T & \forall po \in \mathcal{PO} & \\
 & a_{d_k} \le T + w_k - setup_k & \forall k \in \mathcal{FF} & \\
 & w_k + d_{clk2q_k}(\boldsymbol{x}) \le a_{q_k} & \forall k \in \mathcal{FF} & (1) \\
 & load_g(\boldsymbol{x}) \le max\_load(x_g) & \forall g \in \mathcal{G} & \\
 & slew_g(\boldsymbol{x}) \le max\_slew & \forall g \in \mathcal{G} & \\
 & x_g \in \mathcal{X}_g & \forall g \in \mathcal{G} & \\
 & w_{min} \le w_k \le w_{max} & \forall k \in \mathcal{FF} &
\end{array}
$$

where minimization is over the set of discrete cell variables $\boldsymbol{x}$, continuous arrival time variables $\boldsymbol{a}$, and continuous skew variables $\boldsymbol{w}$. $p(\boldsymbol{x})$ is the sum of the power over all the gates, $p(\boldsymbol{x}) = \sum_{g \in \mathcal{G}} gate\_power(x_g)$. Since Wang et al. [3] did not consider the power cost of skew implementation, for a fair comparison against their NetFlow approach, we also do not account for the clock tree power in our formulation (1). For an example of how to incorporate clock power, refer [4]. Although our EGSS approach complements the work of Shklover et al. [4], we cannot compare against them because they used proprietary designs which are not available.

We use table lookup based non-linear, non-convex delay models for modeling cell arcs. Per ISPD 2012 contest framework, interconnect is modeled by a lumped capacitance without any resistance, so the net timing arcs have zero delay. Even if interconnects are modeled by distributed RC trees as in the ISPD 2013 contest, the

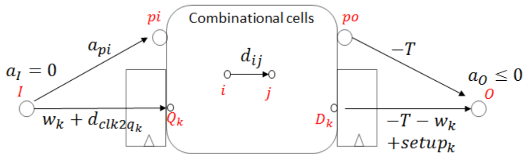

Figure 2: Timing graph for EGSS approach. Graph is directed acyclic with skew variables.

problem formulation and our approach would not change. With RC interconnects, the main challenge is the interconnect and the cell delay modeling which is beyond the scope of this work.

Timing constraints in the primal problem (1) are usually modeled by a timing graph. The underlying timing graph used by our proposed EGSS approach is shown in Figure 2. It was presented by Wang et al. in [3]. The graph has two dummy nodes and several dummy edges. Two dummy nodes are the global input I, and the global output O. There are dummy edges from I to output pins on flip-flops (Q_k); from I to primary inputs (pi); from input pins on flip-flops (D_k) to O; and, from primary outputs (po) to O. Although Wang et al. transformed it to eliminate skew variables and in the process introduced loops (directed cycles) in the timing graph, we instead propose to retain skews as variables and preserve the graph's directed acyclic structure. Further, Wang et al. had a backward edge from O to I with a weight of $-T$ which created more loops. Unlike that, to avoid loops in the timing graph, we adjust weights on the dummy edges so that the arrival times at I and O satisfy the following properties: $a_I = 0$ and $a_O \le 0$, as shown in Figure 2. In the presence of a clock tree, our timing graph can be extended to include timing arcs from the clock tree without creating loops. But it is not clear how skew variables can be eliminated in the presence of timing constraints along the clock tree.

Following the methodology of Chen et al. for gate sizing [16], we also relax the timing constraints. To penalize violations in the timing constraints, each constraint is associated with a non-negative Lagrange multiplier. These multipliers indicate the timing criticality of the corresponding arc. For a given set of Lagrange multipliers $\boldsymbol{\lambda} \ge 0$, the Lagrange function can be defined as:

$$
\begin{aligned}
L_{\boldsymbol{\lambda}}(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{w}): \quad & p(\boldsymbol{x}) + \sum_{(i,j) \in \mathcal{E}} \lambda_{ij} \times \left(a_i + d_{ij}(\boldsymbol{x}) - a_j\right) \\
& + \sum_{po \in \mathcal{PO}} \lambda_{po} \times \left(a_{po} - T\right) \\
& + \sum_{k \in \mathcal{FF}} \lambda_{d_k} \times \left(a_{d_k} - T - w_k + setup_k\right) \qquad (2) \\
& + \sum_{k \in \mathcal{FF}} \lambda_{q_k} \times \left(w_k + d_{clk2q_k}(\boldsymbol{x}) - a_{q_k}\right)
\end{aligned}
$$

LR gate sizing formulations typically do not relax the maximum load and maximum slew constraints. We also propose not to relax the skew bounds (last constraint of (1)), since such violations are easy to compute. The Lagrangian dual function is the minimum value of the Lagrangian function over $\boldsymbol{x}$, $\boldsymbol{a}$ and $\boldsymbol{w}$, for given $\boldsymbol{\lambda}$,

$$
\begin{aligned}
g(\boldsymbol{\lambda}) = \underset{\boldsymbol{x},\boldsymbol{a},\boldsymbol{w}}{\text{minimize}} \quad & L_{\boldsymbol{\lambda}}(\boldsymbol{x},\boldsymbol{a},\boldsymbol{w}) & \\
\text{subject to} \quad & load_g(\boldsymbol{x}) \le max\_load(x_g) & \forall g \in \mathcal{G} \\
& slew_g(\boldsymbol{x}) \le max\_slew & \forall g \in \mathcal{G} \\
& x_g \in \mathcal{X}_g & \forall g \in \mathcal{G} \\
& w_{min} \le w_k \le w_{max} & \forall k \in \mathcal{FF}
\end{aligned}
\tag{3}
$$

The Lagrangian function is affine in arrival times, so for the dual function to be finite, the multipliers must satisfy the *flow constraints* shown in (4). This argument is due to Wang et al. [3].

$$
\sum_{\{v|(i,v)\in\mathcal{E}\}} \lambda_{iv} = \sum_{\{u|(u,i)\in\mathcal{E}\}} \lambda_{ui} \quad \forall i \in \mathcal{N}\backslash\{I,O\}
\tag{4}
$$

Denote by Ω the Lagrange multipliers satisfying (4):

$$
\Omega = \{\boldsymbol{\lambda}|\boldsymbol{\lambda} \text{ satisfies (4) and } \boldsymbol{\lambda} \ge 0\}
$$

Applying the flow constraints (4) to (2), arrival time variables can be eliminated, and by ignoring the constant terms involving T and $setup_k$, the Lagrangian function can be simplified as

$$
\begin{aligned}
P_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{x},\boldsymbol{w}): \quad & p(\boldsymbol{x}) + \sum_{(i,j)\in\mathcal{E}} (\lambda_{ij} \times d_{ij}(\boldsymbol{x})) + \\
& \sum_{k\in\mathcal{FF}} \lambda_{q_k} \times d_{clk2q_k} + \sum_{k\in\mathcal{FF}} \left(\lambda_{q_k} - \lambda_{d_k}\right) \times w_k
\end{aligned}
\tag{5}
$$

For $\boldsymbol{\lambda} \in \Omega$, the dual function can be re-written as the following minimization problem, the *Lagrangian relaxation subproblem* or $LRS_{\boldsymbol{\lambda}}$,

$$
\begin{aligned}
g(\boldsymbol{\lambda} \in \Omega) = \underset{\boldsymbol{x},\boldsymbol{w}}{\text{minimize}} \quad & P_{\boldsymbol{\lambda}}(\boldsymbol{x},\boldsymbol{w}) \\
\text{subject to} \quad & \text{constraints in (3)}
\end{aligned}
\tag{6}
$$

The Lagrange dual problem (LDP) maximizes the dual function (6),

$$
\begin{aligned}
\underset{\boldsymbol{\lambda}\in\Omega}{\text{maximize}} \quad & g(\boldsymbol{\lambda}) - \sum_{po\in\mathcal{PO}} \lambda_{po} \times T + \\
& \sum_{k\in\mathcal{FF}} \lambda_{d_k} \times (setup_k - T)
\end{aligned}
\tag{7}
$$

The LDP is solved iteratively. In each iteration, for given $\boldsymbol{\lambda}$, $LRS_{\boldsymbol{\lambda}}$ (6) is solved to update $\boldsymbol{x},\boldsymbol{w}$; and for given $\boldsymbol{x},\boldsymbol{w}$, LDP objective is maximized in the neighborhood of current $\boldsymbol{\lambda}$. Thus, $\boldsymbol{\lambda}$ are incrementally updated. With updated $\boldsymbol{\lambda}$, $LRS_{\boldsymbol{\lambda}}$ is solved again in the next LDP iteration (also referred to as LR iteration).

4 NETFLOW

Before describing our proposed EGSS approach in Section 5, we first discuss the NetFlow approach. The core idea of NetFlow (but not the name) is due to Wang et al. [3]. We make two changes. To optimally solve $LRS_{\boldsymbol{\lambda}}$, Wang et al. assumed continuous sizes and convex delay models. We extend their $LRS_{\boldsymbol{\lambda}}$ solver to apply it to discrete sizes and table lookup based delay models. Also we fill in the missing details for setting bounds on the Lagrange multipliers.

Wang et al. transformed the timing graph to eliminate skew variables from the set of primal variables. This transformation resulted in directed cycles in the timing graph, as shown in Figure 3. Therefore, their Lagrangian function $L^{NF}_{\boldsymbol{\lambda}}(\boldsymbol{x})$ is different than ours

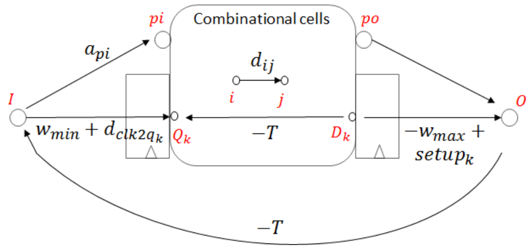

Figure 3: The timing graph for NetFlow which has directed cycles due to additional edges from D_k to Q_k from eliminating skews.

(2). See [3] for the complete expression of $L^{NF}_{\boldsymbol{\lambda}}(\boldsymbol{x})$. The Lagrangian relaxation subproblem for NetFlow ($LRS^{NF}_{\boldsymbol{\lambda}}$) is defined as:

$$
\begin{aligned}
g^{NF}(\boldsymbol{\lambda} \in \Omega): \underset{\boldsymbol{x}}{\text{minimize}} \quad & L^{NF}_{\boldsymbol{\lambda}}(\boldsymbol{x}) \\
\text{subject to} \quad & \text{constraints in (3)} \\
& \text{except } w \text{ bounds}
\end{aligned}
\tag{8}
$$

g^{NF} is the dual function. Then, LDP^{NF} is defined as,

$$
\underset{\boldsymbol{\lambda}\in\Omega}{\text{maximize}} \quad g^{NF}(\boldsymbol{\lambda})
\tag{9}
$$

Wang et al. proposed a minimum cost network flow formulation, $MCNF_{\boldsymbol{\lambda}}$, to solve LDP^{NF} in the neighborhood of current $\boldsymbol{\lambda}$,

$$
\begin{aligned}
\underset{\triangle\boldsymbol{\lambda}}{\text{minimize}} \quad & \left\langle -\nabla g^{NF}(\boldsymbol{\lambda}), \triangle\boldsymbol{\lambda} \right\rangle \\
\text{subject to} \quad & \triangle\boldsymbol{\lambda}_{lb} \le \triangle\boldsymbol{\lambda} \le \triangle\boldsymbol{\lambda}_{ub} \\
& \triangle\boldsymbol{\lambda} + \boldsymbol{\lambda} \in \Omega
\end{aligned}
\tag{10}
$$

where, $\nabla g^{NF}(\boldsymbol{\lambda})$ is the gradient of the dual function g^{NF} at $\boldsymbol{\lambda}$; and $\langle . \rangle$ denotes the dot product. To solve the gate sizing problem, the dual function is maximized over $\boldsymbol{\Delta\lambda} \in \Omega$. The intuition behind the algorithm is to iteratively improve the dual $g^{NF}(\boldsymbol{\lambda})$ by maximizing its first-order approximation in neighborhood of the current $\boldsymbol{\lambda}$.

Pseudo-code is shown in Algorithm 1. All Lagrange multipliers are initialized to 0 as that is a trivial dual feasible solution. $LRS^{NF}_{\boldsymbol{\lambda}}$ is solved to initialize the sizes. Since the multipliers are all zeros, the optimal solution to $LRS^{NF}_{\boldsymbol{\lambda}}$ is the minimum power subject to maximum load and slew constraints. Then, we initialize the skews to 0 and update the timing. The timing is needed to compute the bounds on $\triangle\boldsymbol{\lambda}$. Our bound computation strategy is discussed later in this section, and our skew update strategy is discussed in Section 5. After the initialization, LDP^{NF} is iteratively solved. In each iteration, firstly the lower and the upper bounds are computed. Then, $MCNF_{\boldsymbol{\lambda}}$ (10) is solved using the computed bounds. Optimal solution $\triangle\boldsymbol{\lambda}^*$ gives the steepest ascent direction of $g^{NF}(\boldsymbol{\lambda})$. Then, a line search is performed along $\triangle\boldsymbol{\lambda}^*$ in order to improve g^{NF} in 5 equispaced steps. At each step, $LRS^{NF}_{\boldsymbol{\lambda}}$ is solved. Based on the step-size that yielded the maximum g^{NF}, the multipliers are updated. Then, skews and timing are updated, only for the purpose of computing the bounds. Iterations continue until the change in g^{NF} is below a threshold, or a maximum number of iterations are reached. Since the problem is discrete and non-convex, this approach has several limitations which are discussed in Section 6. Consequently, even though the dual function converges, often there are some timing violations and scope for further power reduction. Therefore, we add a greedy refinement step at the end to try to recover any remaining timing violations and reduce power.

Algorithm 1 Pseudo code for NetFlow

```
1: $\boldsymbol{\lambda} = 0$. Solve $LRS_{\boldsymbol{\lambda}}^{NF}$ (8) for $\boldsymbol{x}$
2: Initialize skew to $w_{min}$. Update timing.
3: while $g^{NF}(\boldsymbol{\lambda})$ has not converged and iterations $< N$ do
4:     $(\triangle\boldsymbol{\lambda}_{lb}, \triangle\boldsymbol{\lambda}_{ub}) \longleftarrow$ compute bounds on $\triangle\boldsymbol{\lambda}$
5:     Solve $MCNF_{\boldsymbol{\lambda}}$ (10) for optimal $\triangle\boldsymbol{\lambda}^*$
6:     Perform line search on $g^{NF}(\boldsymbol{\lambda} + step \times \triangle\boldsymbol{\lambda}^*)$ ▷ $0 < step \leq 1$
       for an increase in $g^{NF}$.
7:     Update $\boldsymbol{\lambda}$
8:     Update skew. Update timing (for bound computation)
9: Greedy refinements
```

Solving $LRS_{\boldsymbol{\lambda}}^{NF}$. The strategy in [3] to solve the subproblem (8) assumes continuity in sizes and convexity of delay models. With discrete sizes and non-convex delay models, it becomes a difficult combinatorial problem. We adapt the LRS solver routine from the discrete LR gate sizing tool [9]. Although it does not guarantee optimality, its variants are the basis of state-of-the-art gate sizers [11, 13]. In this routine, all gates are traversed in the forward topological order. For each gate, assuming other gates are fixed, all the cell sizes are evaluated and the cell is chosen which minimizes the objective. To compute the objective quickly, timing is propagated only among the fanin and the fanout gates. To topologically order the cyclic timing graph, we cut it at the flip-flops. We also apply multi-threading techniques from [12] to parallelize the LRS solver.

Bound Computation. Wang et al. [3] did not give details on how to set the bounds. Based on the insights from the state-of-the-art LR gate sizing works, we propose to set the upper and lower bounds on $\triangle\lambda_{ij}$ for each combinational cell timing arc (i, j), as follows:

$$\begin{aligned}\triangle\lambda_{ij,ub} &= \max\left\{\frac{\lambda_{ij} \times |(q_j - a_i - d_{ij})|}{T}, \bar{\boldsymbol{\lambda}} \times M\right\} \\ \triangle\lambda_{ij,lb} &= \max\left\{-\lambda_{ij}, -\triangle\lambda_{ij,ub}\right\}\end{aligned} \tag{11}$$

where, q_j is the required time at node j; $\bar{\boldsymbol{\lambda}}$ is the average value of multipliers over all the arcs; and M is a tuning parameter. We make the upper bound proportional to the current value of the Lagrange multiplier and the slack along the arc. If either of them is large, that indicates the need for large changes in the multiplier. For near-critical arcs, the upper bound can be very small which slows convergence. Hence, we ensure that the upper bounds are at least of the order of the average multiplier value across the design. Lower bounds are set to guarantee non-negative multipliers. For some arcs, including the arc from O to I, it is not necessary to set explicit bounds as their multiplier values are determined by other incident arcs, being constrained by the Ω space.

5 EGSS: EFFECTIVE GATE SIZING WITH SKEW SCHEDULING

In this section we describe our approach for simultaneous gate sizing and skew scheduling. We call it *EGSS*. We modify portions of a state-of-the-art LR gate sizer. In particular, we re-use the LRS solver from the existing gate sizer, propose a new skew update strategy, and extend the projection based heuristic for the Lagrange multiplier update which is much faster than solving the minimum cost network flow problem as done in NetFlow.

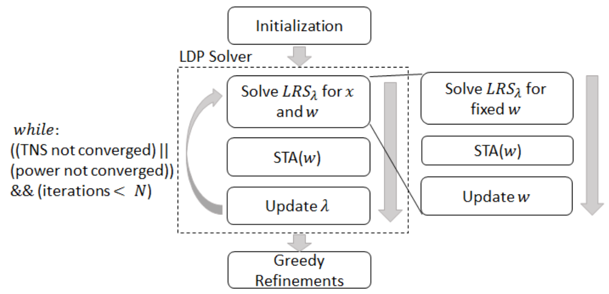
Initialization
LDP Solver
Solve LRS_λ for x and w
STA(w)
Update λ
Solve LRS_λ for fixed w
STA(w)
Update w
while: ((TNS not converged) || (power not converged)) && (iterations $< N$)
Greedy Refinements

Figure 4: Flow of our proposed EGSS. STA is static timing analysis.

Figure 4 shows the overall flow of EGSS. It has three stages: initialization, solving LDP, and greedy refinements. Initially, gates are sized to minimum power cell sizes subject to the maximum load and slew constraints; skews are initialized to the minimum values; timing is updated; and Lagrange multipliers are initialized to one and projected onto the Ω space. Unlike NetFlow, because of the multiplication in the Lagrange multiplier update which is discussed later in this section, the multipliers cannot be initialized to zero.

The LDP solver fixes most timing violations and reduces the power. In each iteration, it solves the $LRS_{\boldsymbol{\lambda}}$ (6) over size and skew variables, updates timing, and updates the multipliers. During the first few iterations, multipliers are updated to reduce total negative slack (TNS). The TNS is the total of all the timing violations in the second and third constraint of (1). Once the timing violations are reduced, multipliers are updated to focus on reducing power. There are extra checks in the LRS solver to avoid timing violations. Like NetFlow, greedy refinements are performed in the last stage.

5.1 Solving $LRS_{\boldsymbol{\lambda}}$

In LR gate sizing, sizes are the only variables in the Lagrangian relaxation subproblem (LRS). The LRS is a tough combinatorial problem, and adding skew variables for skew scheduling makes it even more difficult. Heuristically, we solve $LRS_{\boldsymbol{\lambda}}$ (6) separately for sizes and skews. The objective of $LRS_{\boldsymbol{\lambda}}$ from (5) can be split into two functions: a function of sizes, and a function of skews, $P_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{x}, \boldsymbol{w}) = H_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{x}) + Q_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{w})$ where,

$$\begin{aligned}H_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{x}) &= p(\boldsymbol{x}) + \sum_{(i,j)\in\mathcal{E}} \lambda_{ij} \times d_{ij}(\boldsymbol{x}) + \sum_{k\in\mathcal{FF}} \lambda_{q_k} \times d_{clk2q_k}(\boldsymbol{x}) \\ Q_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{w}) &= \sum_{k\in\mathcal{FF}} \left(\lambda_{q_k} - \lambda_{d_k}\right) \times w_k\end{aligned} \tag{12}$$

In our scheme to solve $LRS_{\boldsymbol{\lambda}}$, we first minimize $H_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{x})$ assuming skews are fixed, then update timing, and lastly update skews based on the $Q_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{w})$ function - see Figure 4. Note that skews do not affect $H_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{x})$, and sizes do not affect $Q_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{w})$, so they can be optimized separately. Shklover et al. [4] had a similar framework to solve $LRS_{\boldsymbol{\lambda}}$, but used different algorithms to minimize $H_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{x})$ and update skew.

To minimize $H_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{x})$, we use the LRS solver from [12]. This LRS solver is similar to that for the NetFlow approach in Section 4, but it performs a *local slack check* during every cell size evaluation. Each new candidate cell size is checked for local timing degradation. Only a cell size that either does not or minimally degrades the local timing can be assigned to a gate. This check is necessary to keep the timing violations under control while reducing power. This check is not applied while solving $LRS^{NF}_{\boldsymbol{\lambda}}$ (8) because the NetFlow objective is dual maximization, and reduction in TNS and power (primal objective) is expected as a consequence.

5.2 Skew Update

Consider minimizing $Q_{\boldsymbol{\lambda}\in\Omega}(\boldsymbol{w})$ (12) to determine skews subject to the bounds. Since the objective is linear in skews, minimization is trivial. For each flip-flop k, if $\lambda_{q_k} > \lambda_{d_k}$, then $w_k = w_{min}$, else $w_k = w_{max}$. Intuitively, if the Q pin is more timing critical than the D pin, then reduce the skew to reduce timing violations at the Q pin. If the D pin is more timing critical, then increase the skew to increase the slack at the D pin. Note that always setting skew to either of the extreme values can cause oscillations. Hence, we propose the following skew update strategy,

$$\begin{aligned}\triangle w_k &= \frac{slack_q - slack_d}{2} \\ w_k &= \max\ \{w_{min},\ \min\{w_{max}, w_k + \triangle w_k\}\}\end{aligned} \tag{13}$$

5.3 Modified Lagrange Multiplier Update

The projection based Lagrange multiplier update strategy is crucial for fast convergence and final solution quality. A simple framework for this multiplier update was proposed in [13]. We extend it to account for the skew impact. If skew at a flip-flop is positive, then the timing paths ending at its D pin have a larger required time which reduces the rate at which multipliers increase. Pseudo code for the Lagrange multiplier update is shown in Algorithm 2.

Algorithm 2 Lagrange multiplier update algorithm

```
for each flip-flop $k$ do
    $\lambda_{d_k} = \lambda_{d_k} \times \left(1 + \frac{a_{d_k}+setup_k-T-w_k}{T}\right)^K$
for each primary output $po$ do
    $\lambda_{po} = \lambda_{po} \times \left(1 + \frac{a_{po}-T}{T}\right)^K$
for each timing arc $(i,j)$ do
    $\lambda_{ij} = \lambda_{ij} \times \left(1 + \frac{a_i+d_{ij}-q_j}{T}\right)^K$        ▷ $q_j$: required time at $j$
Projection to satisfy flow constraints. Refer [15]
```

6 NETFLOW VS EGSS: LIMITATIONS OF OPTIMIZING THE PRIMAL PROBLEM VIA DUAL MAXIMIZATION

The NetFlow approach is based on results from Lagrangian duality theory [19], that under certain conditions which generally hold for the convex and continuous primal problems, primal optimality can be attained by maximizing the dual function. However, for non-convex discrete gate sizing which is NP-hard [20], NetFlow has limitations:

- A non-zero duality gap
- Minimizer $\boldsymbol{x}^*$ of the Lagrangian while solving LRS for the optimal set of dual variables $\boldsymbol{\lambda}^*$ may not be primal feasible
- Discreteness tends to cause oscillations

We explain these limitations with the help of a simple illustration of the process of dual maximization in the primal space.

Consider a single inverter circuit with cell size x as the only variable. Let power of the circuit be $p(x)$ and delay be $d(x)$. Figure 5 shows these values for different sizes of the inverter. Each point corresponds to a distinct size. Let $\lambda \geq 0$ be the Lagrange multiplier associated with the timing arc of the inverter. Then, the Lagrange function can be written as $L_\lambda(x) = p(x) + \lambda \times (d(x) - T)$. In the power-delay space, this is a line with slope $-\lambda$ and the intercept on the $p(x)$ axis is the value of the Lagrangian function. Solving LRS, or computing the dual function, is equivalent to minimizing $L_\lambda(x)$, i.e., pushing the line as low as possible as long as it passes through at least one design point. This process is illustrated in Figure 5.

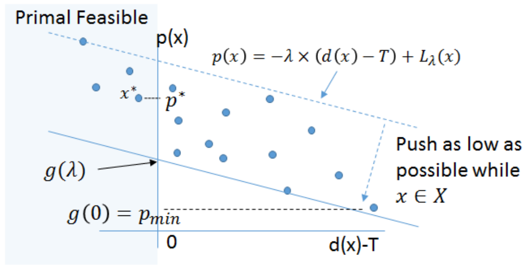
Primal Feasible
p(x)
$p(x) = -\lambda \times (d(x) - T) + L_\lambda(x)$
x^*
p^*
Push as low as possible while $x \in X$
$g(\lambda)$
$g(0) = p_{min}$
0
d(x)-T

Figure 5: Minimization of Lagrangian function $L_\lambda(x)$ for a single gate circuit is shown in the power-delay space parameterized by the discrete cell size x. X-axis is delay shifted to the right by T. Y-axis is the power. Each dot corresponds to a unique cell size x. Left of the power axis ($(d(x) - T \leq 0)$) is primal feasible. Minimum feasible power p^* and minimum possible power p_{min} are indicated above.

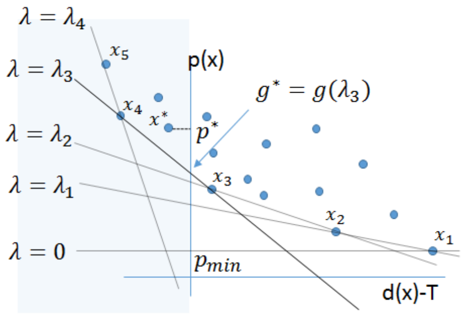

Figure 6: Maximization of dual function $g(\lambda)$ is shown for a single gate circuit in (a) primal space and, (b) dual space. g^* is the dual optimal attained at $\lambda = \lambda_3$. Due to non-convexity and discreteness there is a non-zero duality gap, $p^* - g^*$. For the same reasons, $L_{\lambda_3}(x)$ is minimized at x_4 and x_3. While x_4 is primal feasible, x_3 is not.

Table 2: A summary of the results for ISPD 2012 benchmarks suite [5] is shown for three flows: Sharma et al. [13] which is the baseline for comparison - it assumes a fixed skew; NF (NetFlow); and EGSS. For NetFlow and EGSS minimum and maximum skew bounds are 0 and 165ps. The final optimized netlist for each design from all three flows has zero timing violations.

Benchmark	Comb.	Clock	Leakage Power (W)			Power saved (%)		Total Runtime (min)			Speedup (X)	
	Gates	T, (ps)	[13]	NF	EGSS	Vs [13]	Vs NF	[13]	NF	EGSS	Vs [13]	Vs NF
DMA_slow	23109	900	0.135	0.111	0.104	23.1	6.6	0.07	7.90	0.08	0.9	94.0
pci_bridge32_slow	29844	720	0.098	0.073	0.072	26.9	2.2	0.09	8.70	0.10	0.9	88.0
des_perf_slow	102427	900	0.583	0.420	0.404	30.6	3.7	0.32	21.09	0.30	1.1	69.2
vga_lcd_slow	147812	700	0.329	0.310	0.310	5.9	0.2	0.44	28.35	0.44	1.0	64.0
b19_slow	212674	2500	0.569	0.577	0.556	2.2	3.7	0.83	45.75	1.24	0.7	37.0
leon3mp_slow	540352	1800	1.335	1.326	1.321	1.0	0.4	2.52	194.90	2.91	0.9	67.0
netcard_slow	860949	1900	1.763	1.762	1.762	0.1	0.0	2.35	343.90	2.82	0.8	122.0
DMA_fast	23109	770	0.245	0.173	0.137	44.3	20.8	0.08	9.20	0.10	0.8	92.0
pci_bridge32_fast	29844	660	0.141	0.083	0.078	44.7	6.2	0.10	9.20	0.11	0.9	84.0
des_perf_fast	102427	735	1.436	0.686	0.615	57.2	10.3	0.40	23.39	0.34	1.2	69.1
vga_lcd_fast	147812	610	0.417	0.318	0.316	24.3	0.8	0.56	27.90	0.50	1.1	55.3
b19_fast	212674	2100	0.729	0.823	0.682	6.5	17.1	1.13	19.58	1.61	0.7	12.2
leon3mp_fast	540352	1500	1.449	1.393	1.360	6.1	2.4	3.13	233.10	3.56	0.9	65.5
netcard_fast	860949	1200	1.846	1.804	1.800	2.5	0.2	3.33	237.05	3.98	0.8	59.5
Overall average			**0.791**	**0.704**	**0.680**	**19.7**	**5.3**	**1.10**	**86.43**	**1.29**	**0.9**	**69.9**

Now we explain dual maximization using Figure 6. Initially, when $\lambda = 0$, we know that $g(0) = p_{min}$, where p_{min} is the lowest possible power. $\lambda = 0$ corresponds to the horizontal line and $L_0(x)$ is minimized at x_1 as shown in the Figure 6. As λ increases, the slope of the line increases and the dual cost also increases. For $\lambda < \lambda_1$, x_1 minimizes $L_\lambda(x)$. At $\lambda = \lambda_1$, both x_1 and x_2 minimize the Lagrangian function. The dual function continues to increase with λ as long as $\lambda \leq \lambda_3$, and attains a maximum of g^* at $\lambda = \lambda_3$. At $\lambda = \lambda_3$, the Lagrangian function is minimized at x_3 and x_4, but only x_4 is primal feasible. So even if the dual optimum is attained, there is no guarantee that the primal feasible solution can be achieved. Observe also that the primal optimal value $p^* = p(x^*)$ is more than the dual optimal g^*. The gap $p^* - g^*$ is the duality gap.

These limitations are due to both non-convexity and discreteness. While NetFlow tries to maximize the dual cost and has the above-mentioned limitations, EGSS uses Lagrange multipliers to help attain primal feasibility and then reduce the power. EGSS rapidly increases the Lagrange multipliers initially to attain feasibility. That causes very low dual cost and high power. Then, using the local slack check while solving LRS, the design is forced to stay in the feasible region while Lagrange multipliers are reduced to recover power. NetFlow requires an optimal LRS solver, which it is not, to maximize the dual cost; but EGSS deliberately sacrifices the optimality of the LRS solver to maintain primal feasibility.

7 GREEDY TIMING AND POWER RECOVERY

Optimized designs obtained from the LDP solver have some timing violations and additional power that can be recovered. In the case of NetFlow, due to the limitations discussed in Section 6, there are large timing violations and a lot of power to be recovered. Hence, greedy refinements are more important for NetFlow. In either case, it is a common practice to apply a timing recovery followed by a power recovery algorithm - both are greedy and local in nature. The timing recovery and power recovery algorithms that we implemented have been adapted from [11] and [12]. Below we have summarized only the main steps for both the algorithms. For details refer to [11, 12].

For timing recovery, we sort all gates in the decreasing order of the number of timing critical end-points that are in the fan-out cone of each gate. Each gate is upsized in this order. In order to compute the change in TNS, timing is updated in an incremental fashion. If the TNS improves (reduces) then the new size is committed and the gates are re-sorted, otherwise the upsizing is undone and the next gate in the order is upsized. This process continues as long as the TNS is non-zero and it is reducing. During timing recovery we avoid reducing Vth because that significantly worsens the power.

To reduce power, we first try to increase Vth for each gate. If Vth cannot be increased without worsening TNS, then downsizing is considered. Gates are traversed in forward topological order.

8 EXPERIMENTS AND RESULTS

We used the ISPD 2012 gate sizing contest benchmark suite. We performed experiments on two quad-core Intel(R) Xeon(R) 3.50GHz CPUs. To solve the min-cost flow problem we used Gurobi [21]. For line search, we used a step size of 0.2 and evaluated 5 steps at uniform spacing. We used $M = 10$ for computing bounds. Our C++ code is multi-threaded using OpenMP [22], and we use 8 threads to solve LRS. We used PrimeTime [23] version E-2010.12, to verify the timing after NetFlow and EGSS. Each final design has zero TNS, and all constraints are satisfied.

We use the sizing results from [13], which also used 8 threads, as the baseline as it is the fastest and has competitive power results. This baseline only sizes the gates, with fixed zero skew at all clock pins. We use a minimum skew w_{min} of 0 and a maximum skew bound of 165ps for NetFlow and EGSS. Table 2 summarizes the results from all three flows. Compared to the baseline, EGSS saves on average 19.7% more power, because EGSS schedules skew simultaneously with gate sizing. The benefit of skew scheduling is more on designs with tighter timing constraints ('fast'), where

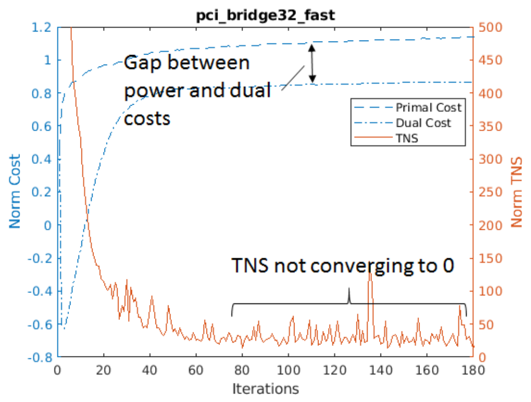

Figure 7: Primal cost (power), dual cost, and TNS profiles for NetFlow for pci_bridge32_fast. The left Y-axis is the normalized dual or primal cost, and the right Y-axis is the TNS normalized with respect to the target clock period of 660ps. Here, the maximum skew bound is zero to highlight the limitations of the NetFlow approach.

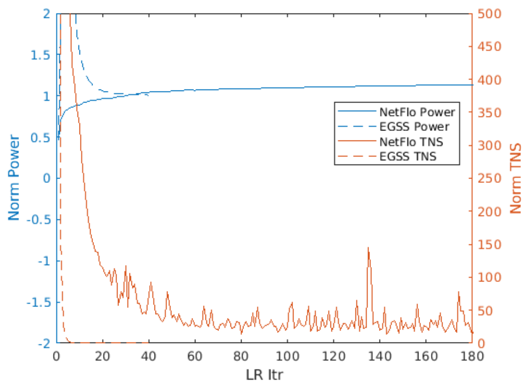

Figure 8: Power and TNS profiles for pci_bridge32_fast for NetFlow and EGSS. The maximum skew bound is set to zero for both.

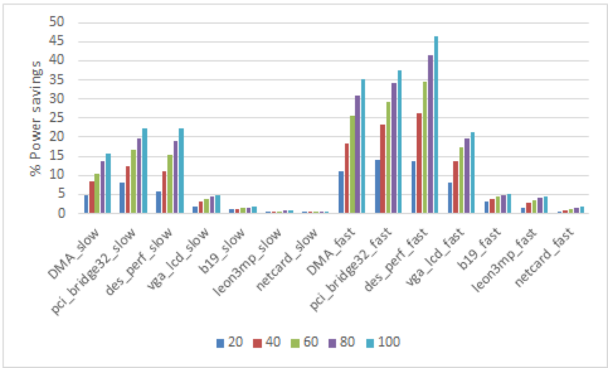

Figure 9: The percentage reduction in power for EGSS for different maximum skew bounds: 20ps, 40ps, 60ps, 80ps, and 100ps. The minimum skew bound is 0.

the average power saved is 26.5%. By carefully scheduling the skew, timing constraints can be satisfied without having to upsize the gates or decrease their Vth, both of which increase leakage power. EGSS has only 17% slowdown in the total runtime.

Compared to NetFlow, EGSS saves 5.3% more power and is 70X faster. In NetFlow, the greedy refinement stage accounts for an average of 5.4% power reduction (not shown in Table 2), whereas it accounts for only 0.4% power reduction in EGSS. This shows that the core idea behind EGSS is more effective than that of NetFlow.

The main reasons for larger runtime of NetFlow are: 1) Solving the min-cost flow problem is orders of magnitude more runtime expensive compared to the projection based Lagrange multiplier update shown in Algorithm 2. The latter has a linear time complexity in the number of gates. Using a network flow solver instead of Gurobi may be faster. 2) NetFlow has a slower convergence than EGSS, so it takes more iterations. 3) Each NetFlow iteration is more expensive due to solving LRS several times during the line search.

Figure 7 shows dual cost, power, and TNS profiles for NetFlow on pci_bridge32_fast. Although the dual cost converges to a maximum value, TNS does not converge down to 0. Also, there is a distinct gap between the dual cost and the primal cost (total gate power of the design), possibly due to a non-zero duality gap.

Figure 8 compares the TNS and power profiles from EGSS and NetFlow. Early on, EGSS has high power as TNS rapidly reduces. But then it recovers the power. It converges in less than 40 iterations with near-zero TNS and lower power than NetFlow. Across all the benchmarks, EGSS takes on average only 2 iterations to converge the TNS and an additional 18 iterations to reduce the power.

In another experiment with EGSS, we varied the maximum skew bound through 20, 40, 60, 80 and 100ps. Correspondingly, the average power savings compared to the baseline are 5.3%, 9.0%, 11.7%, 14.0% and 15.7%. Larger skew bounds allow larger adjustments in the clock arrival times at different clock pins which can be potentially used to save more power. Figure 9 shows the improvement in power with increasing skew bounds for each design. b19, leon3mp and netcard designs are already closer to minimum power, hence we do not observe as significant power savings as other designs.

9 CONCLUSION

Gate sizing is a crucial circuit optimization technique. The power reduction from gate sizing can be enhanced by allowing variable skew. We investigated two approaches for simultaneous gate sizing and skew scheduling. The network flow approach derives a Lagrangian dual problem from the primal problem and tries to maximize the dual objective. We detailed limitations arising from the non-convexity and the discreteness of the primal space, due to which dual maximization cannot guarantee primal feasibility and thus is sub-optimal, as shown by our experimental results. In the second approach, we extend the state-of-the-art high performance Lagrangian relaxation based gate sizing. We first make use of the variable skew to recover the bulk of the timing violations in just two iterations, on average. Then, we iteratively reduce power. In each iteration, skew is updated to redistribute the slack between each side of the flip-flop. Compared to the state-of-the-art gate sizer which treats skew as fixed, our proposed flow for simultaneous gate sizing and clock skew scheduling reduces power by an average of

19.7% on the ISPD 2012 gate sizing contest designs with only 17% higher run time.

10 ACKNOWLEDGMENTS

This work is partially supported by a grant from Mentor Graphics.

REFERENCES

[1] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39(7):945–951, July 1990.

[2] W. Chuang *et al.* Timing and area optimization for standard-cell vlsi circuit design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(3):308–320, 1995.

[3] J. Wang *et al.* Gate sizing by Lagrangian relaxation revisited. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):1071–1084, 2009.

[4] G. Shklover *et al.* Simultaneous clock and data gate sizing algorithm with common global objective. In *International Symposium on Physical Design*, pages 145–152, 2012.

[5] M.M. Ozdal *et al.* The ISPD-2012 discrete cell sizing contest and benchmark suite. In *International Symposium on Physical Design*, pages 161–164, 2012.

[6] M.M. Ozdal *et al.* An improved benchmark suite for the ISPD-2013 discrete cell sizing contest. In *International Symposium on Physical Design*, pages 168–170, 2013.

[7] J. Hu *et al.* Sensitivity-guided metaheuristics for accurate discrete gate sizing. In *International Conference on Computer-Aided Design*, pages 233–239, 2012.

[8] S. Daboul *et al.* Provably fast and near-optimum gate sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(12):3163–3176, 2018.

[9] L. Li *et al.* An efficient algorithm for library-based cell-type selection in high-performance low-power designs. In *International Conference on Computer-Aided Design*, pages 226–232, 2012.

[10] V. S. Livramento *et al.* A hybrid technique for discrete gate sizing based on lagrangian relaxation. *ACM Transactions on Design Automation of Electronic Systems*, 19(4):40, 2014.

[11] G. Flach *et. al.* Effective Method for Simultaneous Gate Sizing and V-th Assignment Using Lagrangian Relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(4):546–557, 2014.

[12] A. Sharma *et al.* Fast Lagrangian relaxation based gate sizing using multi-threading. In *International Conference on Computer-Aided Design*, pages 426–433, 2015.

[13] A. Sharma *et al.* Rapid gate sizing with fewer iterations of lagrangian relaxation. In *International Conference on Computer-Aided Design*, pages 337–343, 2017.

[14] H. Ren *et al.* A Network-Flow Based Cell Sizing Algorithm. In *The International Workshop on Logic Synthesis*, 2008.

[15] H. Tennakoon *et al.* Gate sizing using Lagrangian relaxation combined with a fast gradient-based pre-processing step. In *International Conference on Computer-Aided Design*, pages 395–402, 2002.

[16] C.-P. Chen *et. al.* Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(7):1014–1025, 1999.

[17] H. Sathyamurthy *et al.* 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*, 17(2):173–182, 1998.

[18] S. Roy *et al.* An optimal algorithm for sizing sequential circuits for industrial library based designs. In *Asia and South Pacific Design Automation Conference*, pages 148–151, 2008.

[19] S. Boyd *et al.* *Convex optimization.* Cambridge university press, 2004.

[20] W. Ning. Strongly NP-hard discrete gate-sizing problems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(8):1045–1051, 1994.

[21] Gurobi. http://www.gurobi.com/.

[22] L. Dagum *et al.* OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1):46–55, 1998.

[23] Synopsys PrimeTime user guide. http://http://www.synopsys.com.