Provably Fast and Near-Optimum Gate Sizing

Siad Daboul, Nicolai Hähnle, Stephan Held, and Ulrike Schorr

***Abstract*—We present a new approach for the cell selection problem based on a resource sharing formulation, which is a specialization of Lagrangian relaxation with multiplicative weight updates. For the convex continuous gate sizing problem, we can prove fast polynomial running times. This theoretical result also gives some justification to previous heuristic multiplicative weight update methods.**

For the discrete cell selection problem, where voltage thresholds can also be chosen, we employ the new algorithm heuristically and achieve superior results on industrial benchmarks compared with one of the previously best known algorithms, and competitive results on the ISPD 2013 benchmarks. Finally we demonstrate how the approach can be parallelized effectively achieving speed-ups of up to 16.

***Index Terms*—Gate sizing, threshold voltage, power optimization, algorithms, resource sharing, very large scale integration**

I. Introduction

Gate sizing and voltage threshold (V_t) optimization are essential to achieve timing closure and minimize the power dissipation of integrated circuits. The task to determine sizes and V_t levels for the gates in the netlist (*cell selection problem*) has been studied extensively in the literature. A large variety of approaches was proposed, for example linear programming [5], network flows [35], delay or slew budgeting [27], [13], sensitivity-based heuristics [16], [21], interior point methods [37], [2], or Lagrangian relaxation [3], [41], [44], [29], [9]. A recent survey can be found in [14].

Under two simplifying assumptions, namely 1) convex or convexifiable delay functions, e.g. RC delay models, 2) continuously sizable circuits and a single V_t level, the problem turns into a convex optimization problem, as shown in [8]. Under these assumptions and if a feasible solution exists, the cell selection problem can be solved close to optimality, e.g. by the projected subgradient method applied to the Lagrangian dual function [3], or with interior point methods [37], [2]. However, for large instance sizes interior point methods become impracticable as each iteration has a super-quadratic running time [41], [20]. In contrast, each iteration of the projected subgradient method solves a Lagrangian subproblem in near-linear time [6], but without good bounds on the number of subgradient steps.

The continuous relaxation can be used as a starting point for rounding to a discrete solution [17], [32].

Signoff delay functions are typically not convexifiable without loss of accuracy. Furthermore, despite attempts to combine sizing and multiple V_t levels in a single formulation [39], no useful convex relaxation is known so far. Thus, many successful approaches, in particular those achieving the best results on the ISPD 2013 benchmarks [28], use Lagrangian relaxation heuristically, making always discrete cell choices and using accurate delay functions [22], [24], [9], [36], [33], [34]. Interestingly, all these recent papers as well as [41] update the Lagrangian multipliers multiplicatively and not in subgradient direction.

In this paper we propose a new algorithm for the cell selection problem by modeling it as a resource sharing problem [26], which is solved by a special multiplicative update of Lagrange multipliers. Resource sharing has a long history in global routing, where it has recently been extended to timing-constrained global routing [15]. However, we use a different model of timing constraints proposed in [11]. Instead of modeling delay constraints through a static timing graph, we use an individual constraint for each signal path.

For the continuous gate sizing problem with convex delay functions our algorithm combines the advantages of provably fast sizing iterations as in the Lagrangian relaxation approaches with a provably small number of iterations as in interior point methods. This leads to an overall fast algorithm for the continuous gate sizing problem.

We use the new algorithm heuristically to tackle real-world cell selection instances with non-convexifiable delay functions, discrete cell sizes, and multiple V_t levels. The main contributions of this paper are:

- A new cell selection algorithm based on the resource sharing model, providing a provably fast overall convergence for the continuous gate sizing problem with convex delays.
- Natural extensibility to incorporate further constraints such as placement density or signal integrity.
- A practical implementation, where we use the new algorithm heuristically for discrete and non-convex instances.
- A new parallelization paradigm with high speedups demonstrated on up to 44 cores.
- On state-of-the-art microprocessors provided by our industrial partner IBM, our algorithm improves results of recent approaches [33], [34] at much shorter running times.
- Experimental results on the ISPD 2013 benchmarks demonstrate competitiveness with the best known results [9].

The remainder of this paper is organized as follows: In Section II we provide the problem formulation. Our main theoretical result, the modeling and solving of gate sizing as a resource sharing problem, is presented in Section III. Then, in Section IV, we present our adaption for the discrete cell selection problem, followed by experimental results in Section VI and conclusions.

S. Daboul and S. Held are with the Research Institute for Discrete Mathematics, University of Bonn, 53113 Bonn, Germany (e-mail: daboul@dm.uni-bonn.de, held@dm.uni-bonn.de). N. Hähnle is with Advanced Micro Devices, Germany (e-mail: nhaehnle@gmail.com). U. Schorr is with Cadence Design Systems, Cambridge, UK (e-mail: uschorr@cadence.com).

II. Problem formulation

We denote the set of gates with $\mathcal{G}$ and the set of feasible cell vectors with $X = (X_g)_{g \in \mathcal{G}}$, where X_g is the set of alternative cell types for gate $g \in \mathcal{G}$. A solution $x \in X$ specifies a cell type $x_g \in X_g$ for every gate $g \in \mathcal{G}$.

Furthermore, we assume that timing constraints are modeled by a timing graph $G = (V, E)$ and delay functions $delay_e : X \to \mathbb{R}$ for all edges $e \in E$ that specify the delay $delay_e(x)$ of e given a solution x. Signals start at one of the input vertices $V_{\text{in}} \subset V$, e.g. primary inputs or register outputs, and end at one of the output vertices $V_{\text{out}} \subset V$, e.g. primary outputs or register inputs. Furthermore, we assume that for each gate $g \in \mathcal{G}$ there is a function $power_g : X_g \to \mathbb{R}_+$ specifying the power consumption for choosing a specific cell type for g. The total power consumption is simply the sum of the gate power values. The cell selection problem can then be formulated as follows:

$$\min \quad power(x) := \sum_{g \in \mathcal{G}} power_g(x_g) \tag{1}$$

$$\begin{aligned} s.t. \quad a_v + delay_e(x) &\leq a_w && \forall\ e = (v, w) \in E \\ a_v &\geq 0 && \forall\ v \in V_{\text{in}} \\ a_v &\leq T && \forall\ v \in V_{\text{out}} \\ x &\in X, \end{aligned}$$

where $power_g(x_g)$ denotes the power consumption of cell type x_g, a_v the arrival time at $v \in V$, and T the desired clock cycle time. For a clearer presentation we make the simplifying assumption that all signals start at time 0 and all required arrival times equal a unique clock cycle time T.

An equivalent formulation forces each path delay to be bounded by T. To this end, let $\mathscr{P}$ denote the set of (inclusion-wise) maximal paths in G, i.e. the set of paths between a signal start and end point. Then (1) is equivalent to

$$\min \quad power(x) := \sum_{g \in \mathcal{G}} power_g(x_g) \tag{2}$$

$$\begin{aligned} s.t. \quad \sum_{e \in E(P)} delay_e(x) &\leq T \qquad \forall\ P \in \mathscr{P} \\ x &\in X, \end{aligned}$$

where $E(P) \subseteq E$ denotes the set of timing graph edges in path P. At a first glance, the path formulation (2) appears inferior due to the possibly exponential number of paths and, thus, constraints. However, we will show how to employ this model efficiently.

Note that in early design stages, gate sizing is rather a feasibility problem, where a solution maximizing the total negative slack (TNS)

$$\sum_{v \in V_{\text{out}}} \min\{0, T - a_v\} \tag{3}$$

is desired. As reported by [34], the resulting problem formulation leads to aggressive power minimization in failing paths that merge into more critical ones and, thus, are hidden from the endpoints. In such a scenario designers rather want to achieve a solution that also maximize the total negative path slack (TPNS)

$$TPNS(x) := \sum_{P \in \mathscr{P}} \min\left\{0, T - \sum_{e \in E(P)} delay_e(x)\right\}. \tag{4}$$

As it is #P-hard to compute (4) as shown by [25], we employ the true total negative slack TTNS introduced by [34] to account for failing paths that are hidden in the TNS. TTNS is defined as the sum of negative slacks at timing endpoints and all non-critical subpaths with negative slacks. It is similar similar to TPNS and has the advantage that it can be computed in linear time.

III. Continuous gate sizing

In this section we will present a new provably fast algorithm for the continuous gate sizing problem with convex delay functions. To this end, we make a few simplifying but usual assumptions, e.g. compare [8], [3].

- **A1** $X = [l, u]$, where $l, u \in \mathbb{R}^{\mathcal{G}}_{>0}$ and $1 \leq l \leq u$, where $u_{\max} := \max\{u_g : g \in \mathcal{G}\}$ is a small constant compared to the netlist size.
- **A2** For all $e \in E$, the function $delay_e(x)$ is convex and for all $g \in \mathcal{G}$ $power_g(x)$ is convex and nondecreasing.

Note that after scaling sizes, assumption A1 is usually fulfilled for the continuous relaxation of a finite gate library, where $u_{\max}$ is in the range of 60–130, e.g. 128 on the ISPD 2013 benchmarks [28].

Furthermore, by A1 we have $\max\{\frac{u_g}{l_g} : g \in \mathcal{G}\} \leq u_{\max}$.

Assumption A2 is fulfilled for the RC-delay model using the transformed model: $delay_e(e^x)$ with $e^x = (e^{x_g})_{g \in \mathcal{G}}$ in the domain $\log X := [\log l, \log u]$, where the logarithm of l and u is taken component-wise [8], [3]. With this variable transformation the sizing problem can be formulated as a convex program.

For our running time analysis we need a few more technical assumptions that are usually fulfilled in practice:

- **A3** $power(u)/power(l) \leq \hat{U}$,
- **A4** the gradient $\nabla power(x) = \nabla \sum_{g \in \mathcal{G}} power_g(x_g)$ is Lipschitz continuous with bound K_P,
- **A5** the gradient $\nabla \sum_{e \in E} delay_e(x)$ is Lipschitz continuous with bound K_D, and
- **A6** $\min_{x \in X, e \in E} delay_e(x) \geq d_{\min} > 0$ ($d_{min} \in \mathbb{R}$),

where $\hat{U}, K_P$ and K_D are technology-specific constants independent of the netlist. Note that A3 and A4 hold for prevalent linear power functions with $K_P = \max_{g \in \mathcal{G}, x \in X_g} power_g(x)$ and $\hat{U} \leq u_{\max}$. Exemplary, we will also prove in Section III-G that A5 holds for the RC-delay model if the fan-in and fan-out of each gate is bounded by a constant. Finally, A6 usually holds because a gate of interest will at least drive the input pin capacitance of another gate.

A. The Min-Max Resource Sharing Problem

We will model gate sizing as a min-max resource sharing problem, which is defined as follows (see [26]). An instance of the min-max resource sharing problem consists of a finite set of resources $\mathscr{R}$, a finite set of customers $\mathscr{C}$, and for each customer $c \in \mathscr{C}$ a convex set of feasible solutions $\mathscr{B}_c$. A convex function $\mathrm{usg}_c : \mathscr{B}_c \to \mathbb{R}^{\mathscr{R}}_{\geq 0}$ indicates the resource usages of the feasible solutions for each customer c.

The task is to find a solution for all customers approximately minimizing the maximum resource usage, i.e. to find $x_c \in \mathscr{B}_c$ for all $c \in \mathscr{C}$ approximately attaining

$$\lambda^* := \inf \left\{ \max_{r \in \mathscr{R}} \sum_{c \in \mathscr{C}} (\mathrm{usg}_c(x_c))_r : x_c \in \mathscr{B}_c \, (c \in \mathscr{C}) \right\}. \quad (5)$$

The algorithms that we will employ require an oracle function $f_c : \mathbb{R}^{\mathscr{R}}_{\geq 0} \to \mathscr{B}_c$ for each customer c that computes for resource weights $\omega \in \mathbb{R}^{\mathscr{R}}_{\geq 0}$ a feasible solution $x_c \in \mathscr{B}_c$ approximately minimizing the weighted resource usage of c:

$$\omega^\intercal \mathrm{usg}_c(x_c) \leq \eta \cdot \inf_{x'_c \in \mathscr{B}_c} \omega^\intercal \mathrm{usg}_c(x'_c). \quad (6)$$

Here $\eta \geq 1$ is the approximation factor of the oracle. In our case each $\mathscr{B}_c$ will be a compact set and the infimum in (6) is always attained. The problem

$$\min_{x_c \in \mathscr{B}_c} \omega^\intercal \mathrm{usg}_c(x_c)$$

can be considered as the Lagrangian relaxation of a certain feasibility problem for customer c with multipliers ω.

The fastest algorithm for solving the resource sharing problem was developed in [26] for an application in global routing in chip design. We refer to this paper for a broader overview of previous work.

At the core of the algorithm lies the multiplicative weight update method, which was introduced in [31] for combinatorial packing problems. The basic idea is to introduce a weight ω_r for each resource $r \in \mathscr{R}$. The algorithm iteratively makes calls to the oracles of the customers $c \in \mathscr{C}$. Based on the solution x_c returned by the oracle, the weight for each resource is updated multiplicatively

$$\omega_r \mapsto \omega_r \cdot e^{\gamma \cdot \mathrm{usg}_c(x_c)_r},$$

where $\gamma > 0$ is a parameter that will be chosen based on the desired solution quality and running time. If y_r specifies the sum of total resource usages over all previous iterations, the weight can be computed equivalently via

$$\omega_r = e^{\gamma \cdot y_r}. \quad (7)$$

B. Gate Sizing as Min-Max Resource Sharing Problem

We model continuous gate sizing as a resource sharing problem in the following way. Assume for the time being that we are given a power budget B. Then the following inequality system describes all solutions meeting this budget and all timing constraints:

$$\begin{aligned} power(x) &\leq B, \\ \sum_{e \in E(P)} delay_e(x) &\leq T \qquad \forall\, P \in \mathscr{P}, \\ x &\in X. \end{aligned} \quad (8)$$

By A2 the set of feasible solutions of (8) is convex. If we can approximate this feasibility problem, we can also approximate our original problem (2) by binary search on B. Here, an approximately feasible solution may violate timing and power constraints by a factor $(1+\epsilon)$. The feasibility problem (8) motivates the following resource sharing formulation.

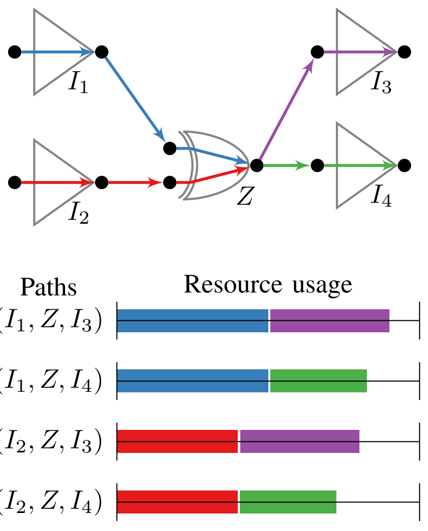

Figure 1. A sample instance together with the associated path resources and their usages.

1) Resources: Every inequality in (8) defines a resource:

- The first inequality is represented by a single resource *power*.
- Each inclusion-wise maximal path $P \in \mathscr{P}$ represents a *timing resource*.

Thus, the set of resources is $\mathscr{R} = \{power\} \cup \mathscr{P}$. This specific way to model timing resources $\mathscr{P}$ was introduced in the context of timing-constrained global routing in [43] and first solved efficiently in [11].

2) Customers: It seems natural to represent each gate $g \in \mathcal{G}$ by a customer, and to define that each customer consumes delay of each path resource that is influenced by its size, which typically would be all paths passing through the driver pins of the input nets of g. However, for most edges in G the delay depends on the sizes of several gates in a non-separable way, such that changing the size of one gate would change the delay usage of other gates. We overcome this difficulty by representing all gates by a single customer $\mathscr{C} = \{\mathcal{G}\}$ with feasible solution set $\mathscr{B}_{\mathcal{G}} = X$. The oracle problem for the single gate customer is equivalent to the Lagrangian subproblem in [3], as we will see in Section III-E.

3) Resource Usages: The resource usages are defined by the quotients of left and right hand sides in the inequalities of (8). More precisely, the *power resource usage* of the gate customer solution $x \in X$ is specified as

$$\mathrm{usg}_{\mathcal{G},power}(x) := \frac{power(x)}{B},$$

and the *timing resource usage* of a path resource P is defined as

$$\mathrm{usg}_{\mathcal{G},P}(x) := \frac{\sum_{e \in E(P)} delay_e(x)}{T}.$$

Path resources are visualized in Figure 1.

By the definition of the resource usages and (8) we have

Observation 1. *A solution $x \in X$ fulfills all timing constraints in (2) and (8) if and only if*

$$\mathrm{usg}_{\mathcal{G},P}(x) \le 1 \quad \textit{for all } P \in \mathscr{P}.$$

C. Computing all Resource Weights in Linear Time

In each iteration of the resource sharing algorithm we need to compute a vector $\omega \in \mathbb{R}^{\mathscr{R}}_{\ge 0}$ with weights for the power resource and each timing path resource. The weight depends on the total resource usage from the previous iterations. Let $y \in \mathbb{R}^{\mathscr{R}}_{\ge 0}$ be the vector of total resource usages just before the current weight update.

Computing the power resource weight is easy. Let y_{power} be the total power usage consumed in the previous iterations. According to (7), the power resource weight is given by

$$\omega_{power} = e^{\gamma \cdot y_{power}}.$$

For the path resources it was shown in [11] that despite their potentially exponential number, the resource weights can be computed implicitly in linear time by decomposing them into edge weights ω_e ($e \in E$). An oracle function for the gate customer computes for given resource weights $\omega \in \mathbb{R}^{\mathscr{R}}$ feasible sizes x for all gates such that the weighted resource usage

$$\omega_{power}\frac{power(x)}{B} + \sum_{P \in \mathscr{P}} \omega_P \frac{\sum_{e \in E(P)} delay_e(x)}{T} \tag{9}$$

is minimized up to a factor of $\eta > 1$. With implicit edge weights

$$\omega_e := \sum_{P \in \mathscr{P}:\ e \in E(P)} \omega_P, \tag{10}$$

the sum (9) can be rewritten as

$$\omega_{power}\frac{power(x)}{B} + \sum_{e \in E} \omega_e \frac{delay_e(x)}{T}. \tag{11}$$

Note that the weights $(\omega_e)_{e \in E}$ fulfill the flow conservation rule $\sum_{e \in \delta^-(v)} \omega_e = \sum_{e \in \delta^+(v)} \omega_e$ because they are derived from the path weights. As an interesting side effect, the Karush-Kuhn-Tucker (KKT) conditions are fulfilled by $(\omega_e)_{e \in E}$ without requiring an extra projection step as in the Lagrangian relaxation based algorithm described in [3]. However, our algorithm does not depend on the KKT conditions.

Now suppose we are in iteration $i+1$ of our algorithm and that the total usage of path $P \in \mathscr{P}$ from the previous i iterations is given by

$$y_P = \sum_{k=1}^{i} \xi^{(k)} \mathrm{usg}_{\mathcal{G},P}(x^{(k)}) = \frac{1}{T}\sum_{k=1}^{i} \xi^{(k)} delay_P(x^{(k)}),$$

where $x^{(k)} \in X$ is the gate sizing solution computed by the resource sharing algorithm in iteration k, and $\xi^{(k)} \in [0,1]$ are scale factors that we will specify in Section III-D. Similarly, we define total usages for implicit edge resources:

$$y_e := \frac{1}{T}\sum_{k=1}^{i} \xi^{(k)} \cdot delay_e(x^{(k)}) \tag{12}$$

for all $e \in E$. It turns out that the edge weights $(\omega_e)_{e \in E}$ in (10) can be computed from y_e in linear time.

Lemma 2. *(Hähnle [11]) The edge weights ω_e for $e \in E$ can be computed in each iteration in time $\mathcal{O}(|E| + |V|)$.*

Proof. By (7) and (10), we have

$$\begin{aligned}
\omega_e &= \sum_{P \in \mathscr{P}: e \in E(P)} \omega_P = \sum_{P \in \mathscr{P}: e \in E(P)} \exp(\gamma \cdot y_P) \\
&= \sum_{P \in \mathscr{P}_{[V_{\text{in}},v]}} \sum_{Q \in \mathscr{P}_{[w,V_{\text{out}}]}} \exp\left(\gamma \cdot \sum_{f \in P \cup Q \cup \{e\}} y_f\right) \\
&= \exp(\gamma \cdot y_e) \cdot \underbrace{\left(\sum_{P \in \mathscr{P}_{[V_{\text{in}},v]}} \exp\left(\gamma \cdot \sum_{f \in P} y_f\right)\right)}_{=: \, \omega_{\mathscr{P}_{[V_{\text{in}},v]}}} \\
&\quad \cdot \underbrace{\left(\sum_{Q \in \mathscr{P}_{[w,V_{\text{out}}]}} \exp\left(\gamma \cdot \sum_{f \in Q} y_f\right)\right)}_{=: \, \omega_{\mathscr{P}_{[w,V_{\text{out}}]}}} \\
&= \exp(\gamma \cdot y_e) \cdot \omega_{\mathscr{P}_{[V_{\text{in}},v]}} \cdot \omega_{\mathscr{P}_{[w,V_{\text{out}}]}},
\end{aligned}$$

where $\mathscr{P}_{[V_{\text{in}},v]}$ denotes the set of all paths from an input vertex to v and $\mathscr{P}_{[v,V_{\text{out}}]}$ denotes the set of all paths from v to an output vertex. All variables $\omega_{\mathscr{P}_{[V_{\text{in}},v]}}, \omega_{\mathscr{P}_{[v,V_{\text{out}}]}}$ ($v \in V$) can be computed in linear time by traversing G once in topological and once in reverse topological order according to EdgeWeights in Algorithm 1.

Algorithm 1 EdgeWeights$(\gamma, (y_e)_{e \in E})$

```
$\omega_{\mathscr{P}_{[V_{\text{in}},v]}} = 1 \quad (v \in V_{\text{in}}); \omega_{\mathscr{P}_{[v,V_{\text{out}}]}} = 1 \quad (v \in V_{\text{out}})$;
for $v \in V$ in topological order do
    $\omega_{\mathscr{P}_{[V_{\text{in}},v]}} = \sum_{(u,v) \in E} \left(e^{\gamma \cdot y_{(u,v)}} \cdot \omega_{\mathscr{P}_{[V_{\text{in}},u]}}\right)$;
end for
for $v \in V$ in reversed topological order do
    $\omega_{\mathscr{P}_{[v,V_{\text{out}}]}} = \sum_{(v,w) \in E} \left(e^{\gamma \cdot y_{(v,w)}} \cdot \omega_{\mathscr{P}_{[w,V_{\text{out}}]}}\right)$;
end for
for $(v,w) \in E$ in reversed topological order do
    $\omega_{(v,w)} = \exp(\gamma \cdot y_{(v,w)}) \cdot \omega_{\mathscr{P}_{[V_{\text{in}},v]}} \cdot \omega_{\mathscr{P}_{[w,V_{\text{out}}]}}$;
end for
return $\omega_E := (\omega_e)_{e \in E}$;
```

□

D. Overall Resource Sharing Algorithm for Gate Sizing

We now describe details of our adaption of the resource sharing algorithm from [26]. The algorithm is shown in Algorithm 2. It takes as input all timing constraints, a power budget, a SizingOracle returning gate sizes minimizing (11), a parameter γ and the number I of iterations.

The variable $x \in \mathbb{R}^{\mathcal{G}}_{\ge 1}$ stores a sum of gate sizing solutions, and $\Xi \in \mathbb{R}_{\ge 0}$ a number. In the end, $\frac{x}{\Xi} \in X$ will be the mean of gate sizing solutions found throughout all iterations. The variable y_{power} stores the total power resource usage and variables $y_e$the total implicit edge resource usage of $e \in E$ (defined in (12)),

Algorithm 2 Resource sharing algorithm for gate sizing

Input: An instance of the gate sizing problem, a power budget B, SIZINGORACLE(ω) for the gate customer, $\gamma > 0$, $I \in \mathbb{N}$.

Output: Convex combination of gate sizes $x \in \operatorname{conv}(X)$.

```
1:  $x \leftarrow 0,\ \Xi \leftarrow 0$;
2:  $y_e \leftarrow 0$ for all $e \in E$, $y_{power} \leftarrow 0$;
3:  for $i = 1, \dots, I$ do
4:      $\omega_{power} \leftarrow e^{\gamma \cdot y_{power}}$;
5:      $\omega_E \leftarrow$EDGEWEIGHTS$(\gamma, y_E)$;                (Section III-C)
6:      $x' \leftarrow$ SIZINGORACLE$(\omega)$;                  (Section III-E)
7:      $\xi \leftarrow \min\left\{\frac{B}{power(x')}, \frac{T}{||delay(x')||_\infty}\right\}$;
8:      if $\xi \geq 1$ then return $x'$;
9:      $y_e \leftarrow y_e + \xi \frac{delay_e(x')}{T}$ for all $e \in E$;
10:     $y_{power} \leftarrow y_{power} + \xi \frac{power(x')}{B}$;
11:     $x \leftarrow x + \xi \cdot x'$;
12:     $\Xi \leftarrow \Xi + \xi$;
13: end for
14: return $\frac{1}{\Xi} x$;
```

In each of the I iterations, the algorithm first computes weights for the power and for all path resources calling EDGEWEIGHTS, implicitly using edge usages and resources as described in Section III-C.

Then, it calls the oracle function SIZINGORACLE(ω) to compute a gate sizing solution x that approximately minimizes the total weighted resource consumption $\omega^\intercal \mathrm{usg}_{\mathcal{G}}(x)$ (11). This is a critical step and we will present its details in Section III-E.

If the current solution x' meets all constraints (8), we get $\xi \geq 1$ and the algorithm will stop and return x'. Otherwise, x' has some resource usages above one and the solution as well as the resource usages are scaled down so that their maximum usage is bounded by 1 (lines 9–10). Finally, the factors of the convex combination which is returned are updated (11–12).

In our resource sharing model there is only a single customer $\mathcal{G}$. Thus, we can omit a special while-loop in the original algorithm from [26] that re-iterates over-consuming customers multiple times before proceeding with the next customer. We will instead capture these special re-iterations a priori in the total number of iterations I. For a single customer the resource sharing algorithm from [26] is essentially the same as the scale-invariant multiplicative weights update algorithm from [11]. The number of iterations will be discussed in Section III-F.

E. Oracle Function for the Gate Customer

By setting $\lambda_{power} = \omega_{power}/B$ and $\lambda_e = \omega_e/T$, we can rewrite (9) as a Lagrange function similar to [3], [42], [24] with weighted power:

$$L(\lambda, x) := \lambda_{power} power(x) + \sum_{e \in E} \lambda_e delay_e(x). \tag{13}$$

For the RC-delay model, the Lagrange function can be minimized in polynomial time with a greedy algorithm as proposed in Chu and Wong [6]. Under certain assumptions, a solution $x \in X$ with $|(x_i^* - x_i)/x_i^*| \leq \epsilon$ for all $i = 1, \dots, n$ can be computed in $\mathcal{O}(n \log(1/\epsilon))$ time for $\epsilon > 0$. However, in our context we are interested in an approximation guarantee on the *value* of (13), whose scale depends on the exponentially growing weights.

Theorem 3. *Assume that A1–A6 hold and let T_{grad} be the time that is needed to compute the gradient $\nabla L(\lambda, x)$, which we also assume to dominate the time it takes for changing x in gradient direction. Let further $\hat{K} := \max\left(\frac{K_P}{power(l)}, \frac{K_D}{d_{\min}}\right)$. Then there exists an oracle for the gate customer that computes for $\omega \in \mathbb{R}^{\mathscr{R}}_{\geq 0}$ and $\eta > 1$ a solution $x \in X$ in time $\mathcal{O}\left(T_{grad} \hat{K} \frac{u_{\max}^2}{\eta - 1}\right)$ such that the weighted resource usage (11) of the gate customer is minimized up to a factor of η.*

Proof. To simplify notation, we consider the minimization of the transformed weighted resource usage $L(\lambda, x)$ in (13) instead of (11). For fixed resource weights λ its gradient $\nabla L(\lambda, x)$ is Lipschitz continuous in x by A4 and A5:

$$\begin{aligned} lip(\lambda) := \max_{x,y \in X} \frac{||\nabla L(\lambda, x) - \nabla L(\lambda, y)||_\infty}{||x - y||_\infty} \\ \leq \lambda_{power} K_P + \max_{e \in E} \lambda_e K_D. \end{aligned}$$

We apply the well-known conditional gradient method of Frank and Wolfe [10] to $L(\lambda, x)$. Starting with an initial solution $x^{(0)} \in X$, in each iteration k of this descent method a minimizer $s := \arg\min_{y \in X} \langle y, \nabla L(\lambda, x^{(k)}) \rangle$ of the linear approximation at $x^{(k)}$ is computed and a step from $x^{(k)}$ towards s is performed: $x^{(k+1)} := x^{(k)} + \theta^{(k)}(s - x^{(k)})$ with step size $\theta^{(k)} = \frac{2}{k+2}$. The linear minimization subproblem can be solved in linear time: For each entry s_g $(g \in \mathcal{G})$ we set

$$s_g = \begin{cases} l_g & \text{if } \nabla L(\lambda, x^{(k)})_g > 0 \\ u_g & \text{if } \nabla L(\lambda, x^{(k)})_g < 0 \\ x_g^{(k)} & \text{otherwise.} \end{cases}$$

Let $diam_X := \max_{x,y \in X} ||x - y||_\infty \leq u_{\max}$ be the diameter of X that is bounded by u_{max} by A1, and let $opt(\lambda) = \min_{x \in X} L(\lambda, x)$ be the minimum resource consumption for weights λ. The convergence analysis of the conditional gradient (see for example Jaggi [18]) yields that after $k \geq 1$ iterations

$$L(\lambda, x^{(k)}) - opt(\lambda) \leq 2 \frac{lip(\lambda)}{k+2} diam_X^2.$$

It follows that a solution x with $L(\lambda, x) - opt(\lambda) \leq \eta - 1$ can be computed in $\mathcal{O}\left(\frac{lip(\lambda) u_{\max}^2}{\eta - 1}\right)$ iterations.

Now let lb_{opt} be a lower bound on $opt(\lambda)$. We run the conditional gradient method up to accuracy $(\eta - 1) \cdot lb_{opt}$ such that $L(\lambda, x) \leq opt(\lambda) + (\eta - 1) \cdot lb_{opt} \leq \eta \cdot opt(\lambda)$ as desired.

It remains to find a good lower bound lb_{opt} to prove the desired total running time. In particular, we are interested in a running time that is independent of the weights λ (and hence independent of ω). We can bound

$$lb_{opt} \geq \max\left\{\lambda_{power} \cdot power(l), \max_{e \in E} \lambda_e \cdot d_{\min}\right\}.$$

This implies a bound for $lip(\lambda)/lb_{opt}$:

$$\begin{aligned} lip(\lambda)/lb_{opt} &= \frac{\lambda_{power} K_P + \max_{e\in E} \lambda_e K_D}{lb_{opt}} \\ &\leq \frac{\lambda_{power} K_P}{\lambda_{power} power(l)} + \frac{\max_{e\in E} \lambda_e K_D}{\max_{e\in E} \lambda_e d_{\min}} \\ &= \frac{K_P}{power(l)} + \frac{K_D}{d_{\min}} \leq 2\hat{K}. \end{aligned}$$

Thus, It takes $\mathcal{O}\left(\frac{lip(\lambda) u_{\max}^2}{(\eta-1)\cdot lb_{opt}}\right) = \mathcal{O}\left(\hat{K}\frac{u_{\max}^2}{\eta-1}\right)$ iterations and $\mathcal{O}\left(T_{grad}\hat{K}\frac{u_{\max}^2}{\eta-1}\right)$ time to achieve an η-approximate solution. □

F. Running Time Analysis

The running time depends on the problem width ρ, which is defined as the maximum ratio by which a single customer can overuse a resource in any solution compared to the optimum:

$$\rho := \max\left\{1, \sup\left\{\frac{(\text{usg}_c(x_c))_r}{\lambda^*} : r \in \mathscr{R}, c \in \mathscr{C}, x_c \in \mathscr{B}_c\right\}\right\}. \tag{14}$$

We will use the following lemma from [26].

Lemma 4 (Müller, Radke, and Vygen [26], Lemma 7)**.** *Let $0 < \delta, \delta' < 1$. Given an instance of the min-max resource sharing problem with $\lambda^* \leq 1$, we can compute a $\left(\eta(1+\delta) + \frac{\delta'}{\lambda^*}\right)$-approximate solution using $\mathcal{O}(\rho(\delta\delta')^{-1}\eta \log|\mathscr{R}|)$ calls to an η-approximate oracle function.*

Proof. Algorithm 2 is the resource sharing algorithm from [26] with one minor modification. The original algorithm in [26] has a mechanism to repeat oracle calls for a single customer if the usage of a resource exceeds one. This mechanism is important if many customers are present. In our case, we have only a single customer and can simplify the algorithm by setting the iteration count I to the worst case number of iterations from [26] right away, i.e. choosing

$$I = \mathcal{O}(\rho(\delta\delta')^{-1}\eta \log|\mathscr{R}|).$$

and $\gamma = \frac{\delta}{3\eta}$ as in [26]. □

As $|\mathscr{R}| = |\mathscr{P}| + 1$, we obtain the following guarantee for the continuous relaxation of the gate sizing problem:

Lemma 5. *Assuming A1–A6, given a power budget $B \in \mathbb{R}_{\geq 0}$, and $0 < \epsilon < 1$, we can decide whether $\lambda^* \leq (1+\epsilon)$ or $\lambda^* > 1$ using Algorithm 2 in time $\mathcal{O}\left(u_{\max}^2\hat{K}\cdot T_{grad}\cdot\rho\cdot\hat{U}\cdot\epsilon^{-3}\log|\mathscr{P}|\right)$.*

Proof. By A3, we have $\lambda^* \geq power(l)/power(u) \geq \hat{U}^{-1}$. We apply Lemma 4 with $\delta = \epsilon/4, \eta = 1+\delta$ and $\delta' = \delta/\hat{U}$. If $\lambda^* \leq 1$, we obtain a solution with maximum resource usage at most $\left((1+\epsilon/4)^2 + \epsilon/4\right) \leq (1+\epsilon)$. Otherwise, if the maximum resource usage of the solution is greater than $(1+\epsilon)$, we can conclude $\lambda^* > 1$. By our choice of η, δ, and δ', Lemma 4, and Theorem 3, the running time is $\mathcal{O}\left(\frac{u_{\max}^2\hat{K}\cdot T_{grad}}{\epsilon}\cdot\rho\cdot\frac{\hat{U}}{\epsilon^2}\log|\mathscr{P}|\right)$. □

Finally, we apply binary search on B to minimize the total power.

Theorem 6. *Assuming A1–A6, $\epsilon > 0$, and that the gate sizing problem (2) has a feasible solution, we can compute a gate sizing solution that minimizes the optimum power up to a factor of $(1+\epsilon)$ and violates any delay constraint by at most a factor of $(1+\epsilon)$ in time $\mathcal{O}\left(u_{\max}^2\hat{K}\cdot T_{grad}\cdot\rho\cdot\hat{U}\cdot\epsilon^{-3}\log|\mathscr{P}|\log\frac{\log\hat{U}}{\epsilon}\right)$.*

Proof. Let $\epsilon' = \epsilon/4$. We perform binary search among the budgets $B_i := power(l)\cdot(1+\epsilon')^i$ for $i = 0,\ldots,i_{\max}$, where $i_{\max} = \min\{i \in \mathbb{N}_0 : power(l)\cdot(1+\epsilon')^i \geq power(u)\} = \mathcal{O}(\frac{\log\hat{U}}{\epsilon'})$. For every tested budget B_i, we call Algorithm 2 with accuracy ϵ' to decide whether B_i is feasible up to a factor $(1+\epsilon')$, i.e. the maximum resource usage and λ^* do not exceed $(1+\epsilon')$ or infeasible, i.e. the maximum resource usage exceeds $(1+\epsilon')$ and, thus, $\lambda^* > 1$.

As the instance is feasible the binary search will identify a locally smallest index $i_0 \in \mathbb{N}_0$ such that we find a solution with maximum usage $1+\epsilon'$ for the instance with budget B_{i_0}.

Let $B^\star$ be the optimum budget. For all $B_i \geq B^\star$ by Lemma 5, Algorithm 2 returns a solution in which the maximum timing violation is $1+\epsilon' \leq 1+\epsilon$. Therefore, $B_{i_0} \leq (1+\epsilon')\cdot B^\star$ and the total power consumption is bounded by $(1+\epsilon/4)B_{i_0} \leq (1+\epsilon/4)^2 B^\star \leq (1+\epsilon)B^\star$. The binary search examines $\mathcal{O}(\log\frac{\log\hat{U}}{\epsilon})$ budgets, and by Lemma 5 the total running time is $\mathcal{O}\left(u_{\max}^2\hat{K}\cdot T_{grad}\cdot\rho\cdot\hat{U}\cdot\epsilon^{-3}\log|\mathscr{P}|\log\frac{\log\hat{U}}{\epsilon}\right)$. □

Similarly, we could perform a search on T to minimize the feasible cycle time, or two searches for minimizing the cycle time and then also the power.

If $\rho\hat{U} \geq |E|$ holds, we can use a different running time analysis by [11] and obtain the following alternative result for Lemma 5.

Lemma 7. *Assuming A1–A6 and given a power budget $budget_{power} \in \mathbb{R} \geq 0$, $0 < \epsilon < 1$, we can decide whether $\lambda^* \leq (1+\epsilon)$ or $\lambda^* > 1$ using Algorithm 2 in time $\mathcal{O}\left(u_{\max}^2\hat{K}\cdot T_{grad}\cdot|E|\epsilon^{-3}\log|\mathscr{P}|\right)$.*

Combined with binary search, this yields a total running time of

$$\mathcal{O}\left(u_{\max}^2\hat{K}\cdot T_{grad}\cdot|E|\epsilon^{-3}\log|\mathscr{P}|\log\frac{\log\hat{U}}{\epsilon}\right).$$

The cardinality $|\mathscr{P}|$ appears only logarithmically, thus as a rough estimate we can derive a linear bound

$$\log|\mathscr{P}| \leq \log 2^{|V|} = \mathcal{O}(|V|).$$

Moreover, under the assumptions of Lemma 5, the running time is provably polynomial. Filtering out the technology-dependent constants $u_{\max}, \hat{K}, \hat{U}$ and ρ the running time in Theorem 6 is essentially

$$\mathcal{O}\left(\frac{T_{grad}}{\epsilon^3}|V|\log\frac{1}{\epsilon}\right).$$

Our algorithm gives some justification to similar heuristic modifications of updating Lagrange multipliers in [42], [24]. It was also shown in [38] that the basic multiplicative weights algorithm described in Arora et al. [1] can be applied here. In contrast to our algorithm, it also decreases weights and in that sense is more similar to the aforementioned heuristics. However, it comes with an inferior running time bound.

G. Characteristics of the RC-Delay Model

In this section we show that for the RC-delay model, A5 holds and the problem width ρ is a small technology-dependent constant. Let $\Gamma^+(g) \subset \mathcal{G}$ denote the set of gates that are direct successors of g. In the RC-delay model, the delay of an edge $e \in E$ leaving a gate $g \in \mathcal{G}$ is of the form

$$\kappa_e + \frac{\alpha_e}{x_g} + \sum_{g' \in \Gamma^+(g)} \beta_e x_{g'} + \sum_{g' \in \Gamma^+(g)} \frac{\gamma_e}{x_g} x_{g'}, \tag{15}$$

where $\kappa_e, \alpha_e, \beta_e$, and $\gamma_e > 0$ are constants [8], [6]. After variable transformation into a convex program as described in Section III, we get

$$delay_e(x) = \kappa_e + \alpha_e e^{-x_g} + \sum_{g' \in \Gamma^+(g)} \beta_e e^{x_{g'}} + \sum_{g' \in \Gamma^+(g)} \gamma_e e^{x_g - x'_g}. \tag{16}$$

With these delay formulas we obtain the Lipschitz-continuity.

Lemma 8. *In the RC-delay model, where delays are given as in (16), the gradient $\nabla \sum_{e \in E} delay_e(x)$ is Lipschitz continuous on the set X with Lipschitz constant K_D bounded by*

$$K_D \leq \max_{x \in X} \max_{g \in \mathcal{G}} \left(\sum_{e \in E_g} delay_e(x) \right),$$

where $E_g \subseteq E$ is the set of edges incident to gate g, i.e. edges whose delay is affected by the size of g.

Proof. $\sum_{e \in E} delay_e(x)$ is a sum of exponential functions of the form

$$\sum_{g \in \mathcal{G}} \zeta_g e^{x_g} + \sum_{g \in \mathcal{G}} \chi_g e^{-x_g} + \sum_{g, g' \in \mathcal{G}} \psi_{g,g'} e^{x_g - x_{g'}},$$

with $\zeta_i, \chi_i, \psi_{ij} \in \mathbb{R}_{\geq 0}$ for all $1 \leq i, j \leq n$. Thus, each partial derivative $\frac{\partial \sum_{e \in E} delay_e(x)}{\partial x_g}$ is the sum of exponential functions of $\pm x_g$ ($g \in \mathcal{G}$).

By the definition of the Lipschitz constant we have:

$$\begin{aligned} lip(\lambda) &= \max_{x,y \in X} \frac{||\nabla L(\lambda, x) - \nabla L(\lambda, y)||_\infty}{||x - y||_\infty}, \\ &\leq \max_{x,y \in X} \max_{g \in \mathcal{G}} \frac{\frac{\partial L}{\partial x_g}(\lambda, x) - \frac{\partial L}{\partial y_g}(\lambda, y)}{x_g - y_g} \\ &\leq \max_{z \in X} \max_{g \in \mathcal{G}} \frac{\partial^2 L}{\partial z_g^2}(\lambda, z) \\ &= \max_{z \in X} \max_{g \in \mathcal{G}} \sum_{e \in E_g} delay_e(z), \end{aligned}$$

where the second inequality follows from the mean value theorem, and for the last equality we use the fact that $L(\lambda, z)$ is the sum of exponential functions. □

We can conclude that on a reasonably buffered netlist, where $|E_g|$ and the maximum delay of any edge in the timing graph are bounded, K_D can also be considered a constant.

Theorem 9. *In the RC-delay model, where delays are given as in (16), and when there are constants κ, τ such that $\max_{e \in E, x \in X} delay_e(x) < \kappa$ and $\max_{g \in \mathcal{G}} |E_g| \leq \tau$, then*

$$K_D \leq \kappa \cdot \tau.$$

Furthermore, we show the following bound on the problem width.

Lemma 10. *In the RC-delay model the problem width ρ can be bounded by $\rho \leq \max\{\hat{U}, u_{\max}^2\}$.*

Proof. Let $x \in X$ be any gate sizing solution, and x^* a solution attaining λ^*. By A3 we have

$$power(x) \leq \hat{U} \cdot power(x^*) \leq \hat{U} \lambda^*.$$

The RC-delay (15) can be bounded as follows:

$$\begin{aligned} delay_e(x) &= \kappa_e + \frac{\alpha_e}{x_g} + \sum_{g' \in \Gamma^+(g)} \beta_e x_{g'} + \sum_{g' \in \Gamma^+(g)} \frac{\gamma_e}{x_g} x_{g'} \\ &= \kappa_e + \frac{\alpha_e}{x_g} \frac{u_g}{u_g} + \sum_{g' \in \Gamma^+(g)} \beta_e x_{g'} \frac{l_{g'}}{l_{g'}} \\ &\quad + \sum_{g' \in \Gamma^+(g)} \frac{\gamma_e}{x_g} x_{g'} \frac{l_{g'}}{l_{g'}} \frac{u_g}{u_g} \\ &\leq \kappa_e + u_{\max} \frac{\alpha_e}{u_g} + u_{\max} \sum_{g' \in \Gamma^+(g)} \beta_e l_{g'} \\ &\quad + u_{\max}^2 \sum_{g' \in \Gamma^+(g)} \frac{\gamma_e}{u_g} l_{g'} \\ &\leq u_{\max}^2 \cdot delay_e(x^*). \end{aligned} \tag{17}$$

Thus, for any path $P \in \mathscr{P}$ we have $delay_P(x)/T \leq u_{\max}^2 \lambda^*$. □

H. Modeling Timing Constraints with Edge Resources

The timing constraints in (1) can also be modeled by explicit edge resources as proposed in the context of timing-driven global routing by Held et al. [15].

Combining a binary search for the power budget with the resource sharing algorithm from [26] we obtain the following runtime bound:

Theorem 11. *Assuming A1–A6 the continuous relaxation of the gate sizing problem can be approximated up to accuracy $(1 + \epsilon)$ for $\epsilon > 0$ in time*

$$\mathcal{O}\left(u_{\max}^2 \hat{K} \cdot \frac{T_{grad}}{\epsilon} |E| \log |E| \epsilon^{-2} \log \frac{\log \hat{U}}{\epsilon} \right).$$

For further details we refer to [38]. However, this model requires additional arrival time customers for each $v \in V$, which would make the practical implementation more involved.

Furthermore, our model with path resources rather targets the feasibility for all timing paths simultaneously. Thus, we believe that the model we apply here is more suitable in practice for maximizing the TPNS (4) and TTNS [34], when no feasible solution exists.

I. Model Extensions

We now discuss extensions of the resource sharing model. It is easy to see that the oracle described in the proof of Theorem

3 for the gate customer can still be utilized after adding the following resources.

1) Load capacitance limits: Under certain assumptions load capacitance limits can be modeled as resources and integrated into the framework. We introduce a resource for each net source pin p on the chip to model constraints on load capacitances:

$$loadcap_p(x) \le loadlim_p(x),\ x \in X, \tag{18}$$

where $loadcap_p(x)$ is the load capacitance seen at pin p and $loadlim_p(x)$ is the load capacitance limit allowed at p with sizes x. If $loadcap_p(x)$ is convex and $loadlim_p(x)$ is concave, we can define a resource for p and a usage function as follows: Let $C_p := \max\{loadlim_p(x) : x \in X\} > 0$, and let the resource usage function be

$$\text{usg}_{\mathcal{G},p}(x) := \frac{loadcap_p(x) + C_p - loadlim_p(x)}{C_p}.$$

This is a convex function in x as required in the resource sharing problem (cf. Section III-A) and (18) is fulfilled if and only if $\text{usg}_{\mathcal{G},p}(x) \le 1$. This model works in particular if $loadcap_p(x)$ and $loadlim_p(x)$ are linear functions.

2) Slews & slew limits: A more accurate delay model with slew propagation was already presented in [3]. Slews and delays are still posynomials and convexifiable. Thus, such more accurate delay models can also be integrated into our resource sharing model. They allow us also to respect slew limits at each gate input pin. We add a resource for each sink pin of each net and define the resource usage as the slew arriving at the corresponding pin divided by its slew limit. Assuming a constant slew limit, which is prevalent in practice, it is easy to see that as in Section III-I1 the resulting resource usage is a convex function in x (cf. Section II).

With these model extensions, electrical violations are not treated as hard constraints, but are punished by increasing resource weights. In a physical design flow however, they are often treated as hard constraints, in particular towards the end of the design flow. This can be accomplished by adding them as constraints in the oracle function.

3) Placement Density Constraints: Placement-unaware gate sizing can overfill certain regions of a chip due to an increase in the area. Resolving overlaps might be hard and causes disruptions in such regions. Therefore, it is beneficial if a global gate sizing algorithm is aware of placement density. In previous works on gate sizing, placement density constraints are usually taken into account implicitly by incorporating area consumption or a violation of the maximum allowed area consumption into the objective function. See for example Cong et al. [7] and Reimann et al. [34]. Density constraints can be easily integrated into the resource sharing framework. We subdivide the chip into regions and add a resource for each placement region R. The resource usage of the gate customer for $x \in X$ then is

$$\text{usg}_{\mathcal{G},R} := \frac{1}{area(R)\cdot target}\cdot\sum_{g\in\mathcal{G}} area_R(x_g), \tag{19}$$

where $area(R)$ is the free area in region R, $0 < target < 1$ is the density target of region R and $area_R(x_g)$ is the area of gate g in region R with size x_g.

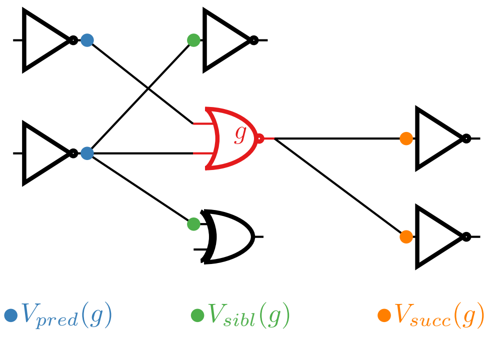

Figure 2. A gate g and the timing subgraph in its region. The pins $p \in \text{Nb}(g) = V_{pred}(g) \cup V_{sibl}(g) \cup V_{succ}(g)$ are considered in slack queries.

IV. Discrete Cell Selection

For the discrete cell selection problem no efficient algorithms are known and are unlikely to exist, because the choices of voltage thresholds are inherently discrete and delay functions in practice are often non-convex. We propose a new cell selection heuristic algorithm based on Algorithm 2, but with a heuristic sizing oracle.

Under the assumption that the number of gates per logic level is bounded by a constant, there is also an FPTAS for the oracle of the gate customer [38], similar to the FPTAS for maximum delay minimization by [23]. However, in practice the number of gates per logic level increases with the instance size, limiting its usefulness.

A. Discrete Cell Selection Oracle

We now describe a heuristic cell selection oracle that aims to minimize the weighted resource usage (11). Our algorithm traverses the gates $g \in \mathcal{G}$ one by one and selects a cell type $x_g \in X_g$ for g while keeping gates in $\mathcal{G} - g$ fixed.

To speed-up computations we restrict delay updates to a small region around g, which is defined by the timing subgraph induced by the gates that are connected with g by a net, i.e. we recompute delays, slews, arrival times, and required arrival times for all timing graph edges inside and between these gates, keeping the boundary constant. Thereby, slew effects in the upstream cone of the region are ignored during the cell refinement.

Figure 2 shows a gate $g \in \mathcal{G}$ and its region. $V_{sibl}(g)$ denotes the set of sink pins in input nets of g excluding the pins of g (siblings), $V_{succ}(g)$ denotes the set of sink pins in output nets of g (successors), and $V_{pred}(g)$ denotes the set of source pins in input nets of g (predecessors). We say that

$$Nb(g) := V_{pred}(g) \cup V_{sibl}(g) \cup V_{succ}(g)$$

is the neighborhood of g.

We found experimentally that the algorithm gives good solution in fewer iterations with two heuristic modifications.

The edge delay changes induced by changing the cell-type of a gate can be estimated by measuring slack changes in a neighborhood of g. We have observed experimentally that

Algorithm 3 Local search oracle with pin weights

```
Input: An initial cell selection $x \in X$, pin weights $(\omega_v)_{v \in V}$,
    a power weight $\omega_{power}$.
Output: A new cell selection $x'_g \in X$.
1: $x' \leftarrow x$
2: for $g \in \mathcal{G}$ do
3:     Update slacks for all $v \in Nb(g)$ with global slew and
       delay propagation.
4:     Choose $x'_g$ s.t. (21) is minimized with slew and delay
       propagation restricted to the region around $g$.
5: end for
```

best results are obtained when slacks are optimized directly and edge delay changes are only captured implicitly. As this approach emphasizes slack changes the timing metrics are optimized more efficiently. To this end, we define for all $v \in V$ the *pin weight*

$$\omega_v := \omega_{\mathscr{P}_{[V_{\text{in}},v]}} \cdot \omega_{\mathscr{P}_{[v,V_{\text{out}}]}} = \max\left\{ \sum_{e \in \delta^+(v)} \omega_e, \sum_{e \in \delta^-(v)} \omega_e \right\}, \tag{20}$$

where $\omega_{\mathscr{P}_{[V_{\text{in}},v]}} \cdot \omega_{\mathscr{P}_{[v,V_{\text{out}}]}}$ is defined in the proof of Lemma 2. Recall that the two sums in the maximum differ only for vertices in $V_{\text{in}} \cup V_{\text{out}}$. We use the following heuristic objective when choosing a cell type for g:

$$\omega_{power} \frac{power(x'_g)}{B} - \sum_{v \in Nb(g)} \omega_v \frac{\text{slack}_{x'}(v)}{T}. \tag{21}$$

In addition, we do not select cell types that increase load or slew violations compared to the current solution. Note that this does not necessarily capture all delay changes, but has the advantage that it optimizes timing metrics more directly.

Second, we scale γ by the iteration and choose

$$\omega_E \leftarrow \text{EDGEWEIGHTS}(\gamma/i, y_E)$$

in line 5 of Algorithm 2. By scaling γ, the total weights stay more stable from iteration to iteration. Algorithm 3 shows the overall discrete oracle. While most local search algorithms for cell selection process the gates either in topological or reverse topological order, we found that the quality hardly depends on the order and an arbitrary order gives similar resutls. We will use this observation in our parallelization approach in the next section.

B. Parallelization

For gate sizing by Lagrangian relaxation an efficient multi-threaded approach was described in [40], parallelizing the Lagrangian subproblem as well as static timing analysis and multiplier update. To avoid conflicts, they propose to not 1) size gates with common input nets and 2) gates on a common path simultaneously.

We did not parallelize the update of the resource weights yet, because it already is very fast. For static timing analysis we use the parallelization build into the timing engine (IBM EinsTimer). Finally, the most time-consuming step is the cell selection oracle. Here, we employ an approach provided by the IBM design environment [4]. It partitions the netlist into logical regions and allows lock-free optimization accross the regions. Input gates of the regions must not be sized. Thereby, two gates with common input net cannot be sized simultaneously, but we might size gates on a common path if they are assigned to different regions. As the input gates of each region are temporarily fixed, we see only marginal degradations after resolving the regions. Multiple partitions ensure that each gate is sizable in some partition.

C. Further implementation details

We omit the outer binary search over the power budget B but instead estimate an initial budget and adjust it incrementally during the algorithm (see Sections VI-A and VI-B).

As we look for a discrete solution we do not keep track of a convex combination of sizes but only keep the best solution that was computed, which is usually the result of the last iteration.

Industrial designs can have multiple clock domains and propagate rise and fall transitions. We compute a weight at each pin for each arrival time that is propagated by the timing engine. In the weight computation T is chosen as the cycle time of the given clock domain.

V. Comparison to the Projected Subgradient Method

We also implemented the projected subgradient method (see [3] for details) to allow comparison with our new resource sharing algorithm. It is built around the same oracle (Algorithm 3) which enables us to directly compare the different weight update schemes. In this particular comparison, the oracle locally minimizes the original objective from (11) instead of (21) for both the subgradient method and the resource sharing algorithm as required by theory.

To get a fair comparison between the two methods, we omitted heuristic modifications of the subgradient method except for those necessary to implement a discrete cell selection oracle.

For the subgradient method, it is difficult to determine initial multipliers and step lengths that work well for a broad range of instances, despite many improvements [41], [44]. We simply initialize each multiplier $(\lambda_e)_{e \in E(G)}$ with a small percentage of the absolute negative slack of e. The Lagrange multipliers are updated by the local edge slack as in [3] and projected to the non-negative flow space with the heuristic from [41]. We also conducted experiments in which we projected the multipliers exactly solving the arising quadratic minimum cost flow problems. However, we observed that the results and convergence behavior did not improve significantly [38].

We also implemented a variant of the subrgradient method suggested by Jiang et al. [19], which solves the feasibility problem (8) and maintains a power multiplier λ_P. This has the advantage that on infeasible designs, where edge delay

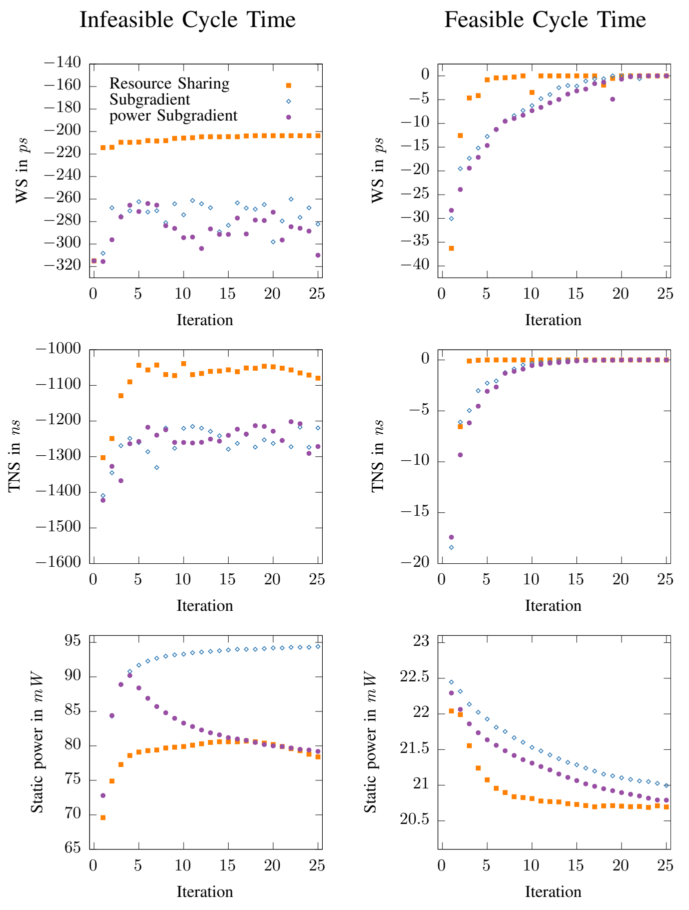

Figure 3. Convergence of resource sharing algorithm vs. subgradient method.

multipliers grow to infinity, the objective is not dominated by the delay multipliers.

We tested several step size rules and found best results when using the step size $1/\sqrt{i}$ in the i-th subgradient iteration, which guarantees convergence.

On the left side of Figure 3, we compare our new resource sharing algorithm (squares) with the regular projected subgradient method [3] (diamonds) and with the power-constrained subgradient method [19] (circles) on an industrial instance with infeasible timing constraints. On the right side, we compare the same methods on the same instance but with a relaxed feasible cycle time. The figures show the development of the worst slack WS, the total negative slack TNS and static power consumption after each of 25 iterations.

On the infeasible instance, both subgradient methods show bouncing WS and TNS. For the regular variant, the power consumption increases as multipliers go to infinity. The power-constrained subgradient method lets the power consumption decrease from iteration 5 on, as the power multiplier grows, too. The resource sharing algorithm shows a stable convergence in all metrics, yielding significantly better WS and TNS. On the feasible instance, all methods converge to feasible solutions. Again, the resource sharing algorithm exhibits a much faster convergence and a better power consumption after 25 iterations.

VI. Experimental Results

We evaluated our discrete cell selection algorithm on industrial 22 nm instances from IBM, which were also used in [33], [34], and on the ISPD 2013 cell selection benchmarks [28]. The signoff timing engine IBM EinsTimer is used for all timing and power calculations using the respective delay calculation methods from [34] and [28].

A. *Results on industrial instances*

The experiments on industrial 22nm instances were conducted on a heterogeneous cluster with Intel Xeon CPUs with clock frequencies between 2.6 and 3.5 GHz. For each instance, all experiments were carried out on the same server. We ran our algorithm with six threads. We compared our approach (*RS*) with the Lagrangian relaxation (*LR*) algorithm by [34], which runs sequentially and which was integrated into the IBM design environment by the authors of [34].

We ran physical design with the same instances as [34]. As [34], we used the result of the current IBM design flow just before detailed routing as input to our cell selection experiments. Thus, the input to cell selection differs slightly to the input in [34], which, in turn, differs to the one in [33], all caused by minor changes of the flow. However, within our new experiments the input to our *RS* algorithm and to the *LR* algorithm coincide. As a fixed power budget B we chose 0.8 times the initial power consumption achieved by the design flow.

Table I shows the results. The instance names and their sizes are given in the first two columns. For each instance, we computed upper bounds ρ^t and ρ^p for the maximum usage of a timing or power resource by any feasible solution, respectively. To this end, we compute a minimum power solution $\bar{l}$ satisfying all capacity constraints but ignoring delay constraints. For ρ^t, we then assert at each sink pin of a net the highest possible pin capacity to get upper bounds for the delays. Then ρ^t is the quotient of the maximum path delay and T. Furthermore, $\rho^p = power(\bar{u})/power(\bar{l})$, where $\bar{u}$ is a solution using lowest V_t levels and largest sizes everywhere. It follows that $\rho \leq \max\{\rho^t, \rho^p\}$.

For every instance three solutions are analyzed. The solution that is given by the industrial flow is named *Industrial*, the solution computed by the Lagrangian relaxation algorithm of [34] is named *LR* and the solution computed by our resource sharing algorithm is called *RS*. For each of those runs we measured the worst slack (WS), the total negative slack (TNS) and the true total negative slack (TTNS) which were introduced in Section II. The electrical violations are shown in terms of the number of slew violations (v_{slew}) and load violations (v_{load}). For the power consumption we distinguish between the static power usage P_{static} and the total power usage $P_{\text{total}} = P_{\text{static}} + P_{\text{dynamic}}$. The testbed contains a high variety in terms of the leakage/dynamic power ratio. ΔP_{total} denotes the change in total power. Running times t_{wall} are given in the last column.

The timing metrics are measured just before detailed routing. All numbers refer to results after subsequent placement

Instance	$\lvert\mathcal{G}\rvert$	ρ^t	ρ^p	Flow	WS $[ps]$	TNS $[ns]$	TTNS $[ns]$	v_{slew}	v_{load}	P_{static} $[\mu W]$	P_{total} $[\mu W]$	ΔP_{total}	t_{wall} $[h{:}m{:}s]$
ibm2016uP_01	99k	16	10	Industrial	-69.5	-101.4	-602.6	11	0	81.7	95.1		
				LR [34]	-69.6	-94.7	-583.4	5	1	65.7	79.0	-16.9%	11:25:03
				RS	-69.4	-103.2	-582.1	3	0	65.7	79.0	-16.9%	57:16
ibm2016uP_02	10k	11	14	Industrial	-156.9	-1.9	-10.0	0	5	1.2	2.5		
				LR [34]	-156.9	-1.9	-10.0	0	3	1.2	2.4	-2.1%	1:47:53
				RS	-156.7	-1.9	-10.2	0	0	1.2	2.4	-1.4%	24:06
ibm2016uP_03	9k	8	7	Industrial	7.0	-0.0	-0.0	0	2	2.7	52.5		
				LR [34]	7.0	-0.0	-0.0	0	2	2.7	52.4	-0.1%	1:19:29
				RS	7.0	-0.0	-0.0	0	2	2.7	52.7	+0.5%	22:13
ibm2016uP_04	7k	14	9	Industrial	-11.2	-0.7	-0.7	0	0	1.6	2.9		
				LR [34]	-11.2	-0.7	-0.7	0	0	1.6	2.9	+0.7%	58:32
				RS	-11.1	-0.7	-0.7	0	0	1.6	2.9	-0.5%	15:22
ibm2016uP_05	16k	9	4	Industrial	-76.6	-36.6	-64.0	91	2	20.3	67.4		
				LR [34]	-76.5	-37.1	-64.5	40	2	18.0	64.8	-3.8%	53:13
				RS	-76.3	-36.6	-64.0	42	1	16.7	63.2	-6.3%	20:19
ibm2016uP_06	77k	13	6	Industrial	-108.9	-15.9	-25.6	20	381	35.7	147.6		
				LR [34]	-108.9	-14.6	-24.5	14	381	33.5	145.3	-1.5%	3:13:37
				RS	-108.9	-13.7	-21.0	10	381	33.9	145.8	-1.3%	38:13
ibm2016uP_07	72k	14	9	Industrial	-33.9	-38.6	-231.6	9	4	60.8	73.2		
				LR [34]	-33.9	-38.7	-235.0	2	2	53.2	65.6	-10.4%	7:55:57
				RS	-33.3	-38.2	-221.3	3	5	53.2	65.6	-10.3%	47:45
ibm2016uP_08	18k	11	4	Industrial	-72.6	-35.1	-176.4	64	4	16.8	85.6		
				LR [34]	-72.6	-35.0	-176.3	40	4	16.7	85.4	-0.3%	2:20:00
				RS	-72.6	-35.4	-175.3	40	2	12.0	79.7	-6.9%	17:19
ibm2016uP_09	18k	11	6	Industrial	-23.2	-8.8	-36.2	3	1	14.5	47.6		
				LR [34]	-22.8	-8.7	-37.0	1	0	12.3	45.2	-5.0%	2:06:49
				RS	-22.6	-9.3	-36.4	1	1	12.4	45.4	-4.7%	17:47
ibm2016uP_10	126k	14	6	Industrial	-43.8	-76.0	-342.6	67	7	91.6	397.1		
				LR [34]	-41.0	-84.2	-401.8	48	7	74.7	371.5	-6.5%	9:05:31
				RS	-38.9	-78.8	-346.9	31	2	65.1	365.3	-8.0%	1:22:58
ibm2016uP_11	25k	16	5	Industrial	-140.7	-167.2	-886.7	19	27	39.7	61.6		
				LR [34]	-140.4	-164.1	-881.8	20	28	36.7	58.7	-4.7%	2:27:04
				RS	-140.4	-164.2	-842.2	20	21	34.5	56.5	-8.4%	19:30
ibm2016uP_12	18k	16	4	Industrial	-417.8	-342.0	-696.1	12	3	5.1	25.4		
				LR [34]	-417.8	-333.7	-680.6	12	3	4.8	25.0	-1.8%	2:46:08
				RS	-417.7	-333.6	-675.6	12	0	5.3	25.5	+0.2%	18:36
ibm2016uP_13	20k	11	4	Industrial	-47.6	-20.8	-103.4	1	2	19.6	80.2		
				LR [34]	-47.4	-20.3	-103.0	0	2	18.2	78.6	-2.0%	1:21:09
				RS	-47.3	-20.1	-101.0	0	2	15.4	75.8	-5.5%	21:35
ibm2016uP_14	13k	7	2	Industrial	-54.8	-5.1	-9.2	1	4	8.2	17.9		
				LR [34]	-54.8	-5.1	-9.2	1	4	8.2	17.9	-0.1%	39:34
				RS	-54.3	-5.1	-9.2	1	4	8.2	17.9	+0.0%	13:04

Table I
Results on IBM 22 nm server instances. The RS flow uses 6 threads.

legalization. The power reductions and running times that we measured for *LR* are comparable to those in [33] and [34].

On these instances the primary purpose of cell selection is to reduce the power consumption while maintaining the timing metrics. Instead of optimizing the power budget B via binary search, we initialize B as 80% of the initial power consumption. Then, after each iteration of Algorithm 2, we increment (decrement) B by the factor 1.15 (1.15^{-1}) if the TTNS decreased (increased) by more than 5% compared to the previous iteration.

In every iteration of the algorithm we invoke the local search oracle exactly once. In global routing one usually chooses γ depending on the amount of iterations. [26] obtain good solutions with $\gamma = \frac{125}{\#\text{iterations}}$. As we only perform four iterations we use a large value of $\gamma = 80$.

Throughout the testbed our algorithm obtains a comparable or a better total power reduction. On instance ibm2016uP_08 we are able to reduce the power by 6.9% while the reference algorithm only reduces the total power by 0.3%. The running time is greatly reduced. On the largest instance ibm2016uP_10 with 126k gates the resource sharing approach takes about 83 minutes. The reference algorithm does not only take more than 6 times the running time but also greatly worsens the TNS and TTNS while it obtains a worse power reduction.

Benchmark	$\lvert\mathcal{G}\rvert$	ρ^t	ρ^p	P_{static}				t_{wall}		
				LR$^{\infty}$[9] $[W]$	*LR*[9] $[W]$	*RS* $[W]$	Change *LR/RS*	*LR*[9] $[m]$	*RS* $[m]$	Change *LR/RS*
usb_phy_slow	510	71	1658	0.00107	0.00107	0.00109	+1.9%	0.49	1.12	+128.6%
usb_phy_fast	510	106	1658	0.00153	0.00155	0.00164	+5.8%	0.42	1.13	+169.0%
pci_bridge32_slow	28k	87	1546	0.05695	0.05696	0.05911	+3.8%	10.53	9.33	-11.4%
pci_bridge32_fast	28k	116	1546	0.08504	0.08544	0.08993	+5.3%	22.62	13.90	-38.5%
fft_slow	31k	311	1751	0.08655	0.08660	0.08928	+3.1%	25.71	15.85	-38.4%
fft_fast	31k	399	1751	0.19391	0.19431	0.20851	+7.3%	40.43	30.22	-25.3%
cordic_slow	42k	221	1632	0.26567	0.27051	0.26917	-0.5%	69.04	31.73	-47.2%
cordic_fast	42k	252	1632	0.98018	1.00099	0.90893	-9.2%	117.08	35.70	-69.5%
des_perf_slow	104k	65	1495	0.32729	0.33042	0.33535	+1.5%	132.27	33.88	-74.4%
des_perf_fast	104k	74	1495	0.64450	0.64882	0.61616	-5.0%	347.87	51.90	-85.1%
edit_dist_slow	121k	193	1124	0.41604	0.42549	0.45703	+7.4%	129.90	38.63	-70.3%
edit_dist_fast	121k	231	1124	0.53547	0.53979	0.57169	+5.9%	352.96	67.45	-80.9%
matrix_mult_slow	153k	77	1533	0.44291	0.44427	0.44911	+1.1%	226.13	87.95	-61.1%
matrix_mult_fast	153k	98	1533	1.54157	1.61093	1.42813	-11.3%	395.96	101.07	-74.5%
netcard_slow	884k	38	547	5.15483	5.15524	5.20912	+1.0%	483.55	89.75	-81.4%
netcard_fast	884k	45	547	5.18159	5.20015	5.25621	+1.1%	400.89	90.52	-77.4%

Table II
RESULTS ON ISPD 2013 CONTEST BENCHMARKS. THE RS FLOW USES 44 THREADS.

B. Results on ISPD 2013 instances

In contrast to the industrial instances, the initial solutions of the ISPD 2013 instances are not pre-optimized. Most gates are set to their largest possible size and lowest possible voltage threshold leading to huge timing violations and an extensive power consumption.

As we have no reasonable estimate for the power budget we use a static usage of 1.05 for the power resource in each of the five resource sharing iterations. This leads to the power being optimized as soon as the timing is almost closed. Again, we use $\gamma = 80$. As the initial solution is meaningless, we perform three oracle calls with the same weights during the first two iterations, two in the next two iterations and a single one in the final iteration. V_t changes are only allowed starting from the third iteration, as otherwise high timing costs lead to many low V_t cells which would need to be deaccelerated in later iterations. After our Algorithm terminated, there may still be small timing violations. Thus, we employ a dedicated V_t post-optimization to achieve timing closure. It is a discrete variant of the algorithm for the linear time-cost tradeoff problem described by Phillips and Dessouky [30], which iteratively speeds up cells along minimum cuts in G weighted by the ratio of power increase/delay reduction. Details can be found in [12], Chapter 6.

If more than 0.1% of the cells are not on the highest V_t level we apply a post-processing by running a single iteration of Algorithm 2 and accept the new solution if the power improved. Final capacitance, slew, and delay violations are fixed by the simple local search from [13]. The full flow for the ISPD instances is sketched in Algorithm 4.

For the ISPD instances we could use a bigger server with two Intel Xeon E5-2699 v4 CPUs having 44 cores in total. We ran our algorithm with 44 threads.

Table II shows our results. We compare our solutions to the best known results which were published by Flach et al. [9].

Algorithm 4 Flow for ISPD 2013 benchmarks

```
1: Assign all gates to their smallest size and highest $V_t$.
2: Run Algorithm 2 with 5 iterations.
3: Fix-up by lowering $V_t$ along minimum cuts [30], [12].
4: if more than $0.1\%$ low $V_t$ gates then
5:     Post-processing: 1 iteration of Algorithm 2.
6: end if
7: Fix remaining violations by local search [13].
```

They made two experiments. In a first experiment, they run their cell selection without a running time limit (*LR*$^{\infty}$), while in the second experiment a reasonable stopping criterion is added (*LR*). The best known power consumptions are obtained by invoking their algorithm without any running time limit.

As both our algorithm and the reference algorithm find violation free solutions with feasible timing on all instances, we only measure the power consumption and the running time. Note that as we do not have access to PrimeTime, the official timing engine used in the benchmark. Instead, we used EinsTimer by IBM, setting precisely the same delay modes as documented with the benchmarks.

Again we specify upper bounds ρ^t and ρ^p for the maximum usage of a timing or power resource. The column labeled *LR*$^{\infty}$ reports the previously best known power consumptions. The power consumption and running times of the reference implementation are given in the columns labeled *LR*. In comparison our power consumption and running times are given in columns labeled *RS*, followed by the percentual change to *LR*.

In terms of solution quality the results are mixed but competitive. On six instances the difference is no more that 2%. On some instances like fft_fast and edit_dist_slow we use up to 7.4% more power than the reference algorithm. On other instances, e.g. des_perf_fast, cordic_fast and matrix_mult_fast, we obtain significant improvements by up to 11.3%.

We remark that the problem width ρ, especially in terms of the maximum usage of the power resource, seems to be much larger for the ISPD instances compared to the industrial instances. This shows that the instances might be harder to solve.

Taking advantage of parallelization (Section IV-B) our running times are highly competitive, on the largest instance netcard_fast with 884k gates we need 91 minutes compared to the 401 minutes of the reference algorithm. On other instances with at least 100k gates the speedup is similar. Note that we perform all timing computations exclusively with the signoff timer EinsTimer. The reference algorithm exploits the simple timing rules on the ISPD instances and implements a fast timing engine for the Lagrangian relaxation, while a signoff timer is only invoked to identify and fix remaining timing violations [9].

C. Parallelization results

We demonstrate the parallelization efficiency of our algorithm in Table III on two representative instances: a medium size and the largest ISPD benchmark.

We report running times for 3 steps. Running times for the sequential weight computation are reported in rows labeled “weights”, and for the first iteration of Algorithm 2, reflecting a sequential weight computation and a gate customer oracle call, in rows labeled “1st iteration”. In the first iteration the algorithm typically alters most cells and exhibits the highest parallelization speed-up over all iterations,. Finally, we show the total running time of Algorithm 4. We compare single-threaded running times with 4, 16, and 44 threads.

One can see that the weight computation takes less than 1% of the running time of the first iteration. Note that the CPUs have a clock speed of 2.2 GHz for 44 threads in contrast to 3.6 GHz for a single thread. Thus, the maximum theoretical speedup factor with 44 threads is roughly 26.9.

For the first iteration we are close to this theoretical bound with a speedup of 24.4 on des_perf_fast and 24.3 on netcard_fast. In later iterations the cell selection oracle performs less cell changes, thereby increasing the sequential overhead and reducing the speedup. Combined with the sequential local search to fix timing violations, the total speedup reduces to 4.9 and 16.2 respectively. In later iterations the weight computation takes up to 5% of the running time of an iteration.

A multi-threaded approach for cell selection based on Lagrangian relaxation was recently described in [40]. Their algorithm traverses gates in topological order and the maximum reported speedup with 16 threads is 8.13 on ISPD 2012 instances.

It is difficult to compare the parallelization efficiency. In [40] a custom static timer is used. On netcard_fast from the ISPD 2012 benchmark [40] is already faster using a single thread (42 minutes) than our algorithm using 44 threads (131 minutes) but an industrial sign-off timing engine.

VII. Conclusion

We showed how the convex and continuous gate sizing problem can be modeled as a resource sharing problem yielding a fast polynomial running time. Note that for the projected subgradient method, when applied to gate sizing, convergence but no polynomial running time was proven yet. Our comparison is also indicating a better practical convergence. We provide a theoretical foundation for combining Lagrangian relaxation with multiplicative weight update methods, which have been applied similarly but heuristically before.

For the discrete cell selection problem, the resource weights lead to a highly competitive heuristic algorithm. On industrial instances, we can improve the power consumption and on ISPD 2013 benchmarks we are competitive to the so far most successful algorithms [34], [9]. Furthermore, our efficient parallelization leads to significantly shorter running times even when using an industrial sign-off timer for all timing calculations.

References

[1] S. Arora, E. Hazan and S. Kale: “The Multiplicative Weights Update Method: A Meta-Algorithm and Applications”, Theory of Computing, vol. 8, pp. 121–164, 2012.

[2] S.B. Boyd, S.-J. Kim, D.D. Patil and M.A. Horowitz: “Digital Circuit Optimization via Geometric Programming”, Geometric Operations Research, vol. 53, no. 6, pp. 899–932, 2005.

[3] C.-P. Chen, C.C.N. Chu and D.F. Wong: “Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation”, IEEE TCAD 18 (7), 1999, 1014–1025.

[4] B. Chen, D.J. Hathaway, N.D. Hieter, K. Kalafala, J.S. Piaget, A.J. Suess: ”Managing Virtual Boundaries to Enable Lock-Free Concurrent Region Optimization of an Integrated Circuit”, US Patent US20160171147 A1, 2016.

[5] D. Chinnery and K. Keutzer: “Linear Programming for Sizing, V_{th} and V_{dd} Assignment”, ISLPED 2005, 149–154.

[6] C.C.N. Chu and D.F. Wong: “VLSI Circuit Performance Optimization by Geometric Programming”, Annals of OR 105 (1-4), 2001, 37–60.

[7] J. Cong, J. Lee and G. Luo: “A Unified Optimization Framework for Simultaneous Gate Sizing and Placement under Density Constraints”, ISCAS 2011, 1207–1210.

[8] J.P. Fishburn and A.E. Dunlop: “TILOS: A Posynomial Programming Approach to Transistor Sizing”, ICCAD 1985, 326–328.

[9] G. Flach, T. Reimann, G. Posser, M. Johann and R. Reis: “Effective Method for Simultaneous Gate Sizing and V_t Assignment Using Lagrangian Relaxation”, IEEE TCAD 33(4), 2014, 546–557.

[10] M. Frank and P. Wolfe: “An algorithm for quadratic programming”, Naval Research Logistics 3, 95–110, 1956.

[11] N. Hähnle: “Time-Cost Tradeoff and Steiner Tree Packing with Multiplicative Weights”, Technical report no. 1511115, Research Institute for Discrete Mathematics, University of Bonn, 2015.

[12] S. Held: “Timing Closure in Chip Design”, dissertation thesis, Research Institute for Discrete Mathematics, University of Bonn, 2008.

[13] S. Held: “Gate Sizing for Large Cell-Based Desings”, DATE 2009, 827–832.

[14] S. Held and J. Hu: “Gate Sizing”, Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology, CRC Press, 245–260, edited by L. Lavagno, I. L. Markov, G. Martin, L. K. Scheffer, 2016.

[15] S. Held, D. Müller, D. Rotter, V. Traub and J. Vygen: “Global Routing with Inherent Static Timing Constraints”, ICCAD 2015, 102–109.

[16] J. Hu, A.B. Kahng, S. Kang, M.-C. Kim and I.L. Markov: “Sensitivity-Guided Metaheuristics for Accurate Discrete Gate Sizing”, ICCAD 2012, 233–239.

[17] S. Hu, M. Ketkar and J. Hu:“Gate sizing for Cell-Library-Based Designs”, IEEE TCAD 28(6), 2009, 818–825.

[18] M. Jaggi: “Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization”, Intl. Conf. on Machine Learning 2013, 427–435.

[19] H.R. Jiang, J.Y. Jou, Y.W Chang:“Noise-Constrained Performance Optimization by Simultaneous Gate and Wire Sizing Based on Lagrangian Relaxation”, DAC 1999, 90–95.

Benchmark	Step	1 thread	Speedup	4 threads	Speedup	16 threads	Speedup	44 threads	Speedup
des_perf_fast	Weights	8s	1.0×	8s	1.0×	8s	1.0×	8s	1.0×
des_perf_fast	1st iteration	903s	1.0×	259s	3.5×	70s	12.9×	37s	24.4×
des_perf_fast	Total	256m	1.0×	93m	2.8×	64m	4.0×	52m	4.9×
netcard_fast	Weights	36s	1.0×	36s	1.0×	36s	1.0×	36s	1.0×
netcard_fast	1st iteration	170m	1.0×	45m	3.8×	12m	14.2×	7m	24.3×
netcard_fast	Total	1472m	1.0×	384m	3.8×	136m	10.8×	91m	16.2×

Table III
A LIST OF THE RUNNING TIMES FOR DES_PERF_FAST (104K GATES) AND NETCARD_FAST (884K GATES). THE MAXIMUM THEORETIC SPEEDUP IS 26.9 AS THERE ARE 44 THREADS WITH A THREADED PERFORMANCE OF 2.2 GHZ AND A SINGLE THREAD PERFORMANCE OF 3.6 GHZ.

[20] S. Joshi and S. Boyd: "An Efficient Method for Large-Scale Gate Sizing", IEEE TCAS I, 55(9), 2008, 2760–2773.

[21] A.B. Kahng, S. Kang, H. Lee, I.L. Markov and P. Thapar, "High-Performance Gate Sizing with a Signoff Timer" ICCAD 2013, 450–457.

[22] L. Li, P. Kang and Y. Lu and H. Zhou: "An Efficient Algorithm for Library-Based Cell-Type Selection in High-Performance Low-Power Designs", ICCAD 2012, 226–232.

[23] C. Liao and S. Hu: "Approximation Scheme for Restricted Discrete Gate Sizing Targeting Delay Minimization" Journal of Combinatorial Optimization 21(4), 2011, 497–510.

[24] V.S. Livramento, C. Guth, J.L. Guntzel and M.O. Johann: "A Hybrid Technique for Discrete Gate Sizing Based on Lagrangian Relaxation" ACM TDAES 19(4), 2014, 1–25.

[25] M. Mihalák, R. Šrámek and P. Widmayer: "Counting Approximately-Shortest Paths in Directed Acyclic Graphs", Approximation and Online Algorithms 8447, 2014, 156–167.

[26] D. Müller, K. Radke and J. Vygen: "Faster Min-Max Resource Sharing in Theory and Practice", Math. Progr. Computation 3(1), 2011, 1–35.

[27] D. Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Thompson and K. Keutzer: "Minimization of Dynamic and Static Power through Joint Assignment of Threshold Voltages and Sizing Optimization", ISLPED 2003, 158–163.

[28] M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke and Cheng Zhuo: "An Improved Benchmark Suite for the ISPD-2013 Discrete Cell Sizing Contest", ISPD 2013, 168–170.

[29] M. Ozdal, S. Burns and J. Hu: "Algorithms for Gate Sizing and Device Parameter Selection for High-Performance Designs", IEEE TCAD 31(10), 2012, 1558–1571.

[30] S. Phillips and M.I. Dessouky: "Solving the Project Time/Cost Tradeoff Problem Using the Minimal Cut Concept" Management Science 24, 1977, 393–400.

[31] S.A. Plotkin, D.B. Shmoys and É. Tardos: "Fast Approximation Algorithms for Fractional Packing and Covering Problems" Mathematics of OR 20, 1995, 257–301.

[32] M. Rahman, H. Tennakoon and C. Sechen: "Power Reduction via Near-Optimal Library-Based Cell-Size Selection", DATE 2011, 1–4.

[33] T. Reimann, C.C.N. Sze and R. Reis, "Challenges of Cell Selection Algorithms in Industrial High Performance Microprocessor Designs", Integration, the VLSI Journal 52 (C), 2016, 347–354.

[34] T. Reimann, C.C.N. Sze, R. Reis: "Cell Selection for High-Performance Designs in an Industrial Design Flow", ISPD 2016, 65–72.

[35] H. Ren and S. Dutt: "Fast and Near-Optimal Timing-Driven Cell Sizing under Cell Area and Leakage Power Constraints Using a Simplified Discrete Network Flow Algorithm", VLSI Design 2013, Article ID 474601, 15 pages.

[36] S. Roy, D. Liu, J. Um and D.Z. Pan: "OSFA: A New Paradigm of Aging Aware Gate-Sizing for Power/Performace Optimizations under Multiple Operating Conditions", IEEE TCAD 35(10), 2016, 1618–1629.

[37] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S.-M. Kang.: "An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Programming", IEEE TCAD 12(11), 1993, 1621–1634.

[38] U. Schorr: "Algorithms for Circuit Sizing in VLSI Design", dissertation thesis, Research Institute for Discrete Mathematics, University of Bonn, 2016.

[39] S. Shah, A. Srivastava, D. Sharma, D. Sylvester, D. Blaauw, and V. Zolotov: "Discrete Vt Assignment and Gate Sizing Using a Self-Snapping Continuous Formulation", ICCAD 2005, 705–712.

[40] A. Sharma, D. Chinnery, S. Bhardwaj and C. Chu: "Fast Lagrangian Relaxation Based Gate Sizing Using Multi-Threading", ICCAD 2015, 426–433.

[41] H. Tennakoon and C. Sechen: "Gate Sizing Using Lagrangian Relaxation Combined with a Fast Gradient-Based Pre-Processing Step", ICCAD 2002, 395–402.

[42] H. Tennakoon and C. Sechen: "Nonconvex Gate Delay Modeling and Delay Optimization", IEEE TCAD 27(9), 2008, 1583–1594.

[43] J. Vygen: "Near-Optimum Global Routing with Coupling, Delay Bounds, and Power Consumption", Intl. Conf. on Integer Programming and Combinatorial Optimization 2004, 308–324.

[44] J. Wang and D. Das and H. Zhou, "Gate Sizing by Lagrangian Relaxation Revisited", ICCAD 2007, 111–118.