# Incremental Layer Assignment Driven by an External Signoff Timing Engine

Vinicius Livramento, Derong Liu, Salim Chowdhury, Bei Yu, *Member, IEEE*, Xiaoqing Xu, *Student Member, IEEE*, David Z. Pan, *Fellow, IEEE*, José Luís Güntzel, *Member, IEEE*, and Luiz C. V. dos Santos

***Abstract*—Modern technologies provide wide and thick metal layers that must be wisely used to reduce the delay of critical interconnections. After global routing, incremental layer assignment can improve the circuit timing by properly selecting critical interconnect segments to be routed in the faster (but very limited) wires on upper layers. Existing techniques based on net-by-net iterative improvement may get stuck at locally-optimal solutions depending on net ordering. Recent techniques rule out such drawback through the simultaneous iterative improvement of all nets, but they unfortunately rely on objective functions that may guide the optimization off critical paths. As opposed to all reported techniques, which rely on simplified, overly pessimistic timing models, this paper proposes the decoupling of incremental layer assignment from the timing analysis and the exploitation of flow conservation conditions so as to enable the use of an external signoff timing engine. The novel technique was experimentally compared with two state-of-the art works, leading to 50% less timing violations under total negative slack metric and 35% less timing violations under worst negative slack metric with similar overhead in number of vias.**

***Index Terms*—Incremental layer assignment, Lagrangian relaxation (LR), min-cost network flow, signoff timing engine.**

## I. Introduction

THE INCREASING impact of interconnect delay on the overall circuit performance represents a bottleneck for timing closure. The worse scaling of interconnect delay, as compared to cell delay, is a consequence of the quadratic increase of wire resistance per unit length [1]. Such scenario has shifted the research spotlight toward efficient interconnect synthesis and timing optimization techniques like buffer insertion, timing-driven placement, and layer assignment.

Manuscript received July 22, 2016; revised October 7, 2016; accepted November 13, 2016. Date of publication December 12, 2016; date of current version June 16, 2017. This work was supported in part by CAPES under Grant 99999.006230/2015-06, in part by CNPQ through Project Universal under Grant 457174/2014-5, and in part by PQ under Grant 310341/2015-9. This paper was recommended by Associate Editor M. Ozdal.

V. Livramento and L. C. V. dos Santos are with the Automation and Systems Engineering Department, Federal University of Santa Catarina, Florianópolis 88040-900, Brazil (e-mail: vinicius.livramento@posgrad.ufsc.br).

D. Liu, X. Xu, and D. Z. Pan are with the Electrical and Computer Engineering Department, University of Texas at Austin, Austin, TX 78712 USA.

S. Chowdhury is with Oracle Corporation, Austin, TX 78729 USA.

B. Yu is with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong.

J. L. Güntzel is with the Computer Science Department, Federal University of Santa Catarina, Florianópolis 88040-900, Brazil.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2016.2638450

Modern technologies may provide twelve or more metal layers with different widths and thicknesses [2], where upper layers are wider and thicker than lower ones. Albeit their resistance is reduced quadratically, the upper layers require more area and therefore offer less resources for routing. This raises the importance of incremental timing-driven layer assignment techniques, which must properly reassign critical interconnect segments to upper layers in order to improve the overall circuit timing, no matter how global routing was performed (either through 3-D or 2-D routers, whether timing-aware or not) [2]–[4].

Although the choice of a proper timing engine is crucial, previous works have relied on simplified models for interconnect capacitance (lumped capacitance) and delay (Elmore's model). Such models are pessimistic because they ignore second order effects (like resistive shielding), which become prominent in the face of multiple metal layers with very different electrical characteristics. Being overly pessimistic, they end up hindering timing closure and resulting in overallocation of resources (such as vias) [5]–[7].

Albeit accurate timing engines (e.g., signoff timing analyzers) are available from conventional electronic design automation (EDA) packages, they have not been exploited for layer assignment because industrial engines (to preserve intellectual property) do not report timing information for (inner) net segments, but only for cell pins and timing endpoints. The techniques proposed so far seem to take such opacity for a barrier and keep relying on inaccurate built-in engines [8], [9].

Another limitation of such techniques lies in the inaccurate objective function [4], [8], [10]. Since they minimize the sum of net delays, which might not lead to timing improvements in the critical paths, they further hinder timing closure.

This paper proposes a novel incremental layer assignment technique that overcomes such limitations of previous works. It not only handles critical and noncritical net segments simultaneously, but also exploits flow conservation conditions to extract information for each net segment individually, thereby enabling the use of an external signoff timing engine. The main contributions of this paper are as follows.

1) A binary integer programming formulation for incremental layer assignment targeting at total negative slack (TNS) optimization while modeling each net segment separately.
2) A cast of the binary integer programming into a Lagrangian relaxation (LR) formulation that exploits flow conservation conditions to decouple the layer assignment technique from the timing analysis engine.
3) A min-cost network flow technique (to solve the LR formulation) that independently models each net segment while capturing the impact of capacitance and slew

variation on neighboring segments and cells. Besides, an efficient edge-pruning methodology reduces runtime and via count.

4) A strategy to exploit the slacks reported by a signoff timer to obtain accurate Lagrange multipliers (LMs) for net segments. A detailed analysis of the impact of the timing engine accuracy in the proposed technique is also presented.

The impact of such contributions on timing closure was experimentally compared with two state-of-the-art techniques [4], [9] for circuits derived from those available within the ICCAD 2015 Contest infrastructure [11]. A signoff analyzer was used as a golden timing engine to evaluate the final 3-D routing solutions obtained after the application of each of the three incremental layer assignment techniques under comparison. The proposed technique resulted in 50% less timing violations (under TNS metric) while using a similar number of vias, as compared to the best results obtained from the related works.

The remaining of this paper is organized as follows. Section II reviews the state-of-the-art of timing-driven layer assignment while Section III details the adopted timing modeling and the problem definition. Then Section IV presents the proposed mathematical formulation for the target problem. Section V describes the proposed technique. Section VI details its experimental evaluation. Finally, Section VII draws the main conclusion and points out future directions.

## II. Related Work

Global routing can be accomplished either by direct 3-D routing (native layer assignment) [12], [13] or through 2-D routing followed by a layer assignment step [14], [15]. *Incremental* layer assignment plays the important role of improving global routing and it can target different objectives, such as via count minimization [16], antenna alleviation [8], [17], and timing optimization [4], [9], the latter being the focus of this paper. That is why this section addresses only related works on incremental layer assignment for timing optimization.

Most methods addressing incremental layer assignment rely on a 3-D routing grid that is defined by parallel routing planes (*layers*), which are divided into rectangular cells, called *G-cells*, as illustrated in Fig. 1(a). The boundaries between adjacent cells on the same plane are associated with intraplane routing tracks. Two *G-cells* from different but consecutive planes are interconnected through vias. As a result, most layer assignment techniques model the routing grid as a graph whose vertices represent *G-cells* and whose edges represent the connectivity between two adjacent *G-cells*, as shown in Fig. 1(b). Capacities are associated with the edges (to capture routing constraints) and net pins are associated with vertices.

Several techniques perform *net-by-net* iterative improvement steps to accomplish the overall timing optimization. A fast iterative heuristic [10] was proposed to minimize the worst net delays. It relies on the notion of area quota to mimic edge capacities. In every iteration, the area quota of each net is kept proportional to its Manhattan wirelength. Li *et al.* [18] presented an interconnect synthesis technique for simultaneous buffer insertion and layer assignment when targeting slew and net delay recovery. The authors extended the classic Van Ginneken's dynamic programming algorithm to

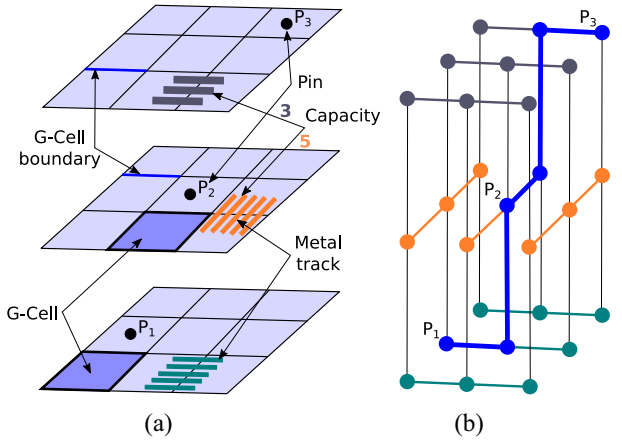


Fig. 1. (a) 3-D global routing grid with three layers and nine *G-cells* each. (b) Grid graph and 3-D routing for a 3-pin net.

accommodate new pruning strategies. The work in [19] proved that layer assignment under timing constraints is NP-complete and devised a polynomial time approximation scheme. The authors proposed a fast binary search technique that queries a dynamic programming oracle about lower and upper-bound solutions. Later on, Hu *et al.* [3] proposed a new polynomial algorithm to improve the theoretical complexity derived in their previous work [19]. The authors revisited some limitations of the dynamic programming oracle and proposed a linear-time algorithm. Dong *et al.* [8] combined dynamic programming and negotiation strategies to minimize the maximum net delay with low overhead in via count. The main limitation of all such techniques results exactly from their net-by-net approach, which may lead to locally-optimal solutions, as highlighted in [4]. The very limited availability of wide and thick wires may lead to poor timing optimization when an inadequate net ordering is adopted. Besides, some of those techniques assume that all segments of a given net must share the same layer, which may induce over-allocation.

A few techniques perform all-net simultaneous optimization. Yu *et al.* [4] observed the limitations of net-by-net strategies and proposed a min-cost flow technique to simultaneously minimize the sum of net segment delays and via delays. To handle via capacity constraints, the authors devised an LR formulation that incorporated those capacity constraints into the objective function. Therefore, the employed min-cost flow modeling was able to handle simultaneously net segment delay, edge capacity, and via overflow. Recently, Liu *et al.* [9] revisited some of the limitations from [4] and proposed a semidefinite programming formulation to handle quadratic constraints, which are used to model via delay and via capacity. Their proposed framework targets nets belonging to the critical path and employs a self-adaptive algorithm to balance the distribution of nets among different threads. Unfortunately, the objective functions adopted in such works, namely the sum of net delays or the maximum net delay, turns out limiting potential improvements. Since large net delays might not lead to a timing violation at a given timing endpoint, those objective functions may end up inducing improvements in paths that do not impair timing closure, as illustrated in Fig. 2. In addition, a net-delay-driven strategy is unaware of different setup constraints at sequential elements, which is essential to identify paths with negative slack. Fig. 3 shows

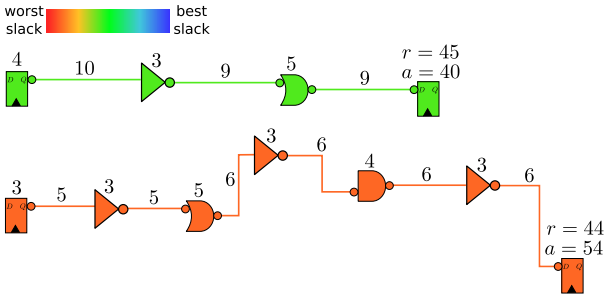


Fig. 2. Comparison between a noncritical (top) and a critical path (bottom). Cells and wires are labeled with their delays. Required (r) and arrival (a) times are indicated at timing endpoints. Note that the noncritical path has the largest net delays, whereas the critical path has the lowest. Thus, net-delay-driven approaches may guide optimization off critical paths.

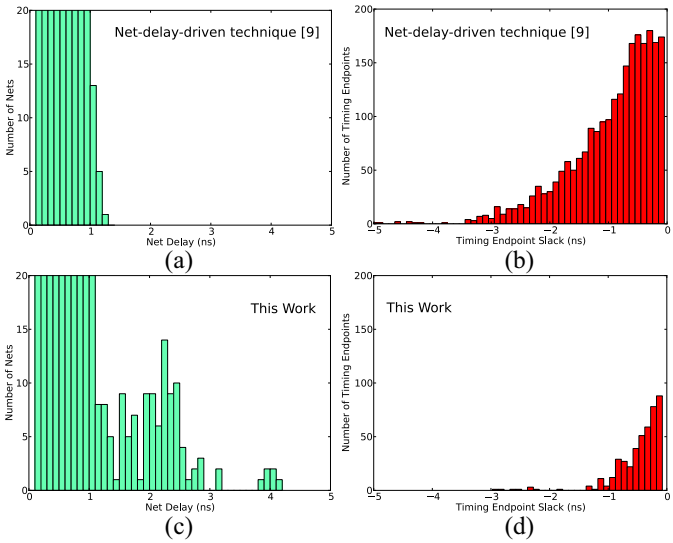


Fig. 3. Net delay histograms (left) and timing endpoint slack histograms (right) for circuit *superblue16*. (a) and (b) Behavior of the net-delay-driven technique [9]. (c) and (d) Illustration of the impact of the technique proposed in this paper.

an example of how a net-delay-driven technique may not be effective to reduce violations. Although the net-delay-driven technique [Fig. 3(a) and (b)] leads to a better compression of the net delay histogram (as expected), this does not translate into an actual compression of the negative slack histogram at the circuit's timing endpoints [Fig. 3(b) and (d)].

Most importantly, the main limitation of all incremental layer assignment techniques reported so far lies in the simplified timing model adopted to guide the optimization. The mismatch between the estimated and the actual timing is likely to hinder timing closure, as illustrated in Fig. 4. Note that the simplified model overestimates WNS in all cases but one, the mismatch ranging from 5% to 40%. It also overestimates TNS in all cases, the mismatch ranging from 20% to almost 400%. Despite the clear inadequacy of overly pessimistic engines, the accurate signoff timing analyzers available from conventional EDA packages were never used by any technique reported so far. We put this down to the opacity of such analyzers, which report timing only for cell pins and timing endpoints, but not for inner net segments. Apparently, previous techniques took such opacity as an insurmountable barrier to the use of signoff timers during optimization. On the contrary, we realized that

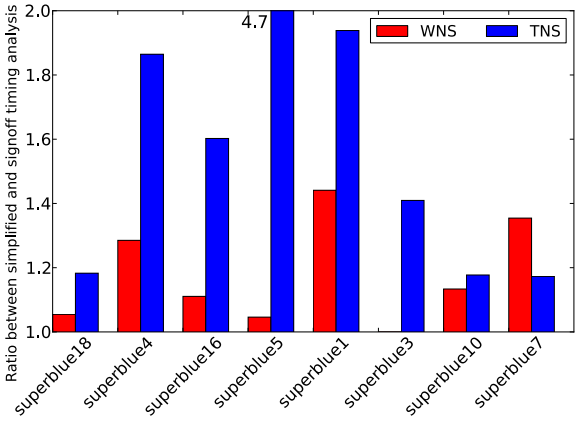


Fig. 4. Ratio between slack values obtained from a simplified timing engine (lumped capacitance and Elmore's delay) and from an industrial signoff timing engine. The over-estimation is shown for two metrics: Worst Negative Slack (WNS) and Total Negative Slack (TNS).

the key to overcoming their lack of inner observability is the exploitation of flow conservation conditions so as to extract inner timing information. This motivated us to decouple incremental layer assignment from timing analysis and to exploit flow conservation conditions for enabling the use of an accurate signoff analyzer during optimization, as described in the next section.

## III. Problem Definition

This section discusses the required background and presents the problem definition. First, Sections III-A and III-B detail the adopted routing grid and timing modeling. Finally, Section III-C defines the target problem tackled in this paper.

### A. Routing Grid Modeling

The layer assignment problem is usually defined over a 3-D routing grid, as already illustrated in Fig. 1. The routing grid can be modeled as a graph $G = (V, E)$, where each vertex represents a *G-cell* and each edge represents the connectivity between two adjacent *G-cells*. The set of edges is a partition $E = E^w + E^v$, where $E^w$ is the set of edges induced by the boundaries between *G-cells* in the same plane and $E^v$ is the set of edges induced by vias. Each edge in $E^w$ has a capacity that represents the number of detailed routing tracks allowed to pass through that edge. Therefore, assuming an initial 3-D global routing solution, the incremental layer assignment problem can be stated as follows: given a set $\mathcal{S}$ of net segment and a set $\mathcal{L}$ of routing layers, reassign the segment layers in order to optimize some objective (for instance, minimize the number of vias [16] or the number of timing violations). This paper focuses on incremental layer assignment for reducing timing violations.

### B. Timing Modeling

Most related layer assignment works are net-delay-driven and focus on the delay of the slowest nets. Nevertheless, these nets might not always represent hurdles for satisfying timing constraints, thereby not translating into improvements in critical paths. To evaluate the circuit performance and assess how far from the target frequency the circuit is, proper modeling of timing and adequate tracking of violations are required.

TABLE I
MAIN NOTATION

| | |
|---|---|
| $c_j$ | cell belonging to the set $\mathcal{C}$ |
| $s_j$ | net segment belonging to the set $\mathcal{S}$ |
| $l_q$ | routing layer belonging to the set $\mathcal{L}$ |
| $\alpha_{j,q}$ | binary variable equals 1 if $s_j$ is assigned to $l_q$ |
| $\mathcal{TE}$ | set of timing endpoints |
| $\mathcal{TS}$ | set of timing startpoints |
| $slk_j$ | slack at a given timing point $j$ |
| $a_j$ | arrival time at a given timing point $j$ |
| $r_j$ | required time at a given timing point $j$ |
| $d^c_{i,j}$ | timing arc delay $i, j$ of cell $c_j$ |
| $d^v_{i,j}$ | delay of vias connecting segments $s_i$ and $s_j$ |
| $d^s_j$ | delay of segment $s_j$ |
| $C^{down}_j$ | downstream capacitance of either cell $c_j$ or segment $s_j$ |
| $\lambda^c_{i,j}$ | Lagrange Multiplier associated with timing arc $i, j$ |
| $\lambda^s_j$ | Lagrange Multiplier associated with segment $s_j$ |

A sequential circuit can be represented by a set $\mathcal{C}$ of standard cells, a set $\mathcal{TE}$ of timing endpoints, and a set $\mathcal{TS}$ of timing startpoints. The set $\mathcal{TE}$ includes both circuit output pads and register input pins, while the set $\mathcal{TS}$ includes both circuit input pads and register output pins. There is also a set $\mathcal{N}$ of nets representing the interconnections between these elements. Circuit arrival times are measured at input/output pins of each cell $c_j \in \mathcal{C}$ and at each $j \in (\mathcal{TE} \cup \mathcal{TS})$. The arrival time, denoted as $a_j$, corresponds to the latest time when a signal transition reaches a given timing point. The required time, denoted as $r_j$, corresponds to the latest time when the signal transition must reach each $j \in \mathcal{TE}$ to ensure the target clock frequency [20]. To evaluate how far a design is from timing closure, slacks are tracked at circuit timing endpoints as: $\text{slk}_j = r_j - a_j, \forall j \in \mathcal{TE}$. Timing optimization techniques, like timing-driven placement [21] and gate sizing [22], typically employ the TNS metric to capture all timing endpoints with timing violations as follows: $\sum_{j \in \mathcal{TE}} \min(0, \text{slk}_j)$.

### C. Target Problem

Based on the previous discussions on routing grid and timing modeling, the proposed timing-driven layer assignment problem can be defined as follows. Given an initial 3-D routing solution, a set of net segments, and a routing grid with edge capacities for each layer, reassign the layers of a subset of segments so as to minimize the circuit TNS while satisfying edge capacity constraints.

## IV. PROPOSED MATHEMATICAL FORMULATION

This section discusses the proposed mathematical formulation for the target problem introduced in the previous section. Section IV-A presents the proposed binary integer programming formulation for incremental layer assignment. Next, Section IV-B shows how we cast that problem into an LR formulation which is decoupled from timing analysis. Finally, Section IV-C details how to obtain an LM for each net segment.

For convenience, Table I presents the main notation adopted in the proposed problem formulation.

### A. Proposed Binary Integer Programming Formulation

In order to formulate incremental layer assignment as a minimization problem, let us first define negative slack as $\text{slk}'_j = \min(0, \text{slk}_j)$ to ensure that only nonpositive slack values are accounted for in the objective function. Therefore the adopted objective function, defined in (1), aims to minimize the TNS. The inequality constraints (2) and (3) are introduced to model the negative slack variable $\text{slk}'_j$ used in the objective function

$$\text{Minimize} \ : -\sum_{j \in \mathcal{TE}} \text{slk}'_j \tag{1}$$

$$\text{Subject to} \ : \text{slk}'_j \leq 0, \quad \forall j \in \mathcal{TE} \tag{2}$$

$$: \text{slk}'_j \leq r_j - a_j, \quad \forall j \in \mathcal{TE}. \tag{3}$$

To obtain the arrival times at timing endpoints, a proper timing modeling is required for cells and interconnects. Let us first introduce the adopted modeling before presenting arrival time definitions. The cell delay and slew for each input/output pin-pair is represented through a nonlinear delay model, which is taken from a standard cell library. Therefore, the delay of a given cell $c_j \in \mathcal{C}$ from an input pin $i$ to its output pin is a function of the cell's input slew ($\sigma_i$) and downstream capacitance ($C^{\text{down}}_j$), being computed as $d^c_{i,j} = \delta(\sigma_i, C^{\text{down}}_j)$, where $\delta(\sigma_i, C^{\text{down}}_j)$ is a function whose value is obtained by a query into the cell lookup table. The cell output slew is computed similarly.

Interconnections are modeled as RC trees, wherein each net segment is defined as a $\pi$-model. The delay of a given net segment $s_j \in \mathcal{S}$ assigned to a layer $l_q \in \mathcal{L}$ is denoted as $d^s_j(q)$. Similar to net segments, each via is modeled as an RC $\pi$-model. In this way, the via delay between two net segments $s_i, s_j \in \mathcal{S}$ at layers $l_p, l_q \in \mathcal{L}$ is computed as $d^v_{i,j}(p, q) = \sum_{k=p}^{q-1} d^v(k)$, where $p < q$ and $d^v(k)$ corresponds to the via delay between two consecutive layers $l_k$ and $l_{k+1}$. Before introducing the arrival time modeling, let us define in (4) a binary decision variable for each net segment. The constraint (5) ensures that each segment is assigned to one and only one layer

$$\alpha_{j,q} = \begin{cases} 1, & \text{if } s_j \in \mathcal{S} \text{ is assigned to } l_q \in \mathcal{L} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$\sum_{l_q \in \mathcal{L}} \alpha_{j,q} = 1, \quad \forall s_j \in \mathcal{S}. \tag{5}$$

Let $\mathcal{I}^c_i$ denote the set of indices to each input pin of $c_j \in \mathcal{C}$. Therefore, the arrival time of cell $c_j$ can be modeled as in (6). Observe that $d^v_{i,j}$ captures the delay of vias connecting the cell input pin $i$ to its corresponding net segment. The delay of via $d^v_{i,j}$ depends on the layer assigned to the segment connected to the input pin $i$ of $c_j$ and the layer of the input pin itself. Therefore, $d^v_{i,j}$ serves as a shorthand notation for (7). For simplicity, this equation assumes that the cell pin is routed in the first layer

$$a_i + d^v_{i,j} + d^c_{i,j} \leq a_j, \quad \forall i \in \mathcal{I}^c_j, \ \text{and} \quad \forall c_j \in \mathcal{C} \tag{6}$$

$$d^v_{i,j} = \sum_{l_p \in \mathcal{L}} \alpha_{i,p} \cdot \sum_{k=1}^{p-1} d^v(k). \tag{7}$$

Typically, timing optimization techniques model the net arrival times only at source and sink pins. Unfortunately, such kind of net modeling is less flexible and therefore more appropriate for steps before global routing, when net segment information is not yet available [7], [22], [23].

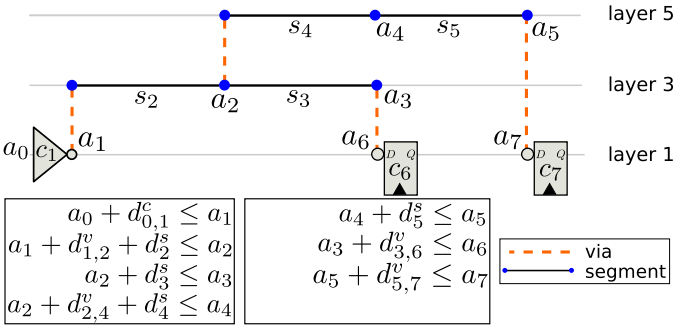


Fig. 5. Circuit example with cell and net segment arrival time modeling for horizontal three layers. Each cell and segment is labeled with an index $i$ and an arrival time $a_i$. Observe that the arrival time at segment $s_2$ is used in the inequalities that model the arrival times at segments $s_3$ and $s_4$.

Differently, we propose to split the net arrival times into a finer segment granularity, as the constraint shown in (8), where $\mathcal{I}^s_j$ denotes the set containing the index to either a segment or a pin connected to the input of $s_j \in \mathcal{S}$. Observe that $d^v_{i,j}$ captures the delay of vias connecting two consecutive segments $s_i$ and $s_j$ and depends on their assigned layers. Therefore, $d^v_{i,j}$ serves as a shorthand notation for (9). The delay of segment $d^s_j$ depends on the layer assigned to segment $s_j$ and thus serves as a shorthand notation for (10). Fig. 5 gives a small example of cell and segment arrival time modeling

$$a_i + d^v_{i,j} + d^s_j \leq a_j, \quad \forall i \in \mathcal{I}^s_j, \text{ and } \forall s_j \in \mathcal{S} \tag{8}$$

$$d^v_{i,j} = \sum_{l_p \in \mathcal{L}} \sum_{l_q \in \mathcal{L}} \alpha_{i,p} \cdot \alpha_{j,q} \cdot \sum_{k=\min(p,q)}^{\max(p,q)-1} d^v(k) \tag{9}$$

$$d^s_j = \sum_{l_q \in \mathcal{L}} d^s_j(q) \cdot \alpha_{j,q}. \tag{10}$$

Finally, the constraint (11) ensures that the edge routing capacity between two adjacent *G-cells* in the same layer is not exceeded, where $\mathcal{R}^k_i$ denotes the set of indices to each net segment routed through the edge $e_i$ on layer $l_q$

$$\sum_{j \in \mathcal{R}^q_i} \alpha_{j,q} \leq c^e_{i,q}, \quad \forall e_i \in E^w, \text{and} \quad \forall l_q \in \mathcal{L}. \tag{11}$$

### B. Proposed Lagrangian Relaxation Formulation

As highlighted in [22], incorporating timing analysis modeling directly into the optimization engine is not adequate due to the complexity of timing models adopted by modern timing engines. Therefore, we propose an LR reformulation for the problem introduced in the previous section. It decouples the optimization engine from the timing analysis tool. LR is a well-known technique that approximates the optimal solution of a given problem by removing the hard constraints and incorporating them into the cost function, as penalty terms, weighted by coefficients known as LMs. The proposed LR formulation has two key differences with respect to well-known formulations adopted by gate sizing techniques found in [7] and [22]–[24].

1) We explicitly model the arrival times at each net segment, resulting in a finer granularity as compared to the conventional modeling.
2) We show how to take advantage of flow conservation conditions to obtain LMs for each net segment individually, thereby providing a better guidance toward layer reassignment.

Therefore, we propose to relax the constraints that model the circuit timing information, i.e., (2), (3), (6), and (8), and reflect them into the objective function. The inequalities modeling negative slacks at timing endpoints are accompanied by non-negative LMs denoted as $\lambda'_j$ and $\lambda_j$. The remaining inequalities are multiplied by non-negative LMs denoted as $\lambda^c_{i,j}$ and $\lambda^s_j$ for cell timing arcs and net segments, respectively. This leads to the following relaxed objective function:

$$\begin{aligned} L_\lambda : & -\sum_{j \in \mathcal{TE}} \text{slk}'_j + \sum_{j \in \mathcal{TE}} \lambda'_j \text{slk}'_j + \sum_{j \in \mathcal{TE}} \lambda_j \left( \text{slk}'_j - r_j + a_j \right) \\ & + \sum_{c_j \in \mathcal{C}} \left( \sum_{i \in \mathcal{I}^c_j} \lambda^c_{i,j} \left( a_i + d^v_{i,j} + d^c_{i,j} - a_j \right) \right) \\ & + \sum_{s_j \in \mathcal{S}} \left( \sum_{i \in \mathcal{I}^s_j} \lambda^s_j \left( a_i + d^v_{i,j} + d^s_j - a_j \right) \right). \end{aligned} \tag{12}$$

For the subcircuit example illustrated in Fig. 5, the Lagrangian function obtained in (12) would be stated as follows:

$$\begin{aligned} L_\lambda : & -\text{slk}'_6 - \text{slk}'_7 + \lambda'_6 \text{slk}'_6 + \lambda'_7 \text{slk}'_7 \\ & + \lambda_6 \left( \text{slk}'_6 - r_6 + a_6 \right) + \lambda_7 \left( \text{slk}'_7 - r_7 + a_7 \right) \\ & + \lambda^c_{0,1} \left( a_0 + d^c_{0,1} - a_1 \right) + \lambda^s_2 \left( a_1 + d^v_{1,2} + d^s_2 - a_2 \right) \\ & + \lambda^s_3 \left( a_2 + d^s_3 - a_3 \right) + \lambda^s_4 \left( a_2 + d^v_{2,4} + d^s_4 - a_4 \right) \\ & + \lambda^s_5 \left( a_4 + d^s_5 - a_5 \right) + \lambda^c_6 \left( a_3 + d^v_{3,6} - a_6 \right) \\ & + \lambda^c_7 \left( a_5 + d^v_{5,7} - a_7 \right). \end{aligned} \tag{13}$$

Since computing arrival times and slacks inside the optimization engine is runtime extensive and not appropriate, we can rely on some flow conditions to eliminate the negative slack terms $\text{slk}'_j$ if, $\lambda'_j + \lambda_j = 1, \forall j \in \mathcal{TE}$. Besides, the required times can also be removed from the objective function because they are constant during optimization [22]. We can also eliminate cell and segment arrival times by assuming the flow conservation derived from Karush–Kuhn–Tucker conditions. Flow conservation implies that the sum of input LMs must be equal to the sum of output LMs. Therefore, cell arrival times are canceled out if $\sum_{i \in \mathcal{I}^c_j} \lambda^c_{i,j} = \sum_{j \in \mathcal{I}^c_k} \lambda^c_{j,k}$. Let $\mathcal{O}^s_j$ denote the set containing the indices to segments or pins connected to output of $s_j \in \mathcal{S}$. Thus, segment arrival times are also canceled out if $\lambda^s_j = \sum_{k \in \mathcal{O}^s_j} \lambda_k$. These flow conservation conditions can be obtained by setting to zero the partial derivatives of $L_\lambda$ with respect to each arrival time constraint [25]. Equations (14)–(16) show how to obtain some of the flow conditions for the Lagrangian function in (13)

$$\frac{\partial L_\lambda}{\partial a_1} = \lambda^s_2 - \lambda^c_{0,1} = 0 \implies \lambda^c_{0,1} = \lambda^s_2 \tag{14}$$

$$\frac{\partial L_\lambda}{\partial a_2} = \lambda^s_3 + \lambda^s_4 - \lambda^s_2 = 0 \implies \lambda^s_2 = \lambda^s_3 + \lambda^s_4 \tag{15}$$

$$\vdots$$

$$\frac{\partial L_\lambda}{\partial a_7} = \lambda_7 - \lambda^c_7 = 0 \implies \lambda^c_7 = \lambda_7. \tag{16}$$

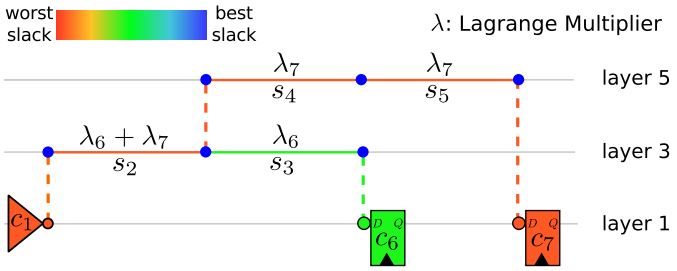


Fig. 6. Obtaining net segment LMs from sink pin LMs. LMs are labeled as $\lambda_6$ and $\lambda_7$ to better illustrate the flow conditions.

These simplifications lead to the following objective function to be minimized:

$$L_\lambda : \sum_{c_j \in \mathcal{C}} \left( \sum_{i \in \mathcal{I}_j^c} \lambda_{i,j}^c \left( d_{i,j}^v + d_{i,j}^c \right) \right) + \sum_{s_j \in \mathcal{S}} \left( \sum_{i \in \mathcal{I}_j^s} \lambda_j^s \left( d_{i,j}^v + d_j^s \right) \right). \tag{17}$$

From the simplified Lagrangian function in (17), we can conclude that, by minimizing the weighted summation of cell/segment delays and LMs, the TNS metric defined in the original objective function (1) is also minimized. Such equation can be minimized using data extracted from the cell library and *.lef* library [11], without having to compute arrival times and slacks.

The associated Lagrangian relaxed subproblem (LRS) aims to minimize the simplified Lagrangian function $L_\lambda$ by assigning a layer for each net segment, assuming a set of fixed LMs, as defined in (18). Observe that, although we relax the timing modeling constraints, the LRS is still subject to the remaining constraints, as shown in (19). From the convex optimization theory it is known that, for any fixed set of LMs, the optimal value of LRS yields a lower bound to the optimal value of the original problem. Since that lower bound depends on the set of LMs, the Lagrange dual problem (LDP) aims to maximize the lower bound from LRS by updating the LMs accordingly [25], as defined in (20), where $Q_\lambda$ represents the optimal value from LRS. Therefore, the LRS and LDP problems are solved iteratively

$$\text{LRS} : \min_{\alpha_{j,q},\ \forall s_j \in \mathcal{S}} L_\lambda \tag{18}$$

$$: \text{s.t. (4), (5), and (11)} \tag{19}$$

$$\text{LDP} : \max_{\lambda \geq 0} Q_\lambda. \tag{20}$$

### C. Obtaining Lagrange Multipliers for Net Segments

During LDP resolution, we rely on slack values computed by the timing analysis engine to update the LMs (as will be detailed in Section V-C). Although industrial timing analyzers report slack values for cell pins and for timing endpoints [22], timing information for inner net segments is not available to protect their intellectual property algorithms and avoid reverse engineering [26]. However, we show how to take advantage of the flow conservation conditions (detailed in the previous section) to obtain an LM for each net segment, even without slack information. Since interconnections are modeled as RC trees, each segment multiplier can be obtained by back-propagating the LMS from the net sinks to the net source, as illustrated in Fig. 6. For example, the multiplier for segment $s_2$ equals to the sum $\lambda_6 + \lambda_7$, as formalized in the flow conditions derived in (15). Also observe that the segment $s_2$ receives a larger LM (compared to $s_4$, for instance) due to its impact on both timing paths. Therefore, the optimization engine can take advantage of those larger multipliers to wisely select their layers and reduce timing violations.

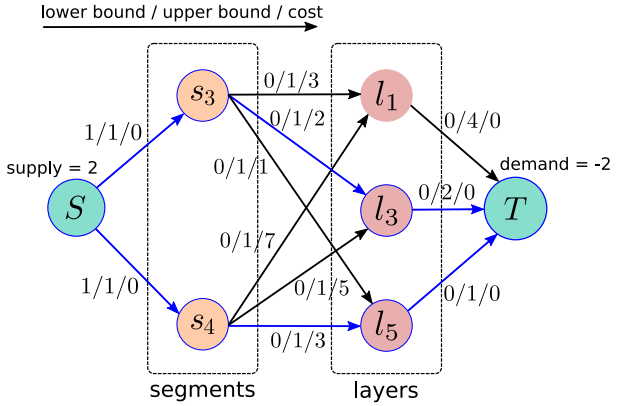


Fig. 7. Min-cost network flow example. The min-cost solution assigns $s_3$ to $l_3$ and $s_4$ to $l_5$.

## V. Proposed Technique

This section presents the proposed iterative technique to solve the problem formulation from the previous section. First, Section V-A discusses how to map the proposed instance of LRS problem into a min-cost network flow model and Section V-B details the adopted cost linearization. Finally, Section V-C overviews the proposed framework and Section V-D details the network flow graph generation.

### A. Min-Cost Network Flow Model

Solving the LRS problem as a general integer programming problem may result in prohibitive runtime, especially for large instances. To avoid this overhead, we show that LRS can be interpreted as a transportation problem and then efficiently solved using network flow algorithms. The transportation problem is a classical problem in the network flow theory to which efficient algorithms with theoretical guarantees are available in [27]. In the case of the LRS problem defined in (18) and (19), a single source and a single terminal vertex represent a factory and a warehouse, respectively, while segments and layers can be interpreted as roads with costs and capacities. The impact of assigning a segment to a layer can be interpreted as the transportation cost. Finally, the number of candidate segments for reassignment corresponds to the total flow to be transported from the source to the terminal vertex. Fig. 7 gives a min-cost network flow example for layer assignment, where each edge has a lower bound flow, an upper bound flow, and a cost. This example depicts the min-cost flow model for the segments $s_3$ and $s_4$ from Fig. 6. Notice that, since $s_4$ is more critical than $s_3$, it will be assigned to layer 5 to reduce its delay.

Considering the LRS problem from Section IV-B, the binary variable (4) is captured through the uni-modularity property inherent from this class of problems [27]. The constraint (5) can be accounted for by restricting to 1 both lower and upper bounds on the flow through the edges connecting the source vertex to the segment vertices. Upper bounds on the edges

connecting layers to the terminal vertex capture the capacity constraint (11). Unfortunately, the cost function (17) presents a few nonlinearities that prevent from modeling the cost as a linear function and use efficient state-of-the-art algorithms like network simplex, cost-scaling, and cycle-canceling [28]. Therefore, the next section presents several adopted strategies for cost linearization.

### B. Cost Linearization

The delay of a segment $s_j$ on a layer $l_q$ is computed (using Elmore's delay) as $d_j^s(q) = R_j^s(q) \cdot (C_j^s(q)/2 + C_j^{\text{down}})$, where $R_j^s(q)$ and $C_j^s(q)$ represent the segment resistance and capacitance on layer $l_q$, while $C_j^{\text{down}}$ refers to $s_j$'s downstream capacitance. The via delay between two consecutive layers $l_q$ and $l_{q+1}$ is computed, similarly to the segment delay, as $d^v(q) = R^v(q) \cdot (C^v(q)/2 + C_j^{\text{down}}))$, where $R^v(q)$ and $C^v(q)$ refer to the via resistance and capacitance, respectively, while $C_j^{\text{down}}$ captures its downstream capacitance. Notably, the impact of reassigning the layer of a given segment $s_j$ to $l_q$ does not restrict to the segment itself. In fact, besides $s_j$ itself, we should also take into account the net driver cell, the upstream segments of $s_j$, and the downstream net sink cells. Therefore, the impact of reassigning the layer of a segment can be divided in four parts.

1) Current $s_j$'s delay and the corresponding via delay from previous segment $s_i$ to $s_j$ itself. Computing the via delay between $s_i$ and $s_j$ requires a binary multiplication [see (9)], which introduces a nonlinearity in the cost function. Therefore, we employ the following linearization strategy: $\alpha_{i,p} \cdot \alpha_{j,q} \approx \alpha'_{i,p} \cdot \alpha_{j,q}$, where $\alpha'_{i,p}$ corresponds to the layer assigned to segment $s_i$ in the previous iteration. This approximation is reasonable as the proposed technique takes advantage of the iterative nature of an LR-based optimization. Especially in later iterations, when the problem starts to converge, fewer reassignments are expected to occur and therefore the approximation discrepancy also tends to be reduced. Therefore, the cost of a segment $s_j$ considering its delay and the via delay from its previous segment on layer $l_p$ is computed as in (21), where $l_q$ corresponds to $s_j$'s layer

$$\lambda_j^s \cdot \left( d_j^s(q) + \sum_{k=p}^{q-1} d^v(k) \right), \quad \text{where } i \in \mathcal{I}_j^s. \tag{21}$$

2) The downstream capacitance of the net driver cell is affected by the segment capacitance variation, which reacts on cell's delay. Assume that the reassignment of $s_j$ from layer $l_q$ to $l_{q+1}$ causes a capacitance variation $\Delta C_j = C_j^s(q+1) - C_j^s(q)$. Therefore, we can use a first-order approximation to estimate the impact on driver's delay, as shown in (22), where the partial derivative term reflects the cell delay linearization with respect to its downstream capacitance and $d_{h,i}^{c'}$ corresponds to the cell's delay in the previous iteration

$$\lambda_{h,i}^c \cdot \left( d_{h,i}^{c'} + \Delta C_j \cdot \frac{\partial d_{h,i}^c}{\partial C_i^{\text{down}}} \right), \quad \forall h \in \mathcal{I}_i^c. \tag{22}$$

3) The downstream capacitances of upstream segments are also affected by the segment capacitance variation $\Delta C_j$. Let $\Upsilon_j$ be the set of indices to $s_j$'s upstream segments. Therefore, (23) accounts for the impact on the delay of each upstream segment, where $l_p$ corresponds to $s_i$'s layer

$$\lambda_i^s \cdot R_i^s(p) \cdot \left( C_i^{\text{down}} + \Delta C_j \right), \quad \forall i \in \Upsilon_j. \tag{23}$$

4) The reassignment of segment $s_j$ also causes a net slew variation, which in turn impacts on the delay of the sink cells belonging to the downstream path of $s_j$. Assuming that the reassignment of segment $s_j$ from layer $l_q$ to $l_{q+1}$ causes a slew variation $\Delta\sigma_j = \sigma_j^s(q+1) - \sigma_j^s(q)$, we can also use a first-order approximation to compute the delay of each sink, as in (24). In this equation, the partial derivative term reflects the cell delay linearization with respect to its input slew and $\mathcal{D}_j^s$ denotes the set of indices to downstream sink pins. To compute the slew of each segment on the path from $s_j$ to the net sinks, we adopt the PERI model and Bakoglu's metric (step slew computation) [29], as detailed in the following equation:

$$\lambda_{j,k}^c \cdot \left( d_{j,k}^{c'} + \Delta\sigma_j \cdot \frac{\partial d_{j,k}^c}{\partial \sigma_j} \right), \quad \forall k \in \mathcal{D}_j^s \tag{24}$$

$$\sigma_j^s(q) = \sqrt{\sigma_i^2 + \left( R_j^s(q) \cdot C_j^{\text{down}} \cdot \ln 9 \right)^2}. \tag{25}$$

Equations (21)–(25) are then used to compute the reassignment cost of each segment. The next section details the proposed min-cost flow framework.

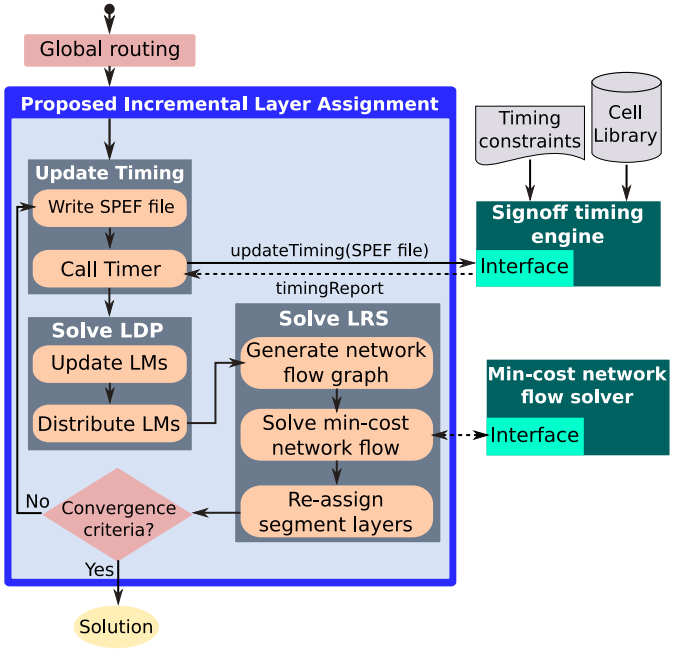


Fig. 8. Overview of the proposed framework.

### C. Proposed Framework

Fig. 8 gives an overview of the proposed incremental layer assignment framework. It receives as inputs a 3-D global routing solution. The incremental layer assignment problem is solved in three major steps.

1) *Update Timing* step first writes a parasitics file in the standard parasitic exchange format (SPEF) containing the distributed RC network information for the

circuit. Then the *Call Timer* invokes the external timing engine through a *tcl-socket* interface (similar to the one described in [6]), which returns a report containing the slacks for each timing point.

2) *Solve LDP* consists of two substeps. The first one updates LMs so as to increase or decrease their values proportionally to the severity of timing violations measured from the reported slacks. The second substep distributes LMs so as to comply with flow conservation conditions. Given the vector $\vec{\lambda}$ of LMs and the vector $\vec{\text{slk}}$ of slacks computed during the timing analysis, Algorithm 1 describes how LMs are updated and distributed. The first substep (encapsulated in function UPDATE_LMs) relies on the subgradient method [22] to update LMs for each circuit pin, including cell pins and timing endpoints. The LMs are updated proportionally to the ratio of pin slack to the worst negative slack (WNS) by visiting each timing endpoint (lines 4–6) and every cell timing arc (lines 7–11). Our multiplier updating approach is straightforward and similar to the one adopted in [30]. The major difference is that we employ the WNS as a normalization factor, instead of the target clock period. The second substep (encapsulated in function DISTRIBUTE_LMs) proportionally distributes the sum of the output LMs of every cell to each of its inputs. The distribution is accomplished by visiting cells in reverse topological order (lines 13–18), where $\mathcal{O}_j^c$ denotes the set of indices to cells or TEs that are connected to the output of $c_j$. Finally, LMs are obtained for each net segment (lines 19–23), as already exemplified in Fig. 6.

3) After defining the LMs for each net segment, the *Solve LRS* step aims to solve the problem stated in (18) and (19). First, it selects critical and noncritical net segments and their respective target layers to generate the network flow graph, as it will be detailed in Section V-D. Then the *Solve min-cost network flow* step invokes the network flow solver. Among several algorithms available in [27], we employed a cancel-and-tighten implementation of cycle-canceling algorithm, which is very efficient and stable in practice and has a strongly polynomial runtime complexity [28]. Finally, *Reassign segment layers* accomplishes the optimal assignment found by the min-cost flow solver.

The three explained steps are repeated until predefined convergence criteria is reached.

**Algorithm 1:** Solve LDP ($\vec{\lambda}$, $\vec{\text{slk}}$)

```
1  UPDATE_LMs(λ⃗, slk⃗);
2  DISTRIBUTE_LMs(λ⃗, slk⃗);
3  Function UPDATE_LMs(λ⃗, slk⃗)
4      foreach j ∈ TE do
5          λ_j ← λ_j × (1 + slk_j / WNS);
6      end
7      foreach c_j ∈ C do
8          foreach i ∈ I_j^c do
9              λ_{i,j}^c ← λ_{i,j}^c × (1 + slk_j / WNS);
10         end
11     end
12 Function DISTRIBUTE_LMs(λ⃗, slk⃗)
13     foreach c_j ∈ C do
14         μ_j ← Σ_{i∈I_j^c} λ_{i,j}^c;
15         foreach i ∈ I_j^c do
16             λ_{i,j}^c ← (λ_{i,j}^c / μ_j) × Σ_{k∈O_j^c} λ_{j,k}^c;
17         end
18     end
19     foreach n ∈ N do
20         foreach s_j ∈ n in reverse topological order do
21             λ_j^s ← Σ_{k∈O_j^s} λ_k^s;
22         end
23     end
```

### D. Network Flow Graph Generation

For a given global routing, alternative network graphs could be built, depending on which segments are considered timing critical or not. That is why the proposed network flow graph generation relies on two input parameters, num_TE and $\alpha$, which provide the criteria to select critical and noncritical segments. The first parameter specifies the number of timing endpoints (output pads and inputs of sequential elements) with negative slacks to be used for candidate net selection; the second parameter specifies a threshold factor to help defining whether a net segment should be considered critical or not. The generation of a network graph is performed in three phases.

1) Selection of critical segments.
2) Insertion of the vertices and edges associated with critical segments (represented by set $\Gamma_c$).
3) Insertion of vertices and edges associated with noncritical segments (represented by set $\Gamma_{nc}$).

Algorithm 2 describes the building of the network flow graph $N(V, E)$ induced by the parameters num_TE and $\alpha$. The algorithm first creates source and terminal vertices, properly initializes the sets in which critical and noncritical segments will be included, and selects the num_TE timing endpoints with worst slack (lines 1–3), before launching the three-phase procedure. Phase 1 (lines 4–14) selects critical segments by traversing the circuit in reverse topological order, starting at each selected timing endpoint until a timing startpoint is reached. For each visited net, the maximum among the LMs of all its segments ($\lambda^s_{\max}$) is determined. The algorithm selects as critical segments (line 10) all segments of a given net whose LMs are higher than or equal to the product of the threshold factor by the maximum LM.

Phase 2 (lines 15–28) inserts in the graph the vertices ($v_j$) representing the critical segments selected by phase 1, the vertices ($v_q$) representing candidate layers, and the edges connecting them ($v_j$, $v_q$). After a vertex $v_j$ is inserted for each critical segment (line 16), candidate layers are selected for it such that all layers below the current one are pruned (line 18). Since the delays of critical segments are expected to be reduced and the benefits from promoting them to upper layers are higher than assigning them to lower layers (because resistance decreases quadratically), the pruning of lower layers reduces the number of edges in the graph without compromising timing improvements. Then the algorithm inserts as many vertices $v_q$ as the number of candidate layers (line 20) and as many edges ($v_j$, $v_q$) (line 21). Next, the algorithm identifies (at line 23) the set of segments overlapping with the current segment $s_j$ at a given layer $l_q$. The algorithm selects as noncritical segments all vertices in that set whose LMs are less than the product of the threshold factor by the LM of the current segment.

Phase 3 (lines 29–39) inserts in the graph the vertices ($v_j$) representing the noncritical segments selected during phase 2,

**Algorithm 2:** Generate Network Flow Graph (num_TE, $\alpha$)

```
1  Insert the source vertex v_src and terminal vertex v_ter in V;
2  Γ_c ← ∅;  Γ_nc ← ∅;
3  B ← num_TE timing endpoints with worst slacks;
4  foreach β ∈ B do
5      pin ← β
6      while pin ≠ timing_startpoint do
7          net ← net connected to pin;
8          λ^s_max ← highest LM for all segments in the set net;
9          foreach segment s_j in the set net do
10             if λ^s_j ≥ (α × λ^s_max) then insert segment s_j in Γ_c;
11         end
12         pin ← input pin of net driver cell with worst slack;
13     end
14 end
15 foreach critical segment s_j ∈ Γ_c do
16     Insert vertex v_j in V;
17     Insert edge (v_j, v_src) in E with cost=0 and required flow=1;
18     foreach layer q = s_j's current layer to |L| do
19         κ ← cost of s_j when assigned to l_q based on (21) to (24);
20         Insert vertex v_q in V;
21         Insert edge (v_j, v_q) in E with cost κ;
22         Insert edge (v_q, v_ter) in E with cost=0 and capacity=c^e_{j,q};
23         O ← segments overlapping with s_j at l_q;
24         foreach segment s_i in the set O do
25             if λ^s_i < (α × λ^s_j) then insert segment s_i in Γ_nc ;
26         end
27     end
28 end
29 foreach non-critical segment s_j ∈ Γ_nc do
30     Insert vertex v_j in V;
31     Insert edge (v_j, v_src) in E with cost=0 and required flow=1;
32     foreach layer q = 1 to s_j's current layer do
33         κ ← cost of s_j when assigned to l_q based on (21) to (24);
34         if assigning s_j to l_q does not introduce slew violation then
35             Insert edge (v_j, v_q) in E with cost=κ;
36             Insert edge (v_q, v_ter) in E cost=0 and capacity=c^e_{j,q};
37         end
38     end
39 end
40 set supply of v_src = |Γ_c| + |Γ_nc|;
41 set demand of v_ter = −(|Γ_c| + |Γ_nc|);
42 return N(V, E);
```

the vertices ($v_q$) representing candidate layers, and the edges connecting them ($v_j$, $v_q$). After a vertex $v_j$ is inserted for each noncritical segment (line 30), candidate layers are selected for it such that all layers above the current one are pruned (line 32). Since noncritical segments might release their layers for critical segments, the pruning of upper layers precludes them to use these scarce resources without compromising timing improvements. Besides, candidate layers that would introduce slew violation are also pruned (line 34). Finally, supply and demand attributes are assigned to source and terminal vertices, respectively (lines 40 and 41).

Albeit no guarantee can be provided to completely rule out the risk for layer oscillation of timing-critical segments from one iteration to another, the proposed technique relies on the following feature to avoid oscillation: the selection of noncritical candidate segments (Algorithm 2, line 25) is based on the scaled LMs. In other words, a segment is selected as a noncritical candidate in a given iteration if its LM is less than the scaled multiplier of the critical segment. Therefore, if a critical segment becomes noncritical from one iteration to another, it is likely that its new LM value is still sufficiently high so as to prevent it from being selected as a noncritical candidate.

Note that the proposed approach, being an incremental optimization, never lets a segment unassigned, because a segment that is not reassigned is kept preassigned according to the initial legal solution used as a starting point.

## VI. Experimental Results

This section presents the experimental evaluation of the proposed technique. The algorithms were implemented in C++ and the min-cost network flow instances were solved using the LEMON library [31].

Section VI-A details the experimental infrastructure. Section VI-B compares the proposed technique with two state-of-the-art timing-driven incremental layer assignment techniques [4], [9] under an industrial timer, while Section VI-C presents an experimental comparison under a simplified timer. Then Section VI-D puts the results into a different perspective to analyze the impact of the timing engine accuracy to guide the optimization. Section VI-E shows how to exploit a hybrid timer to achieve a good tradeoff between runtime and quality. Finally, Section VI-F provides an insightful experimental analysis of the algorithmic decisions.

### A. Infrastructure

Due to the lack of public domain experimental infrastructures on timing-driven layer assignment, we adapted the ICCAD 2015 Incremental Timing-Driven Contest infrastructure [11]. The ICCAD 2015 Contest infrastructure contains 8 circuits with sizes from 768K to 1.93M cells and a nonlinear delay model timing library. That infrastructure was developed considering the Free PDK 45 nm technology library file and unlike the popular ISPD 2008 global routing benchmarks, provides detailed information on cell timing (*Liberty* format) and circuit timing constraints (synopsys design constraints (SDC) format). This makes the ICCAD 2015 benchmarks appropriate to comparatively evaluate techniques targeting the reduction of timing violations.

We adopted $9 \times 9$ circuit row-height as the size of a $G$-cell, which is exactly the same used to compute the placement density. We used the ten metal layers provided in the adopted Free PDK 45 nm library. To compute the number of detailed routing tracks available for each layer, we relied on the metal pitches reported in that library, which also provides resistance and capacitance information for metal layers and vias. After the default grid capacities were set, we adjusted them to account for macro blocks, which are routed in the first four metal layers. The global router NCTU-GR [15] was invoked to generate the initial 2-D routing, while the layer assignment was obtained with the tool NVM [16]. All circuits were routed without any edge capacity overflows. It is worth mentioning that any technique could be used to obtain the initial 3-D routing solution such as [12] and [13].

For all experiments except those reported in Section VI-C, the industrial signoff timer Synopsys PrimeTime, version L-2016.06, was used as golden engine to evaluate the final 3-D routing solutions obtained by the techniques under comparison. To invoke the industrial timing engine, four industrial format files were used.

1) *Verilog* containing the circuit description.
2) *SDC* detailing the timing constraints.
3) *Liberty* library describing the cell timing information.
4) *SPEF* file containing the distributed RC networks.

### B. Comparison Under Industrial Timer

TILA [4] and CPLA [9] are the most recent net-delay-driven approaches and they have the same goal as ours: to improve timing closure. That is why they were selected for a joint comparative evaluation with the proposed technique. The distinct

TABLE II
EXPERIMENTAL RESULTS EVALUATED UNDER AN INDUSTRIAL TIMING ENGINE. FOR THE RUNTIME OF THE PROPOSED TECHNIQUE, WE ALSO REPORTED, BETWEEN PARENTHESIS, HOW MUCH OF THE RUNTIME IS TAKEN BY THE INDUSTRIAL TIMING ANALYZER

| Circuit | Solution | TNS ($\mu s$) | WNS ($ns$) | # of TEs with negative slack | # Vias ($10^6$) | # Vias OVF ($10^3$) | Runtime (min.) | Crit. / Non. Crit. |
|---|---|---|---|---|---|---|---|---|
| **superblue18** | Initial | -1.73 | -6.01 | 612 | 1.24 | 5.87 | — | — |
| Grid: 301x199 | TILA [4] | -1.29 | -5.05 | 446 | 1.34 | 6.20 | 1.6 | 0.5% / 0.5% |
| Nets: 771K | CPLA [9] | -1.24 | -4.80 | 425 | 1.29 | 6.37 | 15.7 | 0.5% / 0.5% |
| Segs: 3.7M | Ours | -0.61 | -3.49 | 396 | 1.28 | 5.93 | 13.3 (11.1) | 0.5% / 1.1% |
| **superblue4** | Initial | -4.02 | -6.64 | 1,745 | 1.61 | 12.04 | — | — |
| Grid: 369x205 | TILA [4] | -3.59 | -5.84 | 1,636 | 1.70 | 12.19 | 2.1 | 0.5% / 0.5% |
| Nets: 802K | CPLA [9] | -3.76 | -6.29 | 1,626 | 1.66 | 12.59 | 14.1 | 0.5% / 0.5% |
| Segs: 4.6M | Ours | -1.87 | -2.93 | 1,116 | 1.70 | 12.29 | 15.2 (12.1) | 0.3% / 1.9% |
| **superblue16** | Initial | -3.08 | -5.06 | 3,064 | 1.48 | 4.99 | — | — |
| Grid: 367x199 | TILA [4] | -2.24 | -4.91 | 2,462 | 1.57 | 5.58 | 2.2 | 0.5% / 0.5% |
| Nets: 999K | CPLA [9] | -2.21 | -5.05 | 2,404 | 1.54 | 5.92 | 24.6 | 0.5% / 0.5% |
| Segs: 6.0M | Ours | -0.20 | -2.98 | 486 | 1.55 | 5.30 | 18.6 (14.7) | 0.3% / 1.8% |
| **superblue5** | Initial | -1.46 | -24.15 | 410 | 2.37 | 23.98 | — | — |
| Grid: 611x282 | TILA [4] | -1.48 | -18.81 | 401 | 2.48 | 24.55 | 2.6 | 0.5% / 0.5% |
| Nets: 1.1M | CPLA [9] | -1.42 | -18.55 | 389 | 2.44 | 24.95 | 43.0 | 0.5% / 0.5% |
| Segs: 6.9M | Ours | -0.64 | -10.27 | 351 | 2.42 | 24.12 | 20.4 (17.0) | 0.4% / 0.6% |
| **superblue1** | Initial | -0.45 | -4.90 | 188 | 2.20 | 11.98 | — | — |
| Grid: 556x204 | TILA [4] | -0.41 | -4.78 | 176 | 2.32 | 12.07 | 3.4 | 0.5% / 0.5% |
| Nets: 1.2M | CPLA [9] | -0.41 | -4.78 | 177 | 2.27 | 12.92 | 28.6 | 0.5% / 0.5% |
| Segs: 6.2M | Ours | -0.19 | -3.23 | 131 | 2.30 | 12.34 | 21.5 (17.4) | 0.6% / 0.8% |
| **superblue3** | Initial | -1.55 | -12.89 | 397 | 2.31 | 10.96 | — | — |
| Grid: 632x205 | TILA [4] | -1.53 | -11.12 | 376 | 2.45 | 11.23 | 2.4 | 0.5% / 0.5% |
| Nets: 1.2M | CPLA [9] | -1.53 | -10.93 | 374 | 2.37 | 11.27 | 30.0 | 0.5% / 0.5% |
| Segs: 7.4M | Ours | -1.40 | -8.86 | 378 | 2.34 | 11.12 | 22.3 (18.7) | 0.6% / 0.6% |
| **superblue10** | Initial | -33.72 | -14.33 | 6,108 | 3.52 | 15.30 | — | — |
| Grid: 538x383 | TILA [4] | -31.08 | -12.88 | 6,031 | 3.69 | 18.71 | 5.9 | 0.5% / 0.5% |
| Nets: 1.8M | CPLA [9] | -30.75 | -12.84 | 6,057 | 3.67 | 21.76 | 101.7 | 0.5% / 0.5% |
| Segs: 13.4M | Ours | -19.70 | -8.62 | 5,398 | 3.73 | 16.20 | 39.6 (30.6) | 0.3% / 1.0% |
| **superblue7** | Initial | -2.67 | -8.68 | 1,438 | 2.92 | 13.21 | — | — |
| Grid: 394x352 | TILA [4] | -1.68 | -7.63 | 1,029 | 3.15 | 13.50 | 4.0 | 0.5% / 0.5% |
| Nets: 1.9M | CPLA [9] | -1.47 | -6.98 | 952 | 3.03 | 14.13 | 35.9 | 0.5% / 0.5% |
| Segs: 9.05M | Ours | -0.76 | -5.29 | 816 | 2.95 | 13.46 | 31.2 (26.5) | 0.1% / 0.3% |
| **Average reduction vs.** | TILA [4] | 50.20% | 35.28% | 23.95% | 2.39% | 2.84% | — | — |
| **Average reduction vs.** | CPLA [9] | 49.10% | 34.36% | 22.18% | -0.07% | 7.41% | — | — |

objective functions of the techniques under comparison should be seen as different drivers toward the same ultimate goal. The simpler objective functions employed by TILA and CPLA (i.e., the sum of net segment delay and via delay) actually drive optimization toward better timing. However, we claim that they miss opportunities to reduce the number of violations. We rely on a golden signoff timing engine to provide evidences for supporting our claim. Although the compared techniques solve distinct instances of (essentially) the same general optimization problem, the direct comparison between them allows us to assess to which extent each instance leads to inferior or superior solutions with respect to the ultimate goal of reaching timing closure.

The binaries of TILA and CPLA were obtained with their authors. The experiments were performed on a workstation with a 3.2 GHz Intel i5 CPU with 32GB RAM. To run our technique, we set the parameters num_TE $= 20$, $\alpha = 0.001$ for Algorithm 2 and we adopt 8 as the number of iterations of our technique's main loop (Fig. 8). The initial values for all the LMs were set to 1. We also set Synopsys PrimeTime as the timing engine to be iteratively invoked within our technique.

Table II displays the overall circuit characteristics and results. For each circuit, four distinct solutions were evaluated under each metric and labeled as follows. Initial corresponds to the initial solutions generated with NCTU-GR and NVM tools. TILA [4] and CPLA [9] correspond to the results from the net-delay-driven techniques. Ours reports the results obtained by the proposed technique. Columns 3–5 show the reports from the signoff timer for the following timing metrics: WNS, TNS, and number of timing endpoints with negative slack. Column 6 reports the number of vias while column 7 shows the number of vias overflow computed according to the metric detailed in [16]. Column 8 displays the runtime in minutes. The last column reports the percentage of critical and noncritical nets selected for each technique for reassignment. Finally, the two bottom rows detail, for each metric, the average reduction obtained by our technique when compared to TILA and CPLA.

The results concerning TNS and WNS metrics reveal that the proposed technique leads to around 50% and 35% less timing violations than the two related techniques. As a consequence, a reduction of roughly 23% in the number of timing

endpoints with violations is observed. Besides the timing violation reductions, the proposed technique requires 2.4% less vias than TILA and roughly the same number of vias as CPLA. In addition, the number of via overflows is 2.8% and 7.4% smaller than TILA and CPLA, respectively. The small overhead in the number of vias required by the proposed technique is an important strength since vias degrade both manufacturing yield and circuit timing [17]. Therefore, the obtained timing reduction with smaller via penalties puts in evidence that, while focusing on the slowest nets, net-delay-driven techniques might overlook several net segments that truly affect the circuit critical paths. This has two main causes.

1) A timing endpoint with negative slack may not result from a single very slow net, but from a chain of several slow nets and cells.
2) Different setup constraints at different sequential elements have also an important impact on circuit slacks, which is overlooked by a net-delay-driven strategy.

Analyzing the runtime column, one can observe that TILA is roughly 8 times faster than both CPLA and the proposed technique. The longer runtime taken by the CPLA technique was due to the semidefine programming solver, which is much slower than a state-of-the-art network flow solver (employed in TILA) [9]. On the other hand, the longer runtime taken by the proposed technique was due to the iterative invocation of the signoff timing engine to obtain up-to-date timing reports. As a consequence, the timing analysis takes around 81.7% of the total runtime of the proposed technique. The average runtime for updating the LMs corresponds to 3.9% of the total runtime while the LRS takes 8.5% (including the min-cost flow solver runtime). The remaining 5.9% of the runtime is consumed by parsing and initialization routines. The longer runtime taken by the proposed technique when compared to TILA can be seen as a price to pay to get accurate timing reports to better guide the net segment reassignments. Nevertheless, the runtime seems reasonable since the proposed technique can optimize circuits with up to 2M nets in less than 40 min.

The last column reports the percentage of critical and noncritical nets selected as candidates for reassignment at each iteration. The two related techniques rely on the same mechanism to select the candidate nets, i.e., 0.5% of critical and 0.5% of noncritical nets are candidates for reassignment. Although these ratios can be adjusted, we employed 0.5% to match the ratios reported in the experimental evaluation from [9]. Recall from Algorithm 2 that the percentage of critical and noncritical nets in the proposed technique is a consequence of parameter num_TE. For a fair comparison with related techniques, we selected num_TE $= 20$ to roughly obtain a similar percentage of critical and noncritical selected nets. Nevertheless, Section VI-F analyzes, for different values of num_TE, the tradeoff between TNS reduction and runtime obtained by the proposed technique.

### C. Comparison Under Simplified Timer

The improvements shown in the previous section are mainly due to our accurate problem formulation and to the use of an industrial timing engine to guide the optimization. To put in evidence the effectiveness of the proposed problem formulation and also isolate the impact of the timing engine, the related techniques should be compared with the proposed technique when the latter is driven by a simplified timer (lumped capacitance and Elmore's delay). Table III reports such experimental comparison with TNS and WNS values reported by a simplified timer. From these results, it is possible to conclude that, even when guided and evaluated by a simplified engine, our technique outperforms the related techniques by 31% and 30% under TNS and WNS metrics, respectively. Furthermore, it is worth noting that, when the proposed technique is guided by the simplified timer, it obtains the shortest runtimes among the techniques under comparison. These results clearly evidence that the proposed problem formulation is robust enough to use different timing engines.

TABLE III
EXPERIMENTAL RESULTS UNDER A SIMPLIFIED TIMER

| Circuit | Solution | TNS ($\mu s$) | WNS ($ns$) | Runtime (min.) |
|---|---|---|---|---|
| **superblue18** | TILA [4] | -1.50 | -5.89 | 1.6 |
| | CPLA [9] | -1.44 | -5.89 | 15.7 |
| | Ours | -1.23 | -5.14 | 1.1 |
| **superblue4** | TILA [4] | -4.84 | -6.43 | 2.1 |
| | CPLA [9] | -5.23 | -6.77 | 14.1 |
| | Ours | -3.36 | -4.19 | 1.2 |
| **superblue16** | TILA [4] | -3.89 | -5.24 | 2.2 |
| | CPLA [9] | -3.87 | -5.17 | 24.6 |
| | Ours | -0.76 | -3.24 | 1.4 |
| **superblue5** | TILA [4] | -6.47 | -19.31 | 2.6 |
| | CPLA [9] | -6.12 | -19.10 | 43.0 |
| | Ours | -6.35 | -11.03 | 1.4 |
| **superblue1** | TILA [4] | -0.74 | -6.69 | 3.4 |
| | CPLA [9] | -0.86 | -7.07 | 28.6 |
| | Ours | -0.53 | -5.35 | 1.8 |
| **superblue3** | TILA [4] | -2.15 | -12.68 | 2.4 |
| | CPLA [9] | -2.16 | -12.68 | 30.0 |
| | Ours | -1.77 | -10.43 | 1.9 |
| **superblue10** | TILA [4] | -36.31 | -14.26 | 5.9 |
| | CPLA [9] | -36.17 | -14.17 | 101.7 |
| | Ours | -24.55 | -9.54 | 2.7 |
| **superblue7** | TILA [4] | -1.89 | -9.90 | 4.0 |
| | CPLA [9] | -1.69 | -9.15 | 35.9 |
| | Ours | -1.14 | -5.48 | 3.0 |
| **Average red. vs.** | TILA | 31.20% | 30.52% | — |
| **Average red. vs.** | CPLA | 30.98% | 30.67% | — |

### D. Impact of Timing Engine Accuracy

In order to evaluate the importance of using of a signoff engine to guide the optimization, we also ran our technique by replacing the industrial timing engine by a simplified (lumped capacitance and Elmore's delay) built-in timing engine. The graphics in Fig. 9 compare the TNS and WNS reductions from Table II with the ones obtained by our technique when the optimization is guided by the simplified timing engine, referred to as *Ours(simplified)*. Both graphics assume as baseline the reductions obtained by our technique when the optimization is guided by the signoff timing engine, referred to as *Ours(industrial)*. The huge pessimism introduced by the simplified engine guided the technique to optimize paths that are not truly critical, achieving roughly 60% of the TNS reduction obtained when the signoff engine is employed. For some particular cases (e.g., *superblue10* and *superblue7*), that gap is smaller due to the reduced impact of second order effects like resistive shielding. For the WNS metric, the pessimism

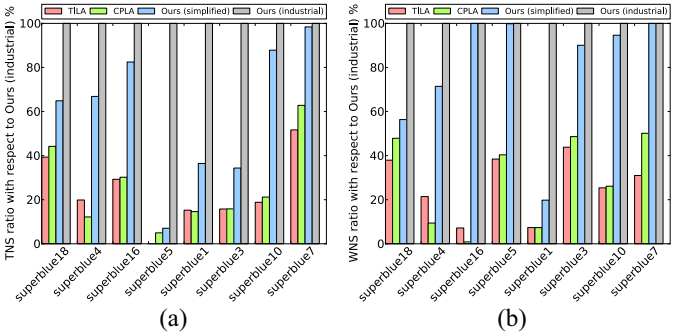


Fig. 9. Ratio between the timing violation reduction obtained by each technique with respect to our technique using the industrial signoff timer under (a) TNS and (b) WNS metric. Ours (industrial) corresponds to the results reported in Table II while ours (simplified) corresponds to the proposed technique using a simplified timing engine.

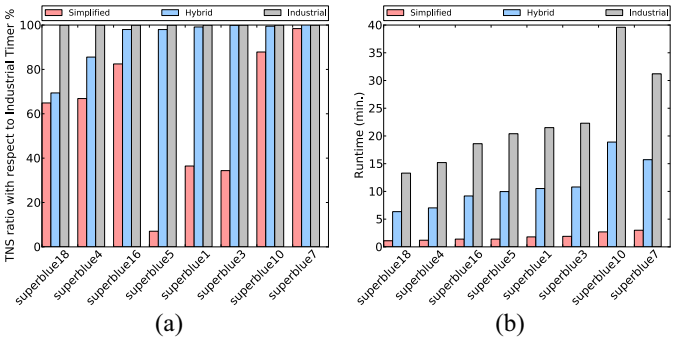


Fig. 10. Impact of using distinct timing engines: simplified, hybrid and industrial (signoff). (a) TNS reduction for distinct timing engines normalized to an industrial engine and (b) runtime.

led to a gap of 20% when different timing engines are used. Besides the importance of using an accurate timing engine, the charts in Fig. 9 also put in evidence the robustness of the proposed technique, since it outperforms the related techniques even when a simplified timing engine is used.

## E. Impact of Hybrid Timer

Although the use of an industrial timer provides more accurate guidance toward timing violation reduction, it leads to a runtime overhead, as shown in Section VI-B. Therefore, this section investigates the use of a hybrid timer as an alternative to alleviate the runtime overhead from the use of an industrial timer during the whole optimization flow. We ran an experiment with the proposed technique where the first four iterations invoked the simplified timing engine and the last four iterations invoked the industrial timing engine. Fig. 10(a) and (b) depicts the TNS reduction and runtime obtained for three different timing engines: simplified, hybrid and industrial (signoff). Observe that the hybrid engine can obtain reductions very close to those obtained by the industrial engine for six out of eight circuits. In addition, it takes approximately half of the runtime of the industrial timing engine. Therefore, it can be used as an effective alternative to reduce the runtime. The worst TNS reduction from the hybrid engine on circuit superblue18 when compared to the industrial engine is due to the over-optimization of the critical path during the first four iterations, leaving behind the near-critical paths, which also affect TNS.

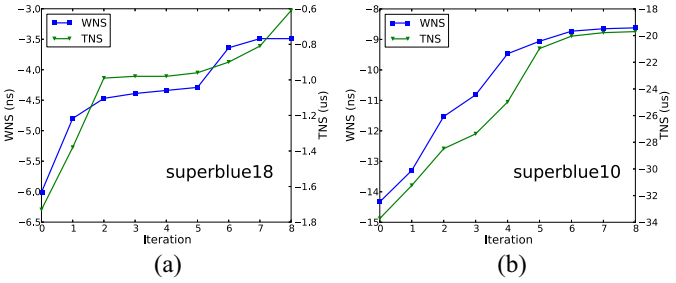


Fig. 11. Algorithm convergence under WNS and TNS metrics through the iterations for (a) *superblue18* and (b) *superblue10*.

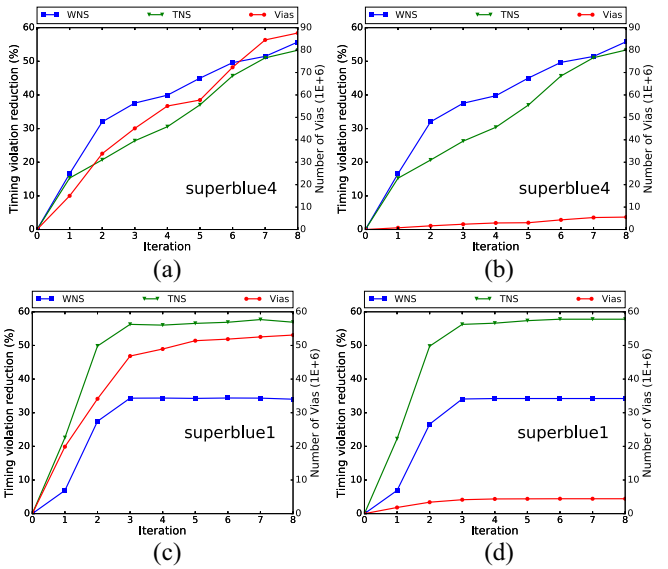


Fig. 12. Impact of the pruning strategy on timing violation reduction (TNS and WNS) and number of vias for circuits *superblue4* and *superblue1* under two different scenarios. Results (a) and (c) without pruning and (b) and (d) with pruning.

## F. Impact of Algorithmic Decisions

Fig. 11 shows the convergence of the proposed technique under WNS (left $y$-axis) and TNS (right $y$-axis) throughout the iterations (iteration ZERO corresponds to the initial solution) for two different circuits. From these charts it is clear that the proposed technique smoothly converges for both WNS and TNS metrics with slight oscillations.

Fig. 12 evaluates the impact of the proposed strategy for pruning lower layer candidates from critical net segments (Algorithm 2, line 18) and upper layer candidates from noncritical ones (Algorithm 2, line 32). The left side charts give the algorithm behavior for two different circuits when the pruning strategy is not employed whereas the right-hand side charts depict the behavior when it is employed. Observe that for both scenarios, the obtained TNS and WNS reductions are almost the same but the required number of vias is much lower when pruning is performed. The higher number of vias required by the scenario without pruning is mainly due to noncritical segments. Since they can be reassigned to lower and upper layers to release edges for critical segments, upper layers are generally preferred. However, the assignment of noncritical segments on upper layers considerably increases the number of vias since most of the circuit net segments are routed in lower

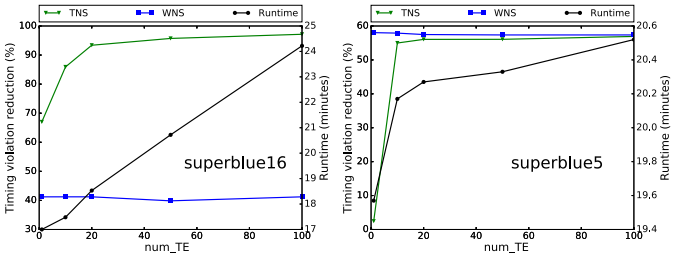


Fig. 13. Tradeoff between timing violation reduction (TNS and WNS) and runtime for different number of TEs. Circuit (a) *superblue16* and (b) *superblue5*.

layers due to their larger capacity. This leads us to conclude that the adopted pruning strategy is effective.

Fig. 13 brings the tradeoff analysis between timing violation reduction (TNS and WNS metrics) and runtime when the number of considered TEs (i.e., argument num_TE from Algorithm 2) is increased. Observe that the proposed technique shows a consistent reduction of TNS when the number of TEs increases, while the runtime increase is roughly linear. The higher number of TEs increases the number of segments for reassignment which, by its turn, affects the min-cost network flow solving runtime. Also observe that for both charts, the TNS curve becomes flat after a certain number of TEs. This happens because, from that point on, the number of selected critical nets is roughly the same, even though the number of TEs increases. This is a consequence of path sharing among different nets. For both charts, the WNS reduction curves are roughly flat, regardless of the number of TEs, and only small WNS oscillations ($<$1.5%) are observed due to the number of near-critical paths.

## VII. Conclusion

We have proposed an LR formulation that decouples timing-driven incremental layer assignment from the timing engine. The exploitation of flow conservation conditions was the key to enabling the use of an external signoff timing engine. The accurate slack values from the signoff engine provided appropriate guidance toward timing closure, while the linear timing models (for delay and slew) employed for the min-cost flow allowed for fast layer reassignment. That is why the experimental results showed by the proposed technique can consistently trade a longer runtime for a smaller number of timing violations. As compared to two state-of-the-art methods, the proposed technique led to roughly 50% less timing violation. Besides, the edge pruning strategy made possible to shrink the problem size while reducing the number of used vias. We also concluded experimentally that the pessimism introduced by a simplified timer guides the optimization engine off critical paths, leaving behind roughly 40% of optimization opportunity. As a future work, we plan to improve the accuracy of the cost linearization through on-the-fly calibration with the signoff timing engine. Another possibility of future direction is to introduce a control mechanism to reduce antenna violations. During the network flow graph construction, the segments with long antenna can be constrained to be assigned only to upper layers in order to release the accumulated charges and thus reduce antenna violations. This can be accommodated during the pruning step of the proposed framework.

## References

[1] Y. Wei *et al.*, "CATALYST: Planning layer directives for effective design closure," in *Proc. DATE*, Grenoble, France, 2013, pp. 1873–1878.
[2] C. J. Alpert *et al.*, "What makes a design difficult to route," in *Proc. ISPD*, San Francisco, CA, USA, 2010, pp. 7–12.
[3] S. Hu, Z. Li, and C. J. Alpert, "A faster approximation scheme for timing driven minimum cost layer assignment," in *Proc. ISPD*, San Diego, CA, USA, 2009, pp. 167–174.
[4] B. Yu, D. Liu, S. Chowdhury, and D. Z. Pan, "TILA: Timing-driven incremental layer assignment," in *Proc. ICCAD*, Austin, TX, USA, 2015, pp. 110–117.
[5] R. Puri, D. S. Kung, and A. D. Drumm, "Fast and accurate wire delay estimation for physical synthesis of large ASICs," in *Proc. GLSVLSI*, New York, NY, USA, 2002, pp. 30–36.
[6] A. B. Kahng, S. Kang, H. Lee, I. L. Markov, and P. Thapar, "High-performance gate sizing with a signoff timer," in *Proc. ICCAD*, San Jose, CA, USA, 2013, pp. 450–457.
[7] T. J. Reimann, C. C. N. Sze, and R. Reis, "Cell selection for high-performance designs in an industrial design flow," in *Proc. ISPD*, Santa Rosa, CA, USA, 2016, pp. 65–72.
[8] S. Dong, J. Ao, and F. Luo, "Delay-driven and antenna-aware layer assignment in global routing under multitier interconnect structure," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 5, pp. 740–752, May 2015.
[9] D. Liu, B. Yu, S. Chowdhury, and D. Z. Pan, "Incremental layer assignment for critical path timing," in *Proc. DAC*, Austin, TX, USA, 2016, pp. 1–6.
[10] P. Saxena and C. L. Liu, "Optimization of the maximum delay of global interconnects during layer assignment," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 4, pp. 503–515, Apr. 2001.
[11] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan, "ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite," in *Proc. ICCAD*, Austin, TX, USA, 2015, pp. 921–926.
[12] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "GRIP: Global routing via integer programming," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 1, pp. 72–84, Jan. 2011.
[13] S. Held, D. Müller, D. Rotter, V. Traub, and J. Vygen, "Global routing with inherent static timing constraints," in *Proc. ICCAD*, Austin, TX, USA, 2015, pp. 102–109.
[14] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "BoxRouter 2.0: Architecture and implementation of a hybrid and robust global router," in *Proc. ICCAD*, San Jose, CA, USA, 2007, pp. 503–508.
[15] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "NCTU-GR 2.0: Multithreaded collision-aware global routing with bounded-length maze routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 5, pp. 709–722, May 2013.
[16] W.-H. Liu and Y.-L. Li, "Negotiation-based layer assignment for via count and via overflow minimization," in *Proc. ASP-DAC*, Yokohama, Japan, 2011, pp. 539–544.
[17] D. Wu, J. Hu, and R. Mahapatra, "Coupling aware timing optimization and antenna avoidance in layer assignment," in *Proc. ISPD*, San Francisco, CA, USA, 2005, pp. 20–27.
[18] Z. Li *et al.*, "Fast interconnect synthesis with layer assignment," in *Proc. ISPD*, Portland, OR, USA, 2008, pp. 71–77.
[19] S. Hu, Z. Li, and C. J. Alpert, "A polynomial time approximation scheme for timing constrained minimum cost layer assignment," in *Proc. ICCAD*, San Jose, CA, USA, 2008, pp. 112–115.
[20] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Amsterdam, The Netherlands: Springer, 2011.
[21] C. Guth *et al.*, "Timing-driven placement based on dynamic net-weighting for efficient slack histogram compression," in *Proc. ISPD*, Monterey, CA, USA, 2015, pp. 141–148.
[22] M. M. Ozdal, S. Burns, and J. Hu, "Algorithms for gate sizing and device parameter selection for high-performance designs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 10, pp. 1558–1571, Oct. 2012.
[23] V. S. Livramento, C. Guth, J. L. Güntzel, and M. O. Johann, "A hybrid technique for discrete gate sizing based on Lagrangian relaxation," *ACM Trans. Design Autom. Electron. Syst.*, vol. 19, no. 4, 2014, Art. no. 40.
[24] C.-P. Chen, C. C. N. Chu, and D. F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 7, pp. 1014–1025, Jul. 1999.
[25] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[26] S.-S. Han, A. B. Kahng, S. Nath, and A. S. Vydyanathan, "A deep learning methodology to proliferate golden signoff timing," in *Proc. DATE*, Dresden, Germany, 2014, pp. 1–6.
[27] M. Bazaraa, J. Jarvis, and H. Sherali, *Linear Programming and Network Flows*. Hoboken, NJ, USA: Wiley, 2011.
[28] Z. Király and P. Kovács, "Efficient implementations of minimum-cost flow algorithms," *arXiv preprint arXiv:1207.6381*, 2012.
[29] Y. Zhang and D. Z. Pan, "Timing-driven, over-the-block rectilinear Steiner tree construction with pre-buffering and slew constraints," in *Proc. ISPD*, Petaluma, CA, USA, 2014, pp. 29–36.
[30] G. Flach, T. Reimann, G. Posser, M. Johann, and R. Reis, "Effective method for simultaneous gate sizing and *V*th assignment using Lagrangian relaxation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 4, pp. 546–557, Apr. 2014.
[31] *LEMON: Library for Efficient Modeling and Optimization in Networks*. Accessed on Mar. 2016. [Online]. Available: https://lemon.cs.elte.hu/

**Vinicius Livramento** received the B.Sc. and M.Sc. degrees in computer science from the Federal University of Santa Catarina, Florianópolis, Brazil, in 2008 and 2013, respectively, where he is currently pursuing the Ph.D. degree in automation and systems engineering.

He was a Visiting Scholar with the University of Texas at Austin, Austin, TX, USA, in 2016. His current research interests include timing and power optimization during physical design, such as placement, gate sizing, and routing.

Mr. Livramento was a recipient of the First Place in the ISPD 2013 Discrete Gate Sizing Contest and the ICCAD 2015 Incremental Timing-Driven Placement Contest.

**Derong Liu** received the B.S. degree in microelectronics from Fudan University, Shanghai, China, in 2011. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA, under the supervision of Prof. D. Z. Pan.

Her current research interests include physical design and design automation for logic synthesis.

**Salim Chowdhury** received the Ph.D. degree from the University of Southern California, Los Angeles, CA, USA, in 1986.

He taught at the University of Iowa, Iowa City, IA, USA, for some years, where he obtained multiple research grants from the National Science Foundation. He had been researching with semiconductor industries since then, with Motorola, Schaumburg, IL, USA, Sun Microsystem, Santa Clara, CA, USA, and until recently with Oracle America, Austin, TX, USA. He holds several patents and publications.

Dr. Chowdhury was a recipient of the Best Paper Award from DAC.

**Bei Yu** (S'11–M'14) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Assistant Professor with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong.

Dr. Yu was a recipient of three Best Paper Awards at the SPIE Advanced Lithography Conference 2016, the International Conference on Computer Aided Design (ICCAD) 2013, and the Asia and South Pacific Design Automation Conference (ASPDAC) 2012, three other Best Paper Award Nominations at the Design Automation Conference (DAC) 2014, ASPDAC 2013, and ICCAD 2011, and three ICCAD Contest Awards in 2015, 2013, and 2012. He has served in the Editorial Boards of the *Integration, the VLSI Journal* and *IET Cyber-Physical Systems: Theory and Applications*.

**Xiaoqing Xu** (S'15) received the B.S. degree in microelectronics from Peking University, Beijing, China, in 2012. He is currently pursuing the Ph.D. degree in electrical and computer engineering, University of Texas at Austin, Austin, TX, USA, under the supervision of Prof. D. Z. Pan.

His current research interests include robust standard cell design, design for manufacturability, and physical design in advanced technology nodes.

Mr. Xu was a recipient of the Gold Medal for ACM Student Research Competition in the International Conference on Computer Aided Design 2016, the SRC Best in Session Award in the SRC TECHCON 2015, and the Excellent Graduate from Peking University in 2012.

**David Z. Pan** (S'97–M'00–SM'06–F'14) received the Ph.D. degree from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 2000.

He was a Research Staff Member with IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, from 2000 to 2003. He is an Engineering Foundation Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin (UT Austin), Austin, TX, USA. He has published over 250 papers and holds eight U.S. patents. His current research interests include cross-layer design for manufacturability, reliability, security, physical design, and computer-aided design for emerging technologies.

Dr. Pan was a recipient of numerous awards, including the SRC 2013 Technical Excellence Award, the DAC Top 10 Author in Fifth Decade, the DAC Prolific Author Award, the ASP-DAC Frequently Cited Author Award, 13 Best Paper Awards, the Communications of the ACM Research Highlights in 2014, the ACM/SIGDA Outstanding New Faculty Award in 2005, the NSF CAREER Award in 2007, the SRC Inventor Recognition Award (thrice), the IBM Faculty Award (four times), the UCLA Engineering Distinguished Young Alumnus Award in 2009, the UT Austin RAISE Faculty Excellence Award in 2014, and several international CAD contest awards.

**José Luís Güntzel** (M'11) received the Electrical Engineering degree and the M.Sc. and Doctorate degrees in computer science from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 1990, 1993, and 2000, respectively.

He spent a one-year doctoral stage with the Laboratory of Informatics, Robotics and Microelectronics of Montpellier, Montpellier, France, in 1996. Since 2007, he has been a Professor with the Federal University of Santa Catarina, Florianópolis, Brazil. His current research interests include physical synthesis of very large-scale integration (VLSI) systems, VLSI architectures for high-resolution video coding, and design of energy-efficient systems-on-chip.

**Luiz C. V. dos Santos** received the doctoral degree from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 1998.

He was with the Design Automation Section of the Information and Communication Systems Group, Eindhoven University of Technology. He is a Professor with the Federal University of Santa Catarina, Florianópolis, Brazil. His current research interests include algorithms for EDA, ranging from physical to high-level design.