Timing-Driven Placement Optimization Facilitated by Timing-Compatibility Flip-Flop Clustering

Dimitrios Mangiras, Apostolos Stefanidis, Ioannis Seitanidis, Chrysostomos Nicopoulos, and Giorgos Dimitrakopoulos

***Abstract*—Timing-driven placement optimization is applied incrementally in various parts of the flow, together with other timing optimization techniques, to achieve timing closure. In this article, we present a generalized approach for Lagrange-relaxation-based timing optimization that is used to iteratively relocate gates, flip-flops, and local clock buffers (LCBs), with the goal being to reduce the timing violations. Cells are allowed to move within an appropriately positioned search window, the location of which is decided by force-like timing vectors covering both late and early timing violations. The magnitude of these timing vectors is determined by the value of the corresponding Lagrange multipliers. The introduced placement optimization is applied in conjunction with a newly proposed flip-flop clustering algorithm that (re)assigns flip-flops to LCBs, to separate flip-flops with incompatible timing profiles and to facilitate the subsequent timing-optimization steps. The proposed approach is tested on the ICCAD-2015 benchmarks, providing the best overall results when compared to state-of-the-art timing-driven placement techniques.**

***Index Terms*—Clustering, Lagrange relaxation (LR), placement, timing optimization.**

I. Introduction

TIMING closure is a complex process that involves many iterative optimization steps applied in various phases of the physical design flow [1], [2]. Placement is instrumental to the performance of the overall flow, since it determines the length of the wires and their congestion in certain regions of the design. Long wires suffer from increased *RC* delay (*R*: Resistance, *C*: Capacitance), while wire congestion may lead to routing critical nets on nonminimum-distance paths to avoid congested regions. Over the last several years, wire *RC* delay has not only accounted for the lion's share of the total delay by far, but its variation across the metal stack has also increased dramatically [3]. Such critical factors have significantly increased the importance of timing-driven placement, which is required to reduce the timing violations within a reasonable runtime, even for very large designs.

During global placement and cell spreading, the timing is optimized by controlling the wire length of selected nets, or by trying to smooth the physical layout of timing-critical paths [4]–[6]. The incremental timing-driven placement steps that follow global placement try to move cells to appropriate locations, to improve timing, with minimal disturbance to the initial placement. In [7], a linear program is utilized to minimize the weighted wire length on critical paths, where the path-delay sensitivities are used as weights. To avoid noncritical paths becoming critical, a novel criticality-adjacency network concept is presented. The work of [8] presents new sensitivity and figure-of-merit functions to guide cell relocation, while, in [9], net weights are computed using a critical-path counting algorithm (the more paths a net affects, the larger its weight). In [10], a differential timing model for moving timing-critical cells is adopted. The validity of the timing model is maintained by constraining the placement changes.

The recently introduced early histogram compression (EHC) [11] technique mitigates hold timing violations through reassignments of local clock buffers (LCBs) to flip-flops and appropriate LCB movements. Better results are achieved in [12], which optimizes the clock arrival at each flip-flop by appropriate reassignments of LCBs to flip-flops and flip-flop movements, to improve hold violations while preserving the preoptimized setup violations. In contrast to such approaches, OWARU [13] utilizes Bézier curves to smoothen the physical curve produced by the placement of the cells participating in timing-critical paths. Other approaches like [14] and [15] rely on analytic formulations for relocating flip-flops, gates, and LCBs, or utilize noncritical cell relocation and cell-swapping to improve the quality-of-results (QoRs).

Other approaches rely on the Lagrange relaxation (LR)-based formulation of timing optimization, whereby the derived cost function guides cell relocation to reduce timing violations [5], [6], [16], [17]. Using LR, the hard constraints of the optimization are removed and incorporated into the objective function, each one multiplied by a penalty term called a Lagrange multiplier (LM). During optimization, LMs act as dynamic weights that reflect both the timing criticality of each net and the number of critical endpoints that it affects.

In this article, we focus on incremental timing-driven placement, with the goal to fix the placement of timing-critical cells and improve overall timing. As opposed to previous methods

Manuscript received April 24, 2019; revised July 26, 2019; accepted August 31, 2019. Date of publication September 17, 2019; date of current version September 18, 2020. The work of D. Mangiras was supported by the Onassis Foundation under Scholarship G ZO 014-1/2018-2019. This article was recommended by Associate Editor I. H.-R. Jiang. *(Corresponding author: Giorgos Dimitrakopoulos.)*

D. Mangiras, A. Stefanidis, and G. Dimitrakopoulos are with the Department of Electrical and Computer Engineering, Democritus University of Thrace, 67100 Xanthi, Greece (e-mail: dmangira@ee.duth.gr; apstefan@ee.duth.gr; dimitrak@ee.duth.gr).

I. Seitanidis is with Mentor, a Siemens Business, 38100 Grenoble, France (e-mail: ioannis_seitanidis@mentor.com).

C. Nicopoulos is with the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia 1678, Cyprus (e-mail: nicopoulos@ucy.ac.cy).

Digital Object Identifier 10.1109/TCAD.2019.2942001

that independently move combinational gates, flip-flops, and/or LCBs using loosely connected algorithms, we propose, for the first time (to the best of our knowledge), an LR-based timing-driven placement algorithm that *handles the relocation of all types of cells in a unified manner*. Each timing-critical cell is relocated iteratively through the selection of an optimized position out of a set of appropriately selected candidate positions. Both the selection of the best position, and the definition of the search window, are based on the value of the LMs in each optimization round.

The proposed LR-based placement optimization methodology is complemented by a pseudo-3-D flip-flop clustering algorithm that clusters flip-flops according to their geographical location and the timing slacks of their D and Q pins. The objective is to guarantee that flip-flops of the same cluster share a compatible timing profile (i.e., the flip-flops should benefit in the same way by an increase, or an equivalent decrease, in the clock arrival time). In this way, nearby timing-compatible flip-flops of the same cluster can be driven by the same LCB, while timing-incompatible flip-flops are driven by different LCBs, even if they are placed in the same region. This separation of timing-incompatible flip-flops facilitates the timing optimizations performed later on by LR-based cell relocation.

The proposed approach effectively combines the application of the following processes to yield very promising results.

1) A proposed LR-based formulation for timing optimization that allows the handling of gates, flip-flops, and LCBs in a unified manner.
2) LM-based calibration of the search window to detect candidate placements for each cell.
3) A flip-flop clustering step that clusters flip-flops unevenly, based on their timing profile.

The entire methodology is implemented in C++ inside the RSyn physical design framework [18] and tested against the benchmarks of the ICCAD 2015 contest [19]. The derived results indicate significant improvements in worst negative slack (WNS) and total negative slack (TNS) at a reasonable runtime, as compared to state-of-the-art timing-driven placement-optimization techniques that include either closely related LR-based optimization [16], or other highly efficient heuristics [11]–[13].

The rest of this article is organized as follows: Section II presents the timing-compatibility clustering method. Section III introduces the proposed LR-based formulation for timing optimization. Section IV outlines the overall placement optimization flow. Section V presents the formation of the search window that contains the candidate positions for each cell. The experimental results are presented and analyzed in Section VI. Finally, Section VII concludes this article.

II. Timing-Compatibility Flip-Flop Clustering

The proposed timing-driven placement methodology relocates the combinational gates, flip-flops, and LCBs in a unified manner, to improve the circuit's timing. Moving LCBs relative to the group of flip-flops that they drive, increases or decreases the clock arrival time. This change in clock arrival may be beneficial to some flip-flops of the group and harmful to other flip-flops in the same group. For example, delaying clock arrival would benefit flip-flops with negative D/positive Q late slack, and hurt the timing of flip-flops with a positive D/negative Q late slack profile. Therefore, before applying any LCB movement, we need to be sure that each LCB drives flops with *compatible* timing profiles (i.e., all need an increase or reduction in clock arrival time, or are neutral to this choice). To achieve this, we use a pseudo-3-D clustering algorithm, where flip-flops are clustered according to their (x, y) position and their timing profile. Flip-flops placed in the same cluster are driven by the same LCB, while timing-incompatible flip-flops are put in different clusters and driven by different LCBs.

Algorithm 1: Timing-Compatibility FF Clustering

```
1  Assign a timing profile to each flip-flop;
2  InitClusters();
3  FF_PriorityList ← PrioritizeFlipFlops();
4  repeat // FF-to-cluster assignment
5      foreach FF i in FF_PriorityList do
6          if i∈fast then
7              C_cand ← {clusters with fast or neutral FFs};
8          else if i∈slow then
9              C_cand ←{clusters with slow or neutral FFs};
10         else
11             C_cand ← {all clusters};
12         end
13         best_cost ← inf;
14         foreach cluster j in C_cand do
15             if size(j)!=MaxSize & canAssign(i,j) then
16                 cost ← AssignmentCost(i,j);
17                 if (cost < best_cost) then
18                     best_cost ← cost;
19                     assign FF i to cluster j;
20                 end
21             end
22         end
23     end
24     UpdateClusterCenter();
25     UpdateTiming();
26     UpdateFFTimingProfiles();
27 until convergence;
```

The proposed clustering algorithm is given in Algorithm 1. Initially, each flip-flop is given a timing profile that can belong to one of three categories depending on how the clock arrival time would benefit the flip-flops' timing: 1) faster clock arrival (fast); 2) slower clock arrival (slow); and 3) neutral. Any finer-grained categorization is possible. Neutral is considered compatible with both the fast and slow categories, while flip-flops that belong to the fast and slow categories are incompatible. The list of clusters is then initialized, and the iterative loop of flop-to-cluster assignment and cluster updating is executed until convergence is reached (i.e., either all flip-flops remain attached to their previously assigned clusters, or the maximum number of iterations is reached).

The proposed clustering algorithm is a variant of k-means clustering [20], [21], which minimizes the squared distance between each cluster center and its assigned flip-flops, while also taking into account the timing profile of each flip-flop and the size of each cluster. In this way, each cluster contains

TABLE I
TIMING PROFILE CATEGORIZATION

Early			Late		
Slack at		Timing	Slack at		Timing
D pin	Q pin	profile	D pin	Q pin	profile
+	+	neutral	+	+	neutral
+	−	slow	+	−	fast
−	+	fast	−	+	slow
−	−−	slow	−	−−	fast
−−	−	fast	−−	−	slow

an appropriate number of timing-compatible flip-flops, thereby facilitating the LCB movement that is subsequently applied.

To improve timing, the proposed clustering creates clusters of unequal sizes on purpose. The LCBs that drive flip-flops with fast timing profiles are less loaded relative to the LCBs that drive flip-flops belonging to the slow category. This uneven loading decreases, or increases, accordingly the delay of the LCB with the goal being to improve timing. Creating clusters of uneven sizes—to facilitate timing optimization—with *k*-means is not directly possible. To achieve this goal in a practical manner, we artificially shrink, or expand, the true Euclidean distance, without any further modification to the clustering algorithm.

A. Assign Timing Profile to Each Flip-Flop

Initially, each flip-flop is assigned to one of the three categories shown in Table I for late timing, and, separately, for early timing, after taking into account the timing slacks (positive/+ or negative/–) at the *D* and *Q* pins of each flip-flop. In the case that the slack is negative at both pins of the flip-flop, the timing profile of the flip-flop is determined by the pin with the most negative slack, depicted as two negative minus signs (−−) in Table I.

The separate late and early timing profiles initially given to each flip-flop should be merged to one final profile. When both the timing profiles are same, the final timing profile for this flip-flop is the same common profile. If one of the two timing profiles for a flip-flop is neutral (either for late or early timing), the timing profile for this flip-flop is determined by the other non-neutral timing profile. On the contrary, when a flip-flop belongs to contradicting categories in the late and early timing modes, the timing profile for this flip-flop is the one selected for the most critical mode, i.e., the one with the most negative slack in either the *D* or *Q* pins.

B. Initialize Clusters and Prioritize Flip-Flops

The total number of clusters is set equal to the number of available LCBs, and the cluster centers are initialized to the positions of the corresponding LCBs. If no LCBs are present, the number of clusters can be selected with any other density or maximum fanout/capacitance criterion, and the center of each cluster is set to the position of a randomly selected flip-flop.[1] Each newly created cluster is assigned to one flip-flop to avoid leaving a cluster empty. We try to initialize each cluster with a fast flip-flop (if a fast flip-flop exists nearby), taken from a list of the 20 most closely placed flip-flops. If this is not possible, the cluster is assigned to a neutral flip-flop to increase the freedom of later assignments. If, however, all nearby flip-flops are slow, the cluster is assigned to a slow flop. In this way, we increase the probability to initialize more fast clusters, implicitly reducing the average size of a fast cluster relative to neutral or slow clusters.

Next, we prioritize the list of flip-flops, so the flip-flops with timing violations select a nearby cluster first. If the number of flip-flops with a fast timing profile is larger than the flip-flops with a slow timing profile, we first examine the fast flip-flops (those flip-flops are put at the top of the *FF_PriorityList*) and then the slow flip-flops. In the opposite case (number of slow flip-flops exceeds the fast flip-flops), we first examine the flip-flops that belong to the slow timing profile. Neutral flip-flops are always examined last.

C. Flip-Flop Clustering

The assignment of flip-flops to clusters is outlined in lines 4–27 of Algorithm 1. For each flip-flop examined, we identify which clusters are considered as valid assignment candidates. For instance, a flip-flop with a fast timing profile considers only the clusters that contain fast or neutral flip-flops as valid. The flip-flops with a neutral timing profile are compatible with all other flip-flops and can be assigned to any cluster. When examining the assignment of flip-flop *i* to cluster *j*, we first check—with the condition in line 15—that no cluster receives more flip-flops than the maximum allowed, and if this assignment would cause flip-flop *i* to be incompatible with the flip-flops already assigned to cluster *j*. The potential incompatibility may arise due to the new wire and LCB delays. These new delays may, in fact, cause marginal, or potentially more notable, alterations to the clock arrival times, which, in turn, may possibly make the timing profiles of the flip-flops outdated. To accurately reflect the new clock arrival times, incremental timing analysis should be performed, but this is prohibitively expensive when checking each independent assignment. Instead, we focus this analysis only on "weak" flops that are most likely to switch to a different timing profile after the changes in the clock arrival times, i.e., flops with timing slacks in their *D/Q* pins in the range of ±50 ps. To quickly estimate the timing impact of assigning flip-flop *i* to cluster *j*, we use the differential timing model of [10] in a manner similar to [16].

This dynamic checking of the timing profiles can be disabled to significantly reduce the overall runtime. As will be demonstrated in Section VI, omission of this step has a minimal impact on the resulting timing QoR. The reason is because flip-flop to cluster reassignment rarely affects the timing profile of the flip-flops with *significant* timing slacks; it may only disturb the weak flip-flops. Therefore, the flip-flops with significant timing slacks would be correctly separated, even if dynamic compatibility checking is disabled. The weak flops with the outdated timing profiles would eventually move to the neutral category via the subsequent timing optimizations, thus becoming compatible with all other categories.

[1]The optimal number of clusters, *k*, for this timing-compatibility clustering is not obvious, and it can only be judged by the final timing QoR obtained after executing the entire timing-driven placement flow for various numbers of clusters. Predicting *a-priori* the optimal *k* is planned as future work.

Algorithm 2: AssignmentCost(FlipFlop f, Cluster c)

$dist(f, c) \leftarrow \left((c.x - f.x)^2 + (c.y - f.y)^2\right)^{1/2}$;
$w \leftarrow 1$;
if $f \in$ *neutral and c has slow or only neutral FFs* **then**
  $w \leftarrow \frac{size(c)}{MaxSize}$;
else if $f \in$ *neutral and c has fast FFs* **then**
  $w \leftarrow 1 + \frac{size(c)}{MaxSize}$;
end
return $w \cdot dist(f, c)$;

From all valid and compatible clusters, each flip-flop is assigned to the cluster that minimizes the assignment cost computed according to Algorithm 2. The cost is the Euclidean distance between flip-flop f and the center of cluster c, multiplied by a weight w. The role of w is to create clusters with unequal sizes, to further improve timing: clusters with fast/neutral flip-flops should contain fewer flip-flops than clusters with slow/neutral flip-flops. The distance between a neutral flip-flop and a cluster that contains slow, or only neutral, flip-flops is scaled down by w, thus increasing the probability that the neutral flip-flop is assigned to this cluster. In contrast, the distance of a neutral flip-flop to a cluster with fast flip-flops is artificially increased, thus increasing the cost of this assignment and making the assignment less favorable. In this way, the clusters with fast flip-flops remain with fewer flip-flops in total. Weight w only scales the distance of neutral flip-flops, while w is always equal to one for flip-flops of the fast or slow categories.

When considering flip-flops of different sizes with different clock-pin capacitances, the contribution of each flip-flop to the size of the cluster should be measured relative to its clock-pin capacitance. The larger the clock-pin capacitance, the larger the equivalent count in terms of primitive flip-flops. Hence, the size of each cluster would reflect the total capacitance driven by the LCB of the cluster. The proposed algorithm will end up with producing clusters of fast flip-flops with less total capacitance relative to clusters of slow or neutral, flip-flops.

D. Update Cluster Centers and Timing Profiles

When all flip-flops are assigned to a cluster, we need to update the center of each cluster (line 24 in Algorithm 1) to the center of gravity of the flip-flops belonging to this cluster. Each LCB moves to the center of the cluster to which it belongs. As the centers of the clusters are moved to new updated positions, the LCBs are also moved and instantly legalized to their new positions using the Jezz legalizer [22] built into Rsyn [18]. If LCB movement is limited by a maximum displacement constraint, and the updated cluster center lies outside the maximum displacement bounding box of the LCB, the center of the cluster is slid toward the nearest location at the bounds of the displacement bounding box. In this way, in every iteration, the location of the cluster center remains valid in terms of detailed placement constraints.

When the dynamic update of timing profiles is enabled, an incremental static timing analysis is performed after moving the LCBs to their new positions, to update the slacks and the timing profiles of the flip-flops (lines 25 and 26 of Algorithm 1). Therefore, the new—and accurate—timing profile of each flip-flop is available for the next iteration.

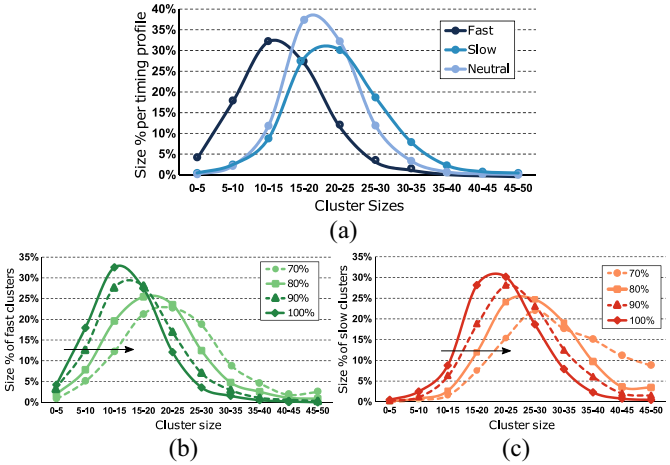

Fig. 1. (a) Percentage of cluster sizes for each timing profile (fast, slow, neutral) for the representative sb10 benchmark of the ICCAD-2015 benchmark set [19]. The percentage of cluster sizes for (b) only fast, and (c) only slow clusters, as the number of cluster centers is artificially decreased.

E. Clustering Behavior

To demonstrate the efficacy of the clustering technique with respect to the generation of clusters of uneven sizes, we analyze the clustering behavior in a representative benchmark of the ICCAD-2015 benchmark set [19]. Specifically, Fig. 1(a) depicts the percentage of cluster sizes for each of the three flip-flop timing profiles (fast, slow, and neutral) for sb10. As shown in the figure, more than 30% of the fast clusters have sizes between 10 and 15 flip-flops, while an additional cumulative 20% corresponds to even smaller clusters of 0–5 and 5–10 flip-flops. In contrast, the majority of clusters of slow and neutral-only flip-flops have larger sizes; around 30% of all slow clusters have sizes of 20–25 flops. Note that the similar trends are observed in all examined benchmarks.

More importantly, the behavior of the proposed clustering technique is robust with respect to the total number of clusters. Fig. 1(b) and (c) illustrates the percentage of cluster sizes for the fast and slow categories, respectively, of the sb10 benchmark, as the total number of clusters, k, decreases. Specifically, the number of clusters decreases from 100% (i.e., the number of clusters in the unmodified benchmark) down to 70% of the initial number. The decrease in clusters is achieved by uniformly removing—while accounting for any disparities in cluster densities—a corresponding number of clusters from the benchmark. As k decreases, both the fast and slow distributions shift to the right, i.e., toward larger cluster sizes. Nevertheless, if we juxtapose the fast and slow distributions, one can clearly see that, even as k decreases, the fast clusters are still smaller, on average, than the corresponding slow clusters. This behavior unequivocally demonstrates that the proposed flip-flop clustering achieves the desired objective, irrespective of the number of available clusters.

III. LR-Based Timing Optimization

Our goal is to minimize the sum of early and late TNS, by appropriately relocating the timing-critical cells of the design.

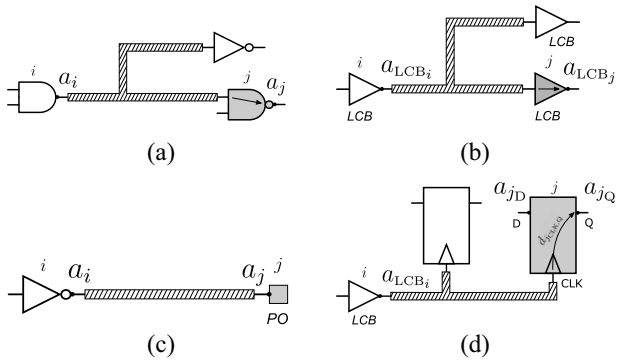

Fig. 2. Definition of arrival times and delays $d_{i,j}$, including both wire- and arc-delay, for different types of cells. (a) Combinational gates. (b) LCBs. (c) Primary outputs (POs). (d) Flip-flops (FFs).

Assuming that TNS is computed over the set $\mathcal{E}$ of all the timing endpoints, including primary outputs POs and the input-D pins of the flip-flops (FFs) [23], the timing optimization problem can be stated as follows:

$$
\begin{aligned}
\textbf{min:}\quad & \sum_{j\in\mathcal{E}}\left(-s_j^L\right)+\sum_{j\in\mathcal{E}}\left(-s_j^E\right)\\
\textbf{s.t.:}\quad & s_j^L\le 0,\quad s_j^E\le 0\\
& \forall \text{ timing endpoint } j\in\mathcal{E}\\
& s_j^L\le T-a_j^L,\quad s_j^E\le a_j^E\\
& \forall \text{ output } j\in \text{POs}\\
& s_j^L\le r_{j_D}^L-a_{j_D}^L,\quad s_j^E\le a_{j_D}^E-r_{j_D}^E\\
& \forall \text{ input } D \text{ pin of flip-flop } j\in\text{FFs}\\
& a_i^L+d_{i,j}^L\le a_j^L,\quad a_i^E+d_{i,j}^E\ge a_j^E,\\
& \forall \text{ gate } j\in\text{Gates [see Fig. 2(a)]}\\
& a_{\mathrm{LCB}_i}^L+d_{i,j}^L\le a_{j_Q}^L,\quad a_{\mathrm{LCB}_i}^E+d_{i,j}^E\ge a_{j_Q}^E\\
& \forall \text{ output } Q \text{ pin of flip-flop } j\in\text{FFs [see Fig. 2(d)]}\\
& a_{\mathrm{LCB}_i}^L+d_{i,j}^L\le a_{\mathrm{LCB}_j}^L,\quad a_{\mathrm{LCB}_i}^E+d_{i,j}^E\ge a_{\mathrm{LCB}_j}^E\\
& \forall \text{ LCB } j\in\text{LCBs [see Fig. 2(b)]}\\
& \text{Cell placement is legalized}\\
& \text{Cell relocation distance} \le \text{maximum displacement.}
\end{aligned}
\tag{1}
$$

Variables s_j^L and s_j^E represent the negative slack at pin $j\in\mathcal{E}$ for late and early timing, while T is the targeted clock period. The timing slack at each timing endpoint is the difference between the required arrival time and the actual arrival time. Parameters a_j and a_{LCBj} are the arrival times (late or early) at the output pins of a gate and an LCB, respectively, while a_{j_D} and a_{j_Q} represent the arrival times at the D/Q pins of flip-flop j. The required arrival time at the D pin of the same flip-flop is expressed as r_{j_D}. The delay $d_{i,j}$ is the sum of the wire and the cell delay from the output pin of cell i to the output pin of cell j. Fig. 2 illustrates in detail the timing arcs involved in the computation of $d_{i,j}$ for every type of cell. In the case of gates and LCBs, shown in Fig. 2(a) and (b), $d_{i,j}$ comprises the wire delay from the output pin of the driving cell of the previous level plus the delay of the jth gate or LCB. When pin j represents a timing endpoint like a primary output [Fig. 2(c)], $d_{i,j}$ is only the corresponding wire delay. Similarly, as shown in Fig. 2(d), in the case of a flip-flop, $d_{i,j}$ is the wire delay from the output pin of the LCB that drives the clock pin of this flip-flop plus the clock-to-Q delay of the flip-flop. Representing with $\mathcal{I}_j$ the set of fanin cells of cell j, and with λ the LMs, LR can incorporate the timing constraints of (1) into the objective as follows:

$$
\begin{aligned}
\textbf{min}:\ & \sum_{j\in\mathcal{E}}\left(-s_j^L\right)+\sum_{j\in\mathcal{E}}\left(-s_j^E\right)+\sum_{j\in\mathcal{E}}\lambda_{0j}^L s_j^L+\sum_{j\in\mathcal{E}}\lambda_{0j}^E s_j^E\\
&+\sum_{j\in\text{POs}}\lambda_{\text{PO}_j}^L\left(s_j^L-T+a_j^L\right)+\sum_{j\in\text{POs}}\lambda_{\text{PO}_j}^E\left(s_j^E-a_j^E\right)\\
&+\sum_{j\in\text{FFs}}\lambda_{D_j}^L\left(s_j^L-r_{j_D}^L+a_{j_D}^L\right)+\sum_{j\in\text{FFs}}\lambda_{D_j}^E\left(s_j^E-a_{j_D}^E+r_{j_D}^E\right)\\
&+\sum_{j\in\text{Gates}}\left(\sum_{i\in\mathcal{I}_j}\lambda_{G_{i,j}}^L\left(a_i^L+d_{i,j}^L-a_j^L\right)\right.\\
&\qquad\left.+\sum_{i\in\mathcal{I}_j}\lambda_{G_{i,j}}^E\left(a_j^E-a_i^E-d_{i,j}^E\right)\right)\\
&+\sum_{j\in\text{FFs}}\left(\lambda_{\text{FF}_{i,j}}^L\left(a_{\mathrm{LCB}_i}^L+d_{i,j}^L-a_{j_Q}^L\right)\right.\\
&\qquad\left.+\lambda_{\text{FF}_{i,j}}^E\left(a_{j_Q}^E-a_{\mathrm{LCB}_i}^E-d_{i,j}^E\right)\right)\\
&+\sum_{j\in\text{LCBs}}\left(\lambda_{\mathrm{LCB}_{i,j}}^L\left(a_{\mathrm{LCB}_i}^L+d_{i,j}^L-a_{\mathrm{LCB}_j}^L\right)\right.\\
&\qquad\left.+\lambda_{\mathrm{LCB}_{i,j}}^E\left(a_{\mathrm{LCB}_j}^E-a_{\mathrm{LCB}_i}^E-d_{i,j}^E\right)\right).
\end{aligned}
\tag{2}
$$

The Karush–Kuhn–Tucker (KKT) optimality conditions for the timing endpoints of the design impose that $\lambda_{0j}^L+\lambda_{\text{PO}j}^L=1$ and $\lambda_{0j}^E+\lambda_{\text{PO}j}^E=1$, for each primary output $j\in$ POs, and $\lambda_{0j}^L+\lambda_{Dj}^L=1$, and $\lambda_{0j}^E+\lambda_{Dj}^E=1$, for the D pin of flip-flop j. Substituting these equalities into (2) causes the slack variables s_j^L and s_j^E to cancel out and simplify the problem as follows:

$$
\begin{aligned}
\textbf{min}:\ & \sum_{j\in\text{POs}}\lambda_{\text{PO}_j}^L\left(-T+a_j^L\right)+\sum_{j\in\text{POs}}\lambda_{\text{PO}_j}^E\left(-a_j^E\right)\\
&+\sum_{j\in\text{FFs}}\lambda_{D_j}^L\left(-r_{j_D}^L+a_{j_D}^L\right)+\sum_{j\in\text{FFs}}\lambda_{D_j}^E\left(-a_{j_D}^E+r_{j_D}^E\right)\\
&+\sum_{j\in\text{Gates}}\left(\sum_{i\in\mathcal{I}_j}\lambda_{G_{i,j}}^L\left(a_i^L+d_{i,j}^L-a_j^L\right)\right.\\
&\qquad\left.+\sum_{i\in\mathcal{I}_j}\lambda_{G_{i,j}}^E\left(a_j^E-a_i^E-d_{i,j}^E\right)\right)\\
&+\sum_{j\in\text{FFs}}\left(\lambda_{\text{FF}_{i,j}}^L\left(a_{\mathrm{LCB}_i}^L+d_{i,j}^L-a_{j_Q}^L\right)\right.\\
&\qquad\left.+\lambda_{\text{FF}_{i,j}}^E\left(a_{j_Q}^E-a_{\mathrm{LCB}_i}^E-d_{i,j}^E\right)\right)\\
&+\sum_{j\in\text{LCBs}}\left(\lambda_{\mathrm{LCB}_{i,j}}^L\left(a_{\mathrm{LCB}_i}^L+d_{i,j}^L-a_{\mathrm{LCB}_j}^L\right)\right.\\
&\qquad\left.+\lambda_{\mathrm{LCB}_{i,j}}^E\left(a_{\mathrm{LCB}_j}^E-a_{\mathrm{LCB}_i}^E-d_{i,j}^E\right)\right).
\end{aligned}
\tag{3}
$$

Previous work on LR-based timing-driven placement [16], [17] assumed that flip-flops and LCBs remain locked to their positions and cannot move. This over-simplification allows the

removal of the required arrival times from (3), since they remain constant. However, this is not allowed in this article. Our goal is to incorporate the movement of all types of cells, gates, flip-flops, and LCBs in the same LR-based formulation, covering both late and early timing constraints. For this reason, we would like to remove the direct contribution of the clock arrival time and keep only the arrival times on the datapath pins in the optimization problem. For the register-to-register paths, we know that

$$r^L_{j_D} = a^E_{j_{\text{CLK}}} + T - t_{\text{setup}} \text{ and } r^E_{j_D} = a^L_{j_{\text{CLK}}} + t_{\text{hold}} \tag{4}$$

where $a^L_{j_{\text{CLK}}}$ and $a^E_{j_{\text{CLK}}}$ represent the late and early arrival times of the clock on flip-flop j, and t_{setup}, t_{hold} are the setup and hold delays of the flip-flop. The clock arrival time $a_{j_{\text{CLK}}}$ and the arrival time at the output Q pin of a flip-flop a_{j_Q} are connected via the clk-to-Q arc delay $d_{j_{\text{CLK,Q}}}$, i.e., $a^L_{j_{\text{CLK}}} = a^L_{j_Q} - d^L_{j_{\text{CLK,Q}}}$ and $a^E_{j_{\text{CLK}}} = a^E_{j_Q} - d^E_{j_{\text{CLK,Q}}}$. Substituting these equalities into the required arrival times of (4) leads to the following equations:

$$r^L_{j_D} = a^E_{j_Q} - d^E_{j_{\text{CLK,Q}}} + T - t_{\text{setup}} \tag{5}$$

$$r^E_{j_D} = a^L_{j_Q} - d^L_{j_{\text{CLK,Q}}} + t_{\text{hold}}. \tag{6}$$

Using (5) and (6) in the place of the required arrival times r_{j_D} of (3), and considering that T, t_{hold}, t_{setup} remain unchanged during timing optimization, we end up with the following formulation:

$$\begin{aligned}
\textbf{min}: & \sum_{j\in\text{POs}} \lambda^L_{\text{PO}_j}\left(a^L_j\right) + \sum_{j\in\text{POs}} \lambda^E_{\text{PO}_j}\left(-a^E_j\right) \\
& + \sum_{j\in\text{FFs}} \lambda^L_{D_j}\left(-a^E_{j_Q} + d^E_{j_{\text{CLK,Q}}} + a^L_{j_D}\right) \\
& + \sum_{j\in\text{FFs}} \lambda^E_{D_j}\left(-a^E_{j_D} + a^L_{j_Q} - d^L_{j_{\text{CLK,Q}}}\right) \\
& + \sum_{j\in\text{Gates}} \left(\sum_{i\in\mathcal{I}_j} \lambda^L_{G_{i,j}}\left(a^L_i + d^L_{i,j} - a^L_j\right) \right. \\
& \qquad \left. + \sum_{i\in\mathcal{I}_j} \lambda^E_{G_{i,j}}\left(a^E_j - a^E_i - d^E_{i,j}\right) \right) \\
& + \sum_{j\in\text{FFs}} \left(\lambda^L_{\text{FF}_{i,j}}\left(a^L_{\text{LCB}_i} + d^L_{i,j} - a^L_{j_Q}\right) \right. \\
& \qquad \left. + \lambda^E_{\text{FF}_{i,j}}\left(a^E_{j_Q} - a^E_{\text{LCB}_i} - d^E_{i,j}\right) \right) \\
& + \sum_{j\in\text{LCBs}} \left(\lambda^L_{\text{LCB}_{i,j}}\left(a^L_{\text{LCB}_i} + d^L_{i,j} - a^L_{\text{LCB}_j}\right) \right. \\
& \qquad \left. + \lambda^E_{\text{LCB}_{i,j}}\left(a^E_{\text{LCB}_j} - a^E_{\text{LCB}_i} - d^E_{i,j}\right) \right).
\end{aligned} \tag{7}$$

By differentiating (7) with respect to the arrival times, according to the KKT optimality conditions, and representing the set of fanin and fanout cells of cell j with $\mathcal{I}_j$ and $\mathcal{O}_j$, we end up with the following LM flow conservation rules.

1) For each gate $j \in$ Gates connected to other gates and flip-flops at its fanin and fanout cones

$$\sum_{i\in\mathcal{I}_j} \lambda^L_{G_{i,j}} = \sum_{k\in\mathcal{O}_j} \lambda^L_{G_{j,k}} + \sum_{k\in\mathcal{O}_j} \lambda^L_{D_k} + \sum_{k\in\mathcal{O}_j} \lambda^L_{\text{PO}_k} \tag{8}$$

$$\sum_{i\in\mathcal{I}_j} \lambda^E_{G_{i,j}} = \sum_{k\in\mathcal{O}_j} \lambda^E_{G_{j,k}} + \sum_{k\in\mathcal{O}_j} \lambda^E_{D_k} + \sum_{k\in\mathcal{O}_j} \lambda^E_{\text{PO}_k}. \tag{9}$$

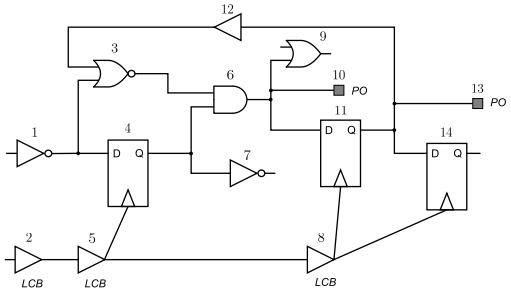

Fig. 3. Example circuit used to illustrate the LM flow conservation rules to preserve the KKT conditions for different types of cells. This figure also highlights the timing arcs involved in the computation of the local cost for each cell relocation in Section IV-A.

2) For each flip-flop $j \in$ FFs connected to other gates or flip-flops at its input D and output Q pins

$$\lambda^E_{D_j} + \sum_{k\in\mathcal{O}_j} \lambda^L_{G_{j,k}} + \sum_{k\in\mathcal{O}_j} \lambda^L_{D_k} + \sum_{k\in\mathcal{O}_j} \lambda^L_{\text{PO}_k} = \lambda^L_{\text{FF}_{i,j}} \tag{10}$$

$$\lambda^L_{D_j} + \sum_{k\in\mathcal{O}_j} \lambda^E_{G_{j,k}} + \sum_{k\in\mathcal{O}_j} \lambda^E_{D_k} + \sum_{k\in\mathcal{O}_j} \lambda^E_{\text{PO}_k} = \lambda^E_{\text{FF}_{i,j}}. \tag{11}$$

3) For each LCB $j \in$ LCBs driving both clock pins of flip-flops or other LCBs

$$\lambda^L_{\text{LCB}_{i,j}} = \sum_{k\in\mathcal{O}_j} \lambda^L_{\text{FF}_{j,k}} + \sum_{k\in\mathcal{O}_j} \lambda^L_{\text{LCB}_{j,k}} \tag{12}$$

$$\lambda^E_{\text{LCB}_{i,j}} = \sum_{k\in\mathcal{O}_j} \lambda^E_{\text{FF}_{j,k}} + \sum_{k\in\mathcal{O}_j} \lambda^E_{\text{LCB}_{j,k}}. \tag{13}$$

4) For each timing endpoint $j \in \mathcal{E}$ driven by pin i

$$\lambda^L_{\text{PO}_j} = \lambda^L_{i,j},\ \lambda^L_{D_j} = \lambda^L_{i,j} \tag{14}$$

$$\lambda^E_{\text{PO}_j} = \lambda^E_{i,j},\ \lambda^E_{D_j} = \lambda^E_{i,j}. \tag{15}$$

For gates, the incoming LMs are distributed to outgoing timing arcs that can be other gates, input D pins of flip-flops, or primary outputs. The same holds for the LMs at the output Q pins of flip-flops, while, on the contrary, the LMs of LCBs are spread to other LCBs, or the clock pin of flip-flops. While the equality constraints for combinational gates have been proven in previous work on LR-based optimization, the optimal relation among LMs on the pins of flip-flops and LCBs, including both late and early timing constraints, are introduced in this article for the first time—to the best of our knowledge—in the open literature. Applying the LM flow conservation equations (8)–(15) to selected cells of Fig. 3 gives the following result.

1) For gate 6 [using (8) and (9)]

$$\lambda^L_{G_{3,6}} + \lambda^L_{G_{4,6}} = \lambda^L_{G_{6,9}} + \lambda^L_{\text{PO}_{10}} + \lambda^L_{D_{11}}$$
$$\lambda^E_{G_{3,6}} + \lambda^E_{G_{4,6}} = \lambda^E_{G_{6,9}} + \lambda^E_{\text{PO}_{10}} + \lambda^E_{D_{11}}.$$

2) For flip-flop 11 [using (10) and (11)]

$$\lambda^E_{D_{11}} + \lambda^L_{G_{11,12}} + \lambda^L_{\text{PO}_{13}} + \lambda^L_{D_{14}} = \lambda^L_{\text{FF}_{8,11}}$$
$$\lambda^L_{D_{11}} + \lambda^E_{G_{11,12}} + \lambda^E_{\text{PO}_{13}} + \lambda^E_{D_{14}} = \lambda^E_{\text{FF}_{8,11}}.$$

3) For LCB 5 [using (12) and (13)]

$$\lambda^L_{\text{LCB}_{2,5}} = \lambda^L_{\text{FF}_{5,4}} + \lambda^L_{\text{LCB}_{5,8}}$$

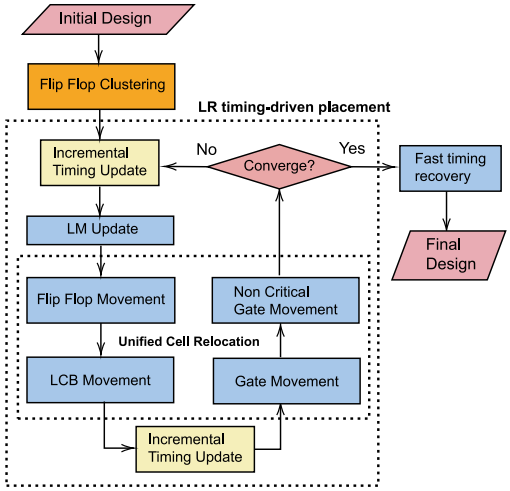

Fig. 4. Overall cell relocation flow. Timing-compatibility flip-flop clustering facilitates the LR-based timing-driven cell relocation that interleaves the LM updates and unified cell relocation until convergence is achieved.

$$\lambda^E_{\mathrm{LCB}_{2,5}} = \lambda^E_{\mathrm{FF}_{5,4}} + \lambda^E_{\mathrm{LCB}_{5,8}}.$$

Substituting the LM equality constraints into the optimization problem, we end up with a simplified objective function that combines the contribution of gates, flip-flops, and LCBs in a unified cost function, for both late and early timing, while it highlights—for the first time—the independent contribution of the clock-to-Q delays and their associated LMs

$$\begin{aligned}\textbf{min}: \quad & \sum_{j\in\mathrm{FFs}} \lambda^L_{D_j}\left(d^E_{j\mathrm{CLK,Q}}\right) + \sum_{j\in\mathrm{FFs}} \lambda^E_{D_j}\left(-d^L_{j\mathrm{CLK,Q}}\right) \\ & + \sum_{j\in\mathrm{Gates}}\left(\sum_{i\in\mathcal{I}_j} \lambda^L_{G_{i,j}}\left(d^L_{i,j}\right) + \sum_{i\in\mathcal{I}_j} \lambda^E_{G_{i,j}}\left(-d^E_{i,j}\right)\right) \\ & + \sum_{j\in\mathrm{FFs}}\left(\lambda^L_{\mathrm{FF}_{i,j}}\left(d^L_{i,j}\right) + \lambda^E_{\mathrm{FF}_{i,j}}\left(-d^E_{i,j}\right)\right) \\ & + \sum_{j\in\mathrm{LCBs}}\left(\lambda^L_{\mathrm{LCB}_{i,j}}\left(d^L_{i,j}\right) + \lambda^E_{\mathrm{LCB}_{i,j}}\left(-d^E_{i,j}\right)\right). \end{aligned} \tag{16}$$

Previous LR-based timing optimizations derived similar cost functions, but of a more limited scope, involving only gates for timing-driven placement (as done in [16] and [17]), or gate sizing (in [24] and [25]), or including gate sizing with local clock skew optimizations for improving the late timing only (in [26] and [27]).

IV. Overall Flow and LR-Based Cell Relocation

The overall flow of applying the LR-based cell relocation process is depicted in Fig. 4. The flip-flop clustering step is executed once at the beginning of the flow, to separate timing-incompatible flip-flops. Then, the iterative LR-based timing-driven placement optimization evolves in two steps. In the first step, assuming constant LMs, we try to move a selected set of cells with the goal of minimizing the cost function (16). In the second step, the LMs are updated to reflect the new criticality of the corresponding timing arcs. On every LM update (and before their initialization), a full incremental timing update takes place.

Algorithm 3: Cell Relocation

```
1  sel_cells←Movable cells with timing violations;
2  sel_cells←sel_cells ∪ Non-critical cells at the immediate
     fanout of sel_cells;
3  foreach cell j ∈ sel_cells in topological order do
4      [best_cost^L,best_cost^E] ← localCost(j);
5      best_location←locationOf(j);
6      cand_pos[j] ← Candidate slots in Search_Window[j];
7      foreach position (x, y) ∈ cand_pos[j] do
8          move cell j to (x, y);
9          update timing locally;
10         [new_cost^L,new_cost^E] ← localCost(j);
11         if (new_cost^L < best_cost^L ) AND
             (new_cost^E < best_cost^E) then
12             best_cost^L ←new_cost^L;
13             best_cost^E ←new_cost^E;
14             best_location← (x, y);
15         end
16     end
17     move cell j to best_location;
18     update timing locally;
19 end
```

In each iteration, all cells are relocated using the procedure described in Algorithm 3, which approximately minimizes (16) by the optimal local relocation (OLR) of one cell at a time, assuming all the other cells are fixed. Flip-flops and LCBs are relocated first, and then gates are traversed in forward topological order from the inputs to the outputs (POs), i.e., OLR of a cell begins after all its fanin gates have been processed. During OLR, each cell is moved conditionally to several candidate locations. For each candidate position examined, timing is updated locally. This update involves building a new Steiner tree for each fanin and fanout net for the cell under relocation using Flute [28] inside RSyn, and recomputing the new delays/slews of the updated nets and all cells connected to those nets, with slew propagation stopping at the immediate fanout of the cell under relocation. Using the new delays computed after the movement, we evaluate the local cost function, as described in the next section. If the new local cost is better than the previous best value, after testing separately the early and the late part of the cost, this candidate location is stored. After trying all the candidate locations, the cell is finally moved to the best stored location.

Every cell movement is made to a legal location using the Jezz legalizer [22] built inside Rsyn [18]. In this way, any disturbance to the neighborhood around the moved cell is directly taken into an account, and, if it degrades timing, it would be handled in the following iterations. Jezz has been used as is, *without* any modifications to prioritize the displacement of timing-critical cells versus noncritical neighbors. Any other legalizer could have readily been used in the place of Jezz.

Contrary to flip-flop clustering, Algorithm 3 does not examine all cells. Only cells with timing violations and noncritical gates (with positive slack) on their fanout are considered. These noncritical gates can change the load of the output net of the critical driver and, therefore, improve its delay. For each

Algorithm 4: localCost(Cell c)

```
[cost^L, cost^E] ← [0,0];
foreach arc i → j of local_arcs of c do
    cost^L ← cost^L + λ^L_{i,j} · d^L_{i,j};
    cost^E ← cost^E + λ^E_{i,j} · (−d^E_{i,j});
    if arc i → j is the clock-to-Q arc of a flip flop then
        cost^L ← cost^L + λ^E_{Dj} · (−d^L_{j_CLK,Q});
        cost^E ← cost^E + λ^L_{Dj} · d^E_{j_CLK,Q};
    end
end
return [cost^L, cost^E];
```

cell, based on its current location, new candidate positions are identified inside an appropriately constructed search window, the size and orientation of which are biased by the local LMs. The formation of this search window is described in Section V.

Before the LM updates and after flip-flop and LCB relocation, the timing is updated incrementally (in a global sense), in order to reflect the new arrival times and the required arrival times that emerges after cell movement.

The iterative optimization stops when a maximum number of iterations is reached, or when TNS stops improving (by more than 1%) for two consecutive iterations. At the end of the main optimization loop, a final brute-force timing recovery step is performed on a list of a few most-critical paths to reduce only early violation.

A. Local Cost Function

To judge the suitability of each candidate location, we compute a local cost, using Algorithm 4, which reflects the local value of the global cost function (16). The local cost involves the summation of the product of the neighbor arc delays and the corresponding LM for late and early timing. The late and early costs are computed separately to enable the cell relocation algorithm to select the best candidate location. The local timing arcs that are included in the calculation of the local cost function involve the timing arcs of the cell under consideration, its immediate fanin and fanout cells, as well as the timing arcs of the cells driven by its fanin cells (side arcs).

For example, in the case of gate 6 shown in Fig. 3, the local arcs used for evaluating the local cost function consist of the timing arcs of the gate itself, ($3 \rightarrow 6$, $4 \rightarrow 6$), the arcs of the cells driving gate 6 ($12 \rightarrow 3$, $1 \rightarrow 3$, $5 \rightarrow 4$), the arcs of the immediate fanout of gate 6 ($6 \rightarrow 9$, $6 \rightarrow 10$, $6 \rightarrow 11$), and the arcs of the cells being driven by the fanins of the gate under consideration ($4 \rightarrow 7$). For an LCB, like LCB 8 in Fig. 3, we consider the timing arcs with respect to other LCBs, or the clock pins of flip-flops: local_arcs = $\{\{5 \rightarrow 8\}, \{2 \rightarrow 5\}, \{8 \rightarrow 11, 8 \rightarrow 14\}, \{5 \rightarrow 4\}\}$. Similarly, in the case of flip-flops, the timing arcs of both D, Q, and clock pins are considered, covering all the local fanin and fanout connections of the flip-flop. For flip-flop 11 in Fig. 3, the local arcs consist of the following set of arcs: local_arcs = $\{\{8 \rightarrow 11, 6 \rightarrow 11\}, \{5 \rightarrow 8, 4 \rightarrow 6, 3 \rightarrow 6\}, \{11 \rightarrow 12, 11 \rightarrow 13, 11 \rightarrow 14\}, \{6 \rightarrow 9, 6 \rightarrow 10, 8 \rightarrow 14\}\}$.

B. Lagrange Multiplier Update

Initially, all LMs are initialized to 1. Then, the LMs for each PO and D pin of a flip-flop, for late and early timing, are updated using the modified subgradient optimization proposed in [16] and [29], at the beginning of each iteration, as follows:

$$\lambda_j^L = \lambda_j^L \left(\frac{a_j^L}{r_j^L} \right), \quad \lambda_j^E = \lambda_j^E \left(\frac{r_j^E}{a_j^E} \right). \tag{17}$$

After updating the output LMs, their values must be distributed to all nets satisfying the flow conservation conditions (8)–(15). The distribution is performed by traversing the circuit in reverse topological order. At each visited cell, the sum of LMs at the output pins are distributed to the LMs of the input pins. When an LM value needs to be distributed to multiple incoming arcs, this distribution is done based on the ratio of the LMs of the corresponding timing arcs. Such distribution increases the LMs on critical paths, and, therefore, the WNS is also expected to be minimized. Also, since LMs are accumulated at each branching point, the higher the number of violating endpoints affected by an arc, the higher the value of the corresponding LM.

The update of the LMs of all internal timing arcs $i \rightarrow j$, $\lambda_{i,j}$, for late and early timing, is done according to (18) and (19), following the method presented in [27]:

$$\lambda_{i,j}^L = \lambda_{i,j}^L \left(1 - \frac{r_j^L - \left(a_i^L + d_{i,j}^L \right)}{T} \right)^K \tag{18}$$

$$\lambda_{i,j}^E = \lambda_{i,j}^E \left(1 - \frac{\left(a_i^E + d_{i,j}^E \right) - r_j^E}{T} \right)^K. \tag{19}$$

The numerator of each fraction is the slack at the output pin of cell j. If the slack is negative, the term in the brackets is greater than 1, thus increasing the corresponding LM. To quickly increase the LM value on timing-critical cells, we empirically set $K = 4$. Smaller values give slower convergence, while larger values do not show further improvement in timing QoR. On the other hand, when the arc is noncritical, the value in the brackets is less than 1, thereby decreasing the LM value. When the slack is positive, we set $K = 1$ to decrease the LMs slowly. This method prevents the criticality of a path from being forgotten immediately after the timing violation at the endpoint is removed, thus avoiding the criticality oscillations, e.g., critical arcs becoming noncritical, and the opposite.

C. Timing Recovery With Flip-Flop-to-LCB Reassignment

The efficiency of the timing-driven placement flow depends on the placement utilization of the design, and on how much space is available for moving cells to their appropriately selected positions with respect to timing. In certain cases, cells do not have much freedom to move, due to nearby placement blockages, or macros. Therefore, when cell relocation has converged—either because cells have reached their maximum displacement limit, or there is limited placement freedom nearby—timing can be improved solely by appropriate LCB-to-flip-flop reassignment. In this context, we examine the 20 most critical flip-flops and test, in a brute-force manner,

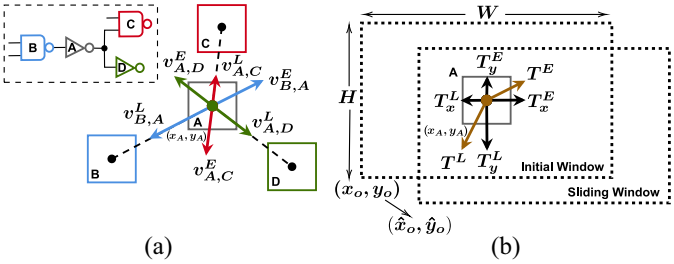

Fig. 5. (a) Two independent vectors (one for late and one for early timing violations) are associated with each fanin and fanout cell; their magnitude reflects their timing criticality. The vectors indicate the desired directions of cell movement that would be beneficial in terms of timing. (b) All late and early vectors are added together to form resultant late and early vectors T^L and T^E. The directions and magnitudes of these resultant vectors determine the orientation of the search window.

whether reconnecting each flip-flop to a different nearby LCB would improve timing or not. Ten nearby LCBs per flip-flop are examined. For each flip-flop-to-LCB reassignment tried, we perform a full incremental timing update, in order to be certain about the expected savings in timing. Each examined flip-flop stays assigned to the LCB that offers the best overall timing. The preferred LCB is the one that reduces TNS on early or late timing, without increasing WNS on the opposite mode, i.e., late or early, respectively.

V. Placement of the Search Window

A critical aspect of the timing optimization process is the identification of appropriate new candidate positions for each cell. In this article, cell movement is facilitated through the use of a *search window*. This window encloses the candidate cell, and it includes all positions that the cell could potentially occupy as part of the optimization process. The proposed methodology, dictating how this search window is positioned around the candidate cell, relies on vectors that indicate desirable cell-movement directions. Specifically, each fanin cell $i \in \mathcal{I}_j$ and each fanout cell $k \in \mathcal{O}_j$ is associated with two independent vectors (one for late and one for early timing violations) on cell j: $v^L_{i,j}$, $v^E_{i,j}$ and $v^L_{j,k}$, $v^E_{j,k}$. These vectors indicate the desired direction of cell movement that would be beneficial in terms of timing. All vectors originate from the center of cell j, and they lay on the imaginary lines connecting the center of cell j to the center of all of its fanin/fanout cells, as depicted in Fig. 5(a).

The magnitude of each vector is equal to the value of the LM between these two cells ($|v^L_{i,j}| = \lambda^L_{i,j}$, $|v^E_{i,j}| = \lambda^E_{i,j}$ and $|v^L_{j,k}| = \lambda^L_{j,k}$, $|v^E_{j,k}| = \lambda^E_{j,k}$). The higher the value of the LM, the more timing-critical the net is, which results in a "stronger" (i.e., of larger magnitude) vector. The LMs are considered ideal proxies, because they encompass the timing criticality of each path, in terms of *all* timing arcs passing through that path.

As illustrated in Fig. 5(a), all the late vectors $v^L_{i,A}$, $v^L_{A,k}$ that act on cell A are always attractive, i.e., they point toward the interconnected nets. By moving the pin in these designated directions, the delay would be reduced, which would decrease the late timing violation. In contrast, all the early vectors $v^E_{i,A}$, $v^E_{A,k}$ are always repulsive, i.e., they point away from the interconnected cells. By moving the pin in these opposite directions, the delay would be increased, which would decrease the early timing violation. All vectors acting on cell j can be viewed as individual "forces" pushing and pulling cell j in their respective directions, toward and away from all interconnected cells. Naturally, these individual forces can be added to evaluate the *net* late and early effects on cell j. All late vectors are added together to form one resultant late vector T^L, while a similar process is followed to yield one resultant early vector T^E, as shown in Fig. 5(b).

The directions of these two resultant vectors must now be used to determine the exact placement location of the search window with respect to cell j. Each of the two resultant vectors T^L and T^E is projected onto the two coordinate axes, in order to extract its x (horizontal) and y (vertical) components. Subsequently, the horizontal and vertical components of the two vectors are added up in the following manner: all the positive horizontal components (pointing to the right) are added to form $|R_j|$; all the negative horizontal components (pointing to the left) are added to form $|L_j|$; all the positive vertical components (pointing upward) are added to form $|U_j|$; and, finally, all the negative vertical components (pointing downward) are added to form $|D_j|$.

Our goal is to place the search window of cell j in a position that is proportional to the calculated values of $|R_j|$, $|L_j|$, $|U_j|$, and $|D_j|$. Let us assume that the bottom-left corner of cell j is placed at the (x_j, y_j) position, the search window has width W and height H, and the location of the bottom-left corner of the search window is at (x_o, y_o). Initially, the search window is placed such that the bottom left corner of cell j is at the center of the search window. Therefore, $x_o = x_j - (W/2)$ and $y_o = y_j - (H/2)$. The new location of the bottom-left corner of the search window, $(\hat{x}_o, \hat{y}_o)$, is determined by using the following equations:

$$\hat{x}_o = x_j - \frac{|L_j|}{|L_j| + |R_j|} \cdot W, \quad \hat{y}_o = y_j - \frac{|D_j|}{|D_j| + |U_j|} \cdot H.$$

Essentially, the ratios of the left-to-right components and the up-to-down components determine the magnitude of the search window's slide in x and y directions.

Having determined the final location of the search window for cell j, we identify a set of N candidate positions uniformly spaced inside the search window. Let us denote the spatial granularity in the x and y dimensions as step_x and step_y, respectively. The values of step_x and step_y can be determined from the maximum displacement constraint and the relationship $W/\text{step}_x \cdot H/\text{step}_y = N$. We iterate from $\hat{y_o}$ to $\hat{y_o} + H$ with granularity step_y, and from $\hat{x_o}$ to $\hat{x_o} + W$ with granularity step_x. In this manner, a total of N different candidate positions are investigated for cell j, all situated within the search window.

VI. Experimental Results

The proposed flow was implemented in C++ using the open-source RSyn framework [18] as a single-threaded application. RSyn provides all necessary functions for netlist traversal and cell relocation, as well as incremental timing analysis needed by the proposed method. The new method is evaluated using the ICCAD-2015 benchmark set [19]. Table II shows some of the basic characteristics of each of the benchmarks used as well as the target density and the short and long maximum allowed displacement constraints that all cells should

TABLE II
ICCAD-2015 CONTEST BENCHMARK CHARACTERISTICS

Circuit	# Nodes	# Flops	# LCBs	Density	Max. Disp. (μm)	
					Short	Long
sb1	1209716	144266	7213	0.80	40	400
sb3	1213253	167923	8396	0.87	40	400
sb4	795645	176895	8843	0.90	20	400
sb5	1086888	114103	5704	0.85	30	400
sb7	1931639	270219	13510	0.90	50	500
sb10	1876103	241267	12063	0.87	20	500
sb16	981559	142543	7126	0.85	30	400
sb18	768068	103544	5177	0.8	20	400

satisfy. All experiments were performed on the same Linux-based workstation using a 3.6 GHz Intel Core i7-4790 with four cores and 32 GB of RAM. The final reported results are validated using the scripts provided by the contest organizers, and OpenTimer [30], which is the reference timer used for evaluation purposes in the above-mentioned contest.

In all cases, our method executes the flow depicted in Fig. 4, where flip-flop clustering precedes the iterative LR-based timing-driven placement. In each iteration, for the relocation of each cell, we identify 20 candidate positions uniformly spaced inside a rectangular search window of size $W = H = 20$ rows. For the examined benchmark set, 20 rows correspond roughly to 68 μm, which covers almost all of the allowed displacement in the case of the short displacement constraint, and it progressively reaches the maximum displacement in the case of the long displacement constraint.

A. Comparison With Winner of the ICCAD 2015 Contest

Table III summarizes the results achieved by the proposed algorithm, as compared to the initial design characteristics ("Init") and the performance of the first-place winner ("1st") of the ICCAD-2015 contest [16]. As shown in Table III, in all cases, the timing of the designs is either almost the same with that derived by the winner of the contest, or significantly better. Our method smoothly reduces both the late and early timing violations, without creating a tradeoff between the two. For instance, for short displacement, the winner of the contest improves the initial late WNS and TNS by 4.57% and 11.34%, respectively, on average, while the proposed method increases the average savings to 5.35% and 15.66%, respectively. At the same time, WNS and TNS for early timing are significantly more improved, as compared to [16]: early WNS is reduced by a further 40% (69.81% versus 29.96%), on average, and early TNS by a further 34% (84.29% versus 50.00%), for the short displacement constraint. The obtained results represent the combined effect of: 1) the timing-compatibility clustering that separates incompatible flip-flops, without enabling the dynamic timing update of the flip-flops's profiles; 2) the newly proposed LR-based timing optimization framework; and 3) the LM-driven cell relocation technique that moves all types of cells in a uniform manner. The number of iterations that LR-based cell relocation requires, per design, are depicted in the last column of Table III.

The proposed approach efficiently utilizes the greater placement freedom given by the long displacement limit. In all cases, timing is improved when compared to the results

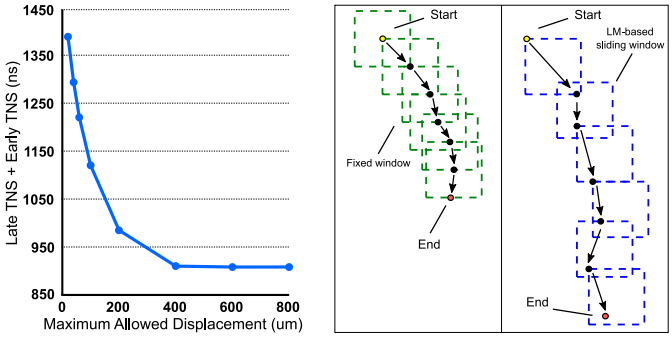

Fig. 6. (a) Final TNS on benchmark *sb3* for various displacement constraints. (b) Evolution of cell relocation using the proposed LM-driven search window and an equally sized search window.

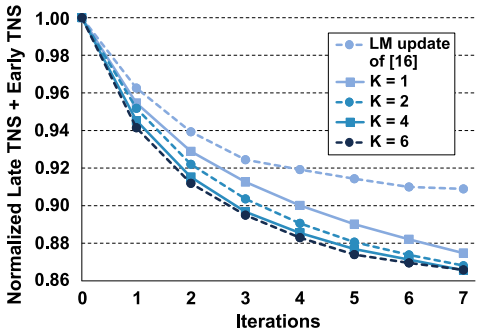

Fig. 7. TNS comparison between the LM update approach of [16] and the proposed LM update approach for the sb10 benchmark. Higher exponent values of K aim to increase the LMs of timing-critical arcs. As K increases, the TNS decreases more quickly, with diminishing returns beyond $K = 4$.

obtained for the short displacement limit, as shown in Table III. To better highlight how the proposed algorithm utilizes the available displacement, we performed timing-driven placement optimization on benchmark sb3 for various displacement constraints. The sum of early and late TNS achieved in each case is depicted in Fig. 6(a). Timing is improved with increasing displacement constraints, until the saturation is reached, which shows that the placement efficiency (in improving timing) has reached a plateau.

This result also stems from the proposed sliding of the search window. Fig. 6(b) displays the trajectory followed by a cell of the sb3 benchmark, and the progressive placement of the LM-based sliding window and a fixed window of the same size. Using the LM-based window, the cell is allowed to move quickly to its final destination by examining more timing-effective candidate positions, thus helping in converging faster to an overall timing-efficient solution. Additional experimental results reveal that, in all benchmarks, the replacement of the LM-based sliding of the search window by a fixed search window would degrade the overall timing quality by 10% (computed as the average degradation across all WNS/TNS timing metrics.)

The way the LMs are updated enables both fast convergence and better overall timing QoR, since the delay of timing-critical arcs is appropriately emphasized relative to other noncritical timing arcs. Fig. 7 compares the normalized sum of late and early TNS obtained using the LM update approach

TABLE III
TIMING IMPROVEMENT WITH SHORT AND LONG DISPLACEMENT LIMITS. LATE/EARLY WNS, LATE/EARLY TNS RESULTS, AS COMPARED TO THE FIRST-PLACE WINNER OF THE ICCAD-2015 CONTEST

Short Displacement

Circuit	Late						Early						LR iter
	WNS (ns)			TNS (ns)			WNS (ps)			TNS (ps)			
	Init	1st	Ours	Init	1st	Ours	Init	1st	Ours	Init	1st	Ours	
sb1	-4.98	-4.66	-4.60	-459.74	-374.31	-369.19	-9.34	-3.83	0.00	-317.44	-41.56	0.00	6
sb3	-10.15	-9.43	-9.12	-1502.83	-1373.12	-1301.26	-78.36	-65.72	-4.74	-1458.78	-683.51	-17.44	5
sb4	-6.22	-5.94	-5.99	-3476.69	-3195.33	-3065.46	-12.55	-6.08	-8.40	-519.39	-173.92	-50.20	14
sb5	-25.70	-25.07	-25.11	-6965.15	-6779.95	-6639.30	-36.77	-36.77	-12.00	-591.42	-585.78	-111.40	4
sb7	-15.22	-15.21	-15.21	-1857.38	-1703.78	-1579.86	-7.65	-6.75	-6.80	-1985.85	-1943.74	-1821.19	6
sb10	-16.49	-16.18	-16.28	-33152.80	-32514.40	-31649.30	-8.62	-8.62	-2.02	-620.95	-361.06	-13.11	6
sb16	-4.58	-4.36	-4.24	-776.04	-514.25	-418.72	-10.65	-8.38	-2.49	-113.75	-30.67	-2.49	8
sb18	-4.55	-4.12	-4.08	-1034.80	-943.64	-929.43	-19.01	-3.81	-0.03	-283.00	-69.38	-0.03	4
Avg	-10.99	-10.62	-10.58	-6153.18	-5924.85	-5744.06	-22.87	-17.50	-4.56	-736.32	-486.20	-251.98	6.62
Save	-	4.57%	5.35%	-	11.34%	15.66%	-	29.96%	69.81%	-	50.00%	84.29%	-

Long Displacement

Circuit	Late						Early						LR iter
	WNS (ns)			TNS (ns)			WNS (ps)			TNS (ps)			
	Init	1st	Ours	Init	1st	Ours	Init	1st	Ours	Init	1st	Ours	
sb1	-4.98	-4.57	-4.42	-459.74	-351.23	-323.94	-9.34	-16.65	0.00	-317.44	-80.89	0.00	9
sb3	-10.15	-8.70	-8.27	-1502.83	-1160.04	-881.59	-78.36	-13.13	-3.29	-1458.78	-214.03	-16.50	16
sb4	-6.22	-5.76	-5.60	-3476.69	-2464.56	-2309.54	-12.55	-12.28	-3.13	-519.39	-53.84	-13.80	8
sb5	-25.70	-24.29	-24.70	-6965.15	-5842.23	-6327.55	-36.77	-36.77	-12.24	-591.42	-618.27	-54.01	5
sb7	-15.22	-15.21	-15.21	-1857.38	-1510.76	-1454.46	-7.65	-6.75	-7.35	-1985.85	-1958.34	-1820.90	7
sb10	-16.49	-16.07	-16.13	-33152.80	-31517.80	-29445.10	-8.62	-5.15	-2.40	-620.95	-373.75	-12.50	5
sb16	-4.58	-3.84	-3.35	-776.04	-265.56	-209.59	-10.65	-7.55	-1.85	-113.75	-37.64	-1.85	15
sb18	-4.55	-3.81	-3.80	-1034.80	-775.84	-701.43	-19.01	-1.95	-0.02	-283.00	-6.86	-0.02	20
Avg	-10.99	-10.28	-10.19	-6153.18	-5486.00	-5206.65	-22.87	-12.53	-3.79	-736.32	-417.95	-239.95	10.62
Save	-	8.79%	11.14%	-	25.76%	31.46%	-	22.26%	74.52%	-	56.33%	86.47%	-

of [16] to that obtained when using the proposed LM update technique, for the sb10 benchmark with long displacement limit. For the proposed method, results with different exponents K for the LM updates of (18) and (19) are also shown in the figure. Obviously, the proposed LM update method yields significantly better TNS results than the LM update approach of [16]. As K is increased, the proposed method exhibits faster convergence and better overall timing QoR. Beyond $K = 4$, further improvement in TNS is marginal, thereby leading us to the selection of $K = 4$ for all our experiments.

The results of the proposed methodology presented thus far were obtained *without dynamic updates* of the timing profiles in the flip-flop clustering step. If dynamic timing update is enabled, the overall timing QoR is improved for the proposed method, as shown in Table IV. As an example, for the short displacement constraint, having the accurate slack values during flip-flop clustering helps improve early WNS by a further 2% (71.84% in Table IV versus 69.81% in Table III), while early TNS improves by a further 1% (85.44% versus 84.29%). Correspondingly, late WNS is improved by 0.2% (5.56% versus 5.35%), and late TNS by 0.5% (16.13% versus 15.66%). The reason for the minimal improvement is the fact that dynamic timing updates only affect weak flops with small timing slacks. Even with stale timing profiles, those flops are easily corrected by gate/flip-flop relocations in the subsequent optimization process.

Unfortunately, the minimal improvements in timing QoR with dynamic timing updates enabled come at a hefty runtime cost of 6×, on average, relative to disabling the dynamic timing updates. Hence, the decision to enable the dynamic timing update feature is left to the engineer, who may wish to investigate whether a particular design benefits from this additional step, or not.

TABLE IV
LATE/EARLY WNS AND LATE/EARLY TNS RESULTS WHEN DYNAMIC UPDATE OF TIMING PROFILES IS ENABLED IN THE FLIP-FLOP CLUSTERING STEP

Short Displacement

Circuit	Late		Early	
	WNS (ns)	TNS (ns)	WNS (ps)	TNS (ps)
sb1	-4.55	-363.10	0.00	0.00
sb3	-9.10	-1290.17	-2.12	-7.21
sb4	-5.99	-3083.13	-10.00	-16.67
sb5	-25.11	-6635.68	-14.00	-115.30
sb7	-15.21	-1562.71	-6.80	-1832.40
sb10	-16.27	-31563.40	-1.37	-6.33
sb16	-4.23	-415.31	0.00	0.00
sb18	-4.07	-924.28	0.00	0.00
Avg	-10.57	-5729.72	-4.29	-247.24
Save	5.56%	16.13%	71.84%	85.44%

Long Displacement

Circuit	Late		Early	
	WNS (ns)	TNS (ns)	WNS (ps)	TNS (ps)
sb1	-4.32	-313.12	0.00	0.00
sb3	-8.18	-860.39	-2.88	-7.99
sb4	-5.60	-2580.72	-4.00	-5.95
sb5	-24.29	-6176.79	-15.00	-60.10
sb7	-15.21	-1428.32	-6.87	-1830.00
sb10	-16.12	-28573.2	-1.37	-7.85
sb16	-3.30	-202.68	0.00	0.00
sb18	-3.79	-673.84	0.00	0.00
Avg	-10.10	-5101.13	-3.77	-238.99
Save	11.88%	32.18%	77.24%	86.84%

B. Comparison With Recent State-of-the-Art

In the next set of experiments, we compare the proposed algorithm to two recent state-of-the-art timing-driven

TABLE V
TIMING IMPROVEMENT WITH SHORT AND LONG DISPLACEMENT LIMITS. LATE/EARLY WNS, LATE/EARLY TNS RESULTS, AS COMPARED TO EHC [11] AND FPUSM [12]

Short Displacement

Circuit	Late						Early					
	WNS (ns)			TNS (ns)			WNS (ps)			TNS (ps)		
	EHC	FPUSM	Ours	EHC	FPUSM	Ours	EHC	FPUSM	Ours	EHC	FPUSM	Ours
sb1	-4.67	-4.66	-4.60	-373.99	-374.25	-369.19	-0.04	0.00	0.00	-0.04	0.00	0.00
sb3	-9.53	-9.43	-9.12	-1373.51	-1373.13	-1301.26	-59.59	-21.58	-4.74	-393.55	-153.08	-17.44
sb4	-5.94	-5.94	-5.99	-3153.70	-3195.38	-3065.46	-5.50	-2.20	-8.40	-65.32	-3.42	-50.20
sb5	-25.08	-25.07	-25.11	-6775.95	-6779.94	-6639.30	-31.33	-36.77	-12.00	-271.98	-265.37	-111.40
sb7	-15.22	-15.21	-15.21	-1696.02	-1703.81	-1579.86	-6.75	-6.36	-6.80	-1875.86	-1858.34	-1821.19
sb10	-16.19	-16.18	-16.28	-32514.40	-32514.50	-31649.30	-5.87	-2.19	-2.02	-306.75	-9.09	-13.11
sb16	-4.04	-4.36	-4.24	-444.16	-514.25	-418.72	-0.05	0.00	-2.49	-0.06	0.00	-2.49
sb18	-4.08	-4.12	-4.08	-938.77	-943.69	-929.43	-2.56	0.00	-0.03	-22.80	0.00	-0.03
Avg	-10.59	-10.62	-10.58	-5908.81	-5924.87	-5744.06	-13.96	-8.64	-4.56	-367.05	-286.16	-251.98
Save	5.39%	4.57%	5.35%	12.74%	11.34%	15.66%	53.02%	68.29%	69.81%	70.31%	81.12%	84.29%

Long Displacement

Circuit	Late						Early					
	WNS (ns)			TNS (ns)			WNS (ps)			TNS (ps)		
	EHC	FPUSM	Ours	EHC	FPUSM	Ours	EHC	FPUSM	Ours	EHC	FPUSM	Ours
sb1	-4.57	-4.57	-4.42	-351.06	-351.21	-323.94	-0.87	-0.43	0.00	-2.18	-0.43	0.00
sb3	-8.69	-8.70	-8.27	-1159.93	-1160.07	-881.59	-4.46	-5.54	-3.29	-9.10	-29.05	-16.50
sb4	-5.76	-5.76	-5.60	-2437.77	-2462.93	-2309.54	-12.28	0.00	-3.13	-55.62	0.00	-13.80
sb5	-24.29	-24.29	-24.70	-5840.52	-5842.28	-6327.55	-58.34	-36.77	-12.24	-61.10	-268.60	-54.01
sb7	-15.22	-15.21	-15.21	-1510.76	-1510.79	-1454.46	-6.75	-6.38	-7.35	-1958.34	-1858.48	-1820.90
sb10	-16.09	-16.07	-16.13	-31563.90	-31518.00	-29445.10	-2.73	-2.20	-2.40	-40.60	-3.47	-12.50
sb16	-3.69	-3.84	-3.35	-234.07	-265.57	-209.59	-0.20	0.00	-1.85	-0.31	0.00	-1.85
sb18	-3.78	-3.81	-3.80	-771.96	-775.87	-701.43	-0.20	0.00	-0.02	-0.20	0.00	-0.02
Avg	-10.26	-10.28	-10.19	-5483.75	-5485.84	-5206.65	-10.73	-6.42	-3.79	-265.93	-270.00	-239.95
Save	9.27%	8.79%	11.14%	26.40%	25.76%	31.46%	50.70%	72.42%	74.52%	84.02%	82.29%	86.47%

placement optimization methods; namely, EHC [11] and FPUSM [12]. The obtained results are summarized in Table V. The results of the proposed methodology are obtained without dynamic updates of the timing profiles in the flip-flop clustering step. In most cases, the proposed methodology achieves similar, or better, results, in terms of early timing, and it is always slightly better in late timing. The average savings ("Save") reported in the last rows of Table V report the average savings achieved by each method relative to the initial designs ("Init") reported in the second column of Table III.

Even though the improvements achieved by the proposed method may initially seem modest, as compared to EHC and FPUSM, the comparison should be viewed in the correct overall context: both EHC and FPUSM are not complete, self-contained optimization methods. Instead, they rely on another preceding technique to first close—as much as possible—the negative slack for late timing. Upon completion of this first step by the other technique, EHC and FPUSM focus on early timing optimization. This is precisely the reason why these two methods were applied to the outcome of the first-place winner of the ICCAD-2015 contest, which significantly preoptimized (especially for late timing) the designs. On the contrary, the proposed methodology is self-contained and does not rely on other techniques for any preoptimizations.

It should also be noted that, while EHC and FPUSM focus on LCB-to-flip-flop reassignments and LCB/flip-flop movements, the proposed approach *generalizes* this methodology by moving LCBs and flip-flops inside LR-based placement. In general, prior work only moves gates within LR-based placement. The new methodology is holistic and all-encompassing in its optimization approach, surpassing the performance of EHC and FPUSM in most cases for late timing, and closely matching their performance in early timing. More specifically, the new approach always performs better in *sb3* and *sb5* in early timing, and it is always worse in *sb10* in early TNS timing. The latter is attributed to the fact that, in *sb10*, the early critical paths are between flip-flops with very low placement freedom, because they are placed close to macros that block placement.

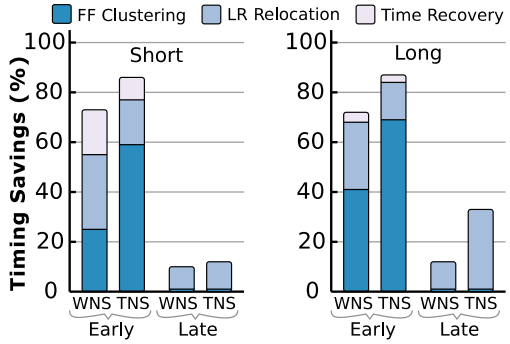

Fig. 8. Incremental timing savings relative to the initial benchmarks obtained by each step of the proposed method, for early and late timing.

The obtained timing QoR is the synergistic result of all three algorithms shown in the cell relocation flow of Fig. 4, i.e., FF clustering, LR-based cell relocation, and early timing recovery. Fig. 8 depicts the average impact of each step over all ICCAD 2015 benchmarks for short and long displacement limits. Timing-compatibility flip-flop clustering prepares the LCB-to-Flip-Flop connections in such a way that it reduces early timing, but leaves late timing unaffected. In contrast, the LR-based iterative optimization that follows is more complete

TABLE VI
TIMING IMPROVEMENT OF LATE WNS AND LATE TNS AS COMPARED TO OWARU [13] USING ONLY GATE RELOCATION

Circuit	Late Timing/Short Displacement			
	WNS (ns)		TNS (ns)	
	OWARU	Ours	OWARU	Ours
sb1	-4.8	-4.61	-426	-368.30
sb3	-9.8	-9.12	-1408	-1301.49
sb4	-6.1	-5.99	-3379	-3092.90
sb5	-25.5	-25.13	-6916	-6660.28
sb7	-15.2	-15.21	-1759	-1582.20
sb10	-16.4	-16.31	-32816	-31707.00
sb16	-4.4	-4.25	-605	-424.06
sb18	-4.2	-4.08	-997	-928.58
Avg	-10.8	-10.6	-6038.3	-5758.1
Save	2.75%	5.26%	6.15%	15.44%

and provides a combined improvement in both early and late timing. Finally, early timing recovery is a fast step that only contributes in early timing and its effect is more pronounced in short displacement limits.

Finally, the proposed incremental timing-driven placement technique was compared to OWARU [13], which focuses entirely on late timing optimizations. The obtained results are presented in Table VI. For a fair comparison with OWARU, we have replaced the performance numbers of our method (when applied as a whole) with the numbers achieved for late timing when performing only LR-based gate relocation, *excluding* FF clustering, FF and LCB movement, and timing recovery. In all cases, the proposed approach achieves better results than OWARU [13].

C. Runtime Comparisons

In addition to yielding timing improvements, the *runtime* of an optimization methodology is also a critical attribute. The runtime evaluation results in Table VII indicate that the proposed approach is close to the state-of-the-art. Note that the reported runtimes for the competing techniques are taken verbatim from their respective papers. Consequently, those runtimes correspond to other machines with different specifications than the one we used. Therefore, the comparisons can only be broadly and generally indicative. Regardless, we include those runtime numbers here in a big-picture context, to demonstrate that the runtimes of the proposed approach are reasonable, as compared to the others.

The runtimes of the new technique are close—in some cases better, in others worse—to the first-place winner of the ICCAD-2015 contest. However, the new approach utilizes its runtime more effectively, by yielding substantially better QoR, especially for early timing. In the case of long displacement, the new technique's runtimes are longer, since there are more options to search before converging to a solution. Also, the proposed method exhibits more-or-less similar runtimes to EHC and FPUSM. However, for a fair and meaningful runtime comparison to EHC and FPUSM, one would need to also add the runtime of the first-place winner of the ICCAD-2015 contest to those techniques. Recall, that both EHC [11] and FPUSM [12] are applied after the completion of the optimization of the first-place winner.

TABLE VII
RUNTIME COMPARISON OF STATE-OF-THE-ART TIMING-DRIVEN PLACEMENT TECHNIQUES. RUNTIMES ARE REPORTED IN MINUTES

Circuit	Short Displacement				Long Displacement			
	1st	EHC	FPUSM	Ours	1st	EHC	FPUSM	Ours
sb1	23	84	10	31	32	84	10	43
sb3	23	92	23	21	27	92	25	49
sb4	17	39	11	137	19	39	10	65
sb5	23	88	16	17	25	88	15	20
sb7	43	172	21	38	53	172	20	41
sb10	40	166	18	102	37	166	18	88
sb16	19	73	9	28	22	73	10	41
sb18	15	41	9	10	16	41	10	32

Runtime (in minutes) of each step of the proposed methodology.

Method		sb1	sb3	sb4	sb5	sb7	sb10	sb16	sb18
Short	FF-Clust	3.8	4.8	5.1	2.2	12.4	9.8	3.5	1.9
	LR-Reloc	24.7	13.9	129.2	13.1	22.8	88.3	22.0	6.4
	Tim-Rec	2.5	2.0	2.8	1.7	3.0	4.0	2.1	1.4
Long	FF-Clust	3.8	4.8	5.1	2.2	12.4	9.8	3.5	1.9
	LR-Reloc	36.6	42.1	56.5	15.7	25.8	73.9	35.0	28.2
	Tim-Rec	2.6	2.1	2.9	1.8	3.0	4.1	2.1	1.8

The runtime of the proposed method is the additive result of the three main steps of the overall flow. The contribution of each part is shown at the bottom of Table VII for all benchmarks. As expected, the LR-based cell relocation consumes the majority of the runtime, while early timing recovery has a small marginal contribution. The FF clustering's runtime is always a small, or medium, percentage of the runtime of LR relocation, and its runtime is, by construction, the same for long and short displacement limits.

VII. CONCLUSION

Timing-driven placement optimization is an integral cog of the complex process of achieving timing closure, and it is one of the key determinants of the overall QoR. This article presented a novel timing-driven placement optimization methodology based on an extended LR formulation. The fundamental contribution of this new approach is the concerted relocation of all types of cells (gates, flip-flops, and LCBs) in a unified manner. The LR-based placement optimization is complemented by a flip-flop clustering algorithm that ensures the timing compatibility of the flip-flops of each cluster, thus facilitating the timing optimization through LCB movement. Additionally, a simple, yet effective, scaling factor artificially changes the distances of the members of the cluster from the cluster center. This helps to create the clusters of uneven sizes, thereby appropriately delaying, or speeding up, the clock arrival time. Extensive experimental evaluations using the ICCAD-2015 benchmarks demonstrate the efficacy of the proposed methodology, as compared to four state-of-the-art timing-driven placement-optimization techniques.

ACKNOWLEDGMENT

The authors would like to thank D. Chinnery, Mentor, a Siemens Business, USA, for his valuable comments.

REFERENCES

[1] L. Lavagno, I. L. Markov, G. Martin, and L. K. Scheffer, *Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology*. Boca Raton, FL, USA: Taylor and Francis Group, 2016.

[2] N. D. MacDonald, "Timing closure in deep submicron designs," in *Proc. Design Autom. Conf. (DAC)*, 2010.
[3] L.-C. Lu, "Physical design challenges and innovations to meet power, speed, and area scaling trend," in *Proc. ISPD*, 2017, p. 63.
[4] N. Viswanathan *et al.*, "ITOP: Integrating timing optimization within placement," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 2010, pp. 83–90.
[5] G. Wu and C. Chu, "Two approaches for timing-driven placement by Lagrangian relaxation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 12, pp. 2093–2105, Dec. 2017.
[6] A. Srinivasan, K. Chaudhary, and E. S. Kuh, "RITUAL: A performance driven placement algorithm for small cell ICs," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 1991, pp. 48–51.
[7] T. Luo, D. Newmark, and D. Z. Pan, "A new LP based incremental timing driven placement for high performance designs," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2006, pp. 1115–1120.
[8] H. Ren, D. Z. Pan, and D. S. Kung, "Sensitivity guided net weighting for placement driven synthesis," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 2004, pp. 10–17.
[9] T. Kong, "A novel net weighting algorithm for timing-driven placement," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2002, pp. 172–176.
[10] A. Chowdhary *et al.*, "How accurately can we model timing in a placement engine?" in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2005, pp. 801–806.
[11] C.-C. Huang, Y.-C. Liu, Y.-S. Lu, Y.-C. Kuo, Y.-W. Chang, and S.-Y. Kuo, "Timing-driven cell placement optimization for early slack histogram compression," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, 2016, p. 8.
[12] S. Kim, S. Do, and S. Kang, "Fast predictive useful skew methodology for timing-driven placement optimization," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2017, pp. 1–6.
[13] J. Jung, G. Nam, L. Reddy, I. Jiang, and Y. Shin, "OWARU: Free space-aware timing-driven incremental placement with critical path smoothing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 9, pp. 1825–1838, Sep. 2018.
[14] G. Flach *et al.*, "An incremental timing-driven flow using quadratic formulation for detailed placement," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2015, pp. 1–6.
[15] G. Flach, M. Fogaça, J. Monteiro, M. D. O. Johann, and R. A. D. L. Reis, "Drive strength aware cell movement techniques for timing driven placement," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 2016, pp. 73–80.
[16] C. Guth, V. S. Livramento, R. Netto, R. Fonseca, J. L. Güntzel, and L. C. V. D. Santos, "Timing-driven placement based on dynamic net-weighting for efficient slack histogram compression," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 2015, pp. 141–148.
[17] V. S. Livramento, R. Netto, C. Guth, J. L. Güntzel, and L. C. V. D. Santos, "Clock-tree-aware incremental timing-driven placement," *ACM Trans. Design Autom. Electron. Syst.*, vol. 21, no. 3, pp. 1–27, Apr. 2016.
[18] G. Flach, M. Fogaça, J. Monteiro, M. D. O. Johann, and R. A. D. L. Reis, "RSYN: An extensible physical synthesis framework," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 2017, pp. 33–40.
[19] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan, "ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 921–926.
[20] A. K. Jain, "Data clustering: 50 years beyond *k*-means," *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, 2010.
[21] G. Wu, Y. Xu, D. Wu, M. Ragupathy, Y. Mo, and C. C. N. Chu, "Flip-flop clustering by weighted *k*-means algorithm," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.
[22] J. C. Puget, G. Flach, R. A. D. L. Reis, and M. D. O. Johann, "JEZZ: An effective legalization algorithm for minimum displacement," in *Proc. Symp. Integr. Circuits Syst. Design (SBCCI)*, Aug. 2015, pp. 1–5.
[23] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*. Heidelberg, Germany: Springer, 2009.
[24] V. S. Livramento, C. Guth, J. L. Güntzel, and M. O. Johann, "Fast and efficient Lagrangian relaxation-based discrete gate sizing," in *Proc. Design Autom. Test Europe (DATE)*, Mar. 2013, pp. 1855–1860.
[25] A. Sharma, D. G. Chinnery, S. Bhardwaj, and C. C. N. Chu, "Fast Lagrangian relaxation based gate sizing using multi-threading," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2015, pp. 426–433.
[26] G. Shklover and B. Emanuel, "Simultaneous clock and data gate sizing algorithm with common global objective," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 2012, pp. 145–152.
[27] A. Sharma, D. G. Chinnery, and C. Chu, "Lagrangian relaxation based gate sizing with clock skew scheduling—A fast and effective approach," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 2019, pp. 129–137.
[28] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 1, pp. 70–83, Jan. 2008.
[29] H. Tennakoon and C. Sechen, "Gate sizing using Lagrangian relaxation combined with a fast gradient-based pre-processing step," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2002, pp. 395–402.
[30] T.-W. Huang and M. D. F. Wong, "OpenTimer: A high-performance timing analysis tool," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2015, pp. 895–902.

Dimitrios Mangiras received the Diploma degree in electrical and computer engineering from the Democritus University of Thrace, Xanthi, Greece, in 2017, where he is currently pursuing the Ph.D. degree.

His current research interests include electronic design automation for physical design, clock tree synthesis and machine-learning-based optimization, as well as design of energy-efficient integrated circuits and automated verification methodologies.

Apostolos Stefanidis received the Diploma degree in electrical and computer engineering from the Democritus University of Thrace, Xanthi, Greece, in 2017, where he is currently pursuing the Ph.D. degree.

His current research interests include electronic design automation, with emphasis in machine learning applications on autonomous timing and power optimization.

Ioannis Seitanidis received the Dipl.Ing. and Ph.D. degrees in electrical and computer engineering from the Democritus University of Thrace, Xanthi, Greece, in 2013 and 2018, respectively.

Since 2018, he has been a Software Research and Development Engineer with Mentor, a Siemens Business, Grenoble, France. His current research interests include electronic design automation, design optimization, and physical synthesis.

Chrysostomos Nicopoulos received the B.S. and Ph.D. degrees in electrical engineering with a specialization in computer engineering from Pennsylvania State University, State College, PA, USA, in 2003 and 2007, respectively.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus. His current research interests include networks-on-chip, computer architecture, multi/many-core microprocessor, and computer system design.

Giorgos Dimitrakopoulos received the B.S., M.Sc., and Ph.D. degrees in computer engineering from the University of Patras, Patras, Greece, in 2001, 2003, and 2007, respectively.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece. His current research interests include the design of digital integrated circuits, electronic design automation, and computer architecture, with emphasis in low-power systems design.