MrDP: Multiple-Row Detailed Placement of Heterogeneous-Sized Cells for Advanced Nodes

Yibo Lin, Bei Yu, *Member, IEEE*, Xiaoqing Xu, *Member, IEEE*, Jhih-Rong Gao, Natarajan Viswanathan, Wen-Hao Liu, Zhuo Li, *Senior Member, IEEE*, Charles J. Alpert, *Fellow, IEEE*, and David Z. Pan, *Fellow, IEEE*

***Abstract*—As very large-scale integration technology shrinks to fewer tracks per standard cell, e.g., from 10 to 7.5-track libraries (and lesser for 7 nm), there has been a rapid increase in the usage of multiple-row cells like two- and three-row flip-flops, buffers, etc., for design closure. Additionally, the usage of multibit flip-flops or flop trays to save power creates large cells that further complicate critical design tasks, such as placement. Detailed placement happens to be a key optimization transform, which is repeatedly invoked during the design closure flow to improve design parameters, such as wirelength, timing, and local wiring congestion. Advanced node designs, with hundreds of thousands of multiple-row cells, require a paradigm change for this critical design closure transform. The traditional approach of fixing multiple-row cells during detailed placement and only optimizing the locations of single-row standard cells can no longer obtain appreciable quality of results. It is imperative to have new techniques that can simultaneously optimize both multiple- and single-row height cell locations during detailed placement. In this paper, we propose a new density-aware detailed placer for heterogeneous-sized netlists. Our approach consists of a chain move scheme that generalizes the movement of heterogeneous-sized cells, a nested dynamic programming-based approach for ordered double-row placement and a network flow-based formulation to solve ordered multiple-row placement for wirelength and density optimization. Experimental results demonstrate the effectiveness of these techniques in wirelength minimization and density smoothing compared with the most recent detailed placers for designs with heterogeneous-sized cells.**

***Index Terms*—Chain move, detailed placement, multiple-row height cells, network flow, physical design.**

I. Introduction

USING single-row height standard cells has been the dominant methodology for modern very large-scale integration (VLSI) digital design. For a given technology node, the height and width of standard cells are carefully selected to optimize various characteristics, such as timing, packing, and pin accessibility. The common nomenclature for cell libraries is "*N*"-track, with *N* being the height of the circuit row and standard cells in terms of the number of covered routing tracks. The last few years have seen a steady decrease in *N* with each new technology node, e.g., from 10 to 7.5 (and possibly lesser for 7 nm). In this scenario, it is getting increasingly difficult to design complex circuit components (flip-flops, muxes, etc.) as single-row height cells, while satisfying required performance and routing characteristics. As a result, advanced node designs are increasingly adopting the design and usage of multiple-row height cells for such complex circuit components.

Additionally, to satisfy stringent power requirements, flip-flop merging and usage of multibit flip-flops (MBFFs) or flop trays is becoming increasingly prevalent [1]–[3] in modern designs. MBFF enables the sharing of clock buffers between flip-flops, which decreases both power and area. Statistics show that a 2-bit MBFF is able to achieve around 14% power reduction and 4% area reduction per bit, while a 4-bit MBFF can achieve around 22% power reduction and 29% area saving per bit [3]. But MBFFs happen to be large, multiple-row height cells. This significantly increases the complexity for steps like legalization and detailed placement.

In addition, to meet die-size requirements for area, power, and cost reduction, design densities are approaching the limit. It is common for designs with up to 90% density, which makes detailed placement critical to resolve local wiring congestion. In an extremely dense design, it is very difficult to insert or move large cells during legalization and detailed placement without significant disruption to the local neighborhood. Furthermore, the number of interconnect pins per standard cell varies for a given cell library and often lacks correlation to the cell area. Without careful planning, local congestion can be caused by accumulation of cells with high pin count. Therefore, it is critical to make proper usage of the limited die area for optimizing both wirelength and congestion.

Placement is usually divided into three steps, global placement, legalization, and detailed placement [4]. Global placement determines the rough locations of cells while minimizing objectives, such as wirelength, routability, and timing. But the solution from global placement often contains overlap and thus is not design rule friendly. Legalization removes overlaps and aligns cells to placement sites. Finally, detailed placement tries to further improve the solution by moving cells locally. Sometimes legalization is integrated into detailed placement instead of a separate step.

Global placement techniques are fairly mature in handling the mixed-sized placement problem [5]–[10]. But there has been little research in detailed placement for heterogeneous-sized netlists, especially where the number of multiple-row

Manuscript received March 21, 2017; revised June 30, 2017; accepted August 3, 2017. Date of publication August 31, 2017; date of current version May 18, 2018. This work was supported in part by the National Science Foundation under Project CCF-1218906 and Project CCF-1718570, and in part by the Research Grants Council of Hong Kong SAR under Project CUHK24209017. The preliminary version has been presented at the International Conference on Computer-Aided Design in 2016. This paper was recommended by Associate Editor P. Gupta. *(Corresponding author: Yibo Lin.)*

Y. Lin, X. Xu, and D. Z. Pan are with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78741 USA (e-mail: yibolin@utexas.edu).

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

J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, and C. J. Alpert are with Cadence Design Systems Inc., Austin, TX 78759 USA.

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.2017.2748025

height cells ranges in the hundreds of thousands, as seen in advanced node designs. Wu and Chu [11] proposed a straightforward technique to handle double-row height cells during detailed placement. In their method, they use cell grouping and cell inflation to convert all the single-row height cells in the design to double-row height cells. This results in a placement problem with only double-row height cells. Consequently, a conventional placement engine can be used to optimize the designs. However, this approach is unable to handle the power line alignment constraint from multiple-row height cells; e.g., cells with power rail on top and bottom have to be placed in rows with the same power line configuration. Another key drawback with this approach is its inability to handle larger cells that span three or more circuit rows. Chow *et al*. [12] proposed the first legalization algorithm for multiple-row height standard cells with an objective of displacement minimization. They explored the insertion points in the layout and try to remove overlaps with minimum displacement. Wang *et al*. [13] adapted Abacus engine to handle multiple-row height cells. Hung *et al*. [14] improved the solution quality by linear programming (LP). Chen *et al*. [15] solved a linear complementary problem for displacement minimization in legalization. A summary on recent detailed placement challenges and approaches can be found in [16].

To address the challenges in placement for advanced technology nodes, we propose a detailed placer for heterogeneous-sized netlists that addresses the traditional detailed placement objectives of wirelength, cell density, and pin density [4], [5], [11], [17]–[20]. The major contributions are summarized as follows.

1) A chain move scheme that generalizes the movement of heterogeneous-sized cells to optimize wirelength, cell density, and pin density by searching for the maximum prefix sum of the improvements.
2) A nested dynamic programming (DP)-based technique solving ordered double-row (ODR) placement for wirelength optimization.
3) A network flow-based formulation to solve ordered multiple-row (OMR) placement that is flexible to both displacement minimization and wirelength optimization.
4) Outperform the most recent detailed placer for multiple-row height cells by 3.7% in scaled wirelength, 20.2% in cell density, and 13.4% in pin density.

The rest of this paper is organized as follows. Section II illustrates the special constraints and problem formulation for the placement. Section III provides a detailed explanation of our proposed techniques. Section IV verifies the effectiveness of our approach, followed by the conclusion in Section V.

II. Preliminaries and Overall Flow

In this section, we will explain the constraints in placement for designs with heterogeneous-sized standard cells and give the problem formulation.

A. Power Line Alignment

Power line alignment is a special placement constraint from a multiple-row height cell. In modern VLSI layouts, the power lines that connect to standard cells are typically located at the bottom and top of placement rows. Meanwhile, standard cells have to align to placement rows for proper power line alignment. Fig. 1 illustrates an layout example of seven

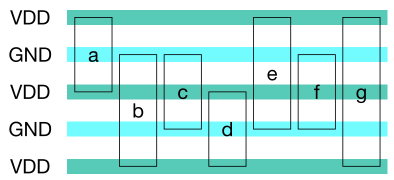


Fig. 1. Example of multiple-row height cells in a layout.

multiple-row height cells, where five cells take even number of rows (i.e., cells a, c, d, f, and g). Cells a, d, and g have power rails (VDD) on top and bottom of the cells, and ground rails (GND) in the middle. They must be placed in alternative rows with proper VDD/GND alignment, since we cannot fix the alignment through cell flipping or rotation. Similarly, cells c and f have VDD in the middle and GND on the top and bottom. The bottom of such cells must be aligned to rows with GND at the bottom. However, for cells with odd number of rows, such as cell b and e, there is no such constraint, since it has power rail on the top or bottom and ground rail on the other side. This configuration is the same as single-row height cells, so cell flipping and rotation can fix the alignment issue.

The constraint for power line alignment can be summarized as follows. An even-row height cell must align to placement rows with the same type of power line at the bottom as that in the cell, while any odd-row height cell, including single-row height cell, can align to any placement row with proper orientation.

B. Problem Formulation

In modern VLSI placement, the optimization usually includes multiple objectives, such as wirelength and density. Wirelength is still regarded as the major objective, while density metrics cannot be neglected, because pure wirelength-driven placement often produces congested solution that results in difficulty for post-placement stages, such as routing. Therefore, in this paper we adopt the scaled wirelength metric from International Conference on Computer-Aided Design (ICCAD) 2013 placement contest [21] considering both wirelength and cell density. Half-perimeter wirelength (HPWL) is used as the wirelength metric, which is defined as follows:

$$\mathrm{HPWL} = \sum_{n \in N} \max_{i \in n} x_i - \min_{i \in n} x_i + \max_{i \in n} y_i - \min_{i \in n} y_i \tag{1}$$

where N denotes the set of interconnections in the circuit.

Average bin utilization (ABU) evaluates the density of a placement solution [8]. The average density of the top γ% bins of highest utilization is denoted by ABU_γ. The ABU penalty for density is computed from a weighted sum of overflow, which is defined in the following equations:

$$\mathrm{overflow}_\gamma = \max\left(0, \frac{\mathrm{ABU}_\gamma}{d_t} - 1\right) \tag{2a}$$

$$\mathrm{ABU} = \frac{\sum_{\gamma \in \Gamma} w_\gamma \cdot \mathrm{overflow}_\gamma}{\sum_{\gamma \in \Gamma} w_\gamma}, \Gamma \in \{2, 5, 10, 20\} \tag{2b}$$

where d_t denotes the target utilization and w_2, w_5, w_{10}, and w_{20} are set to 10, 4, 2, and 1, respectively. With the definition of ABU penalty, ICCAD 2013 placement contest defines a scaled wirelength cost to generalize both wirelength and density costs as

$$\mathrm{sHPWL} = \mathrm{HPWL} \cdot (1 + \mathrm{ABU}). \tag{3}$$

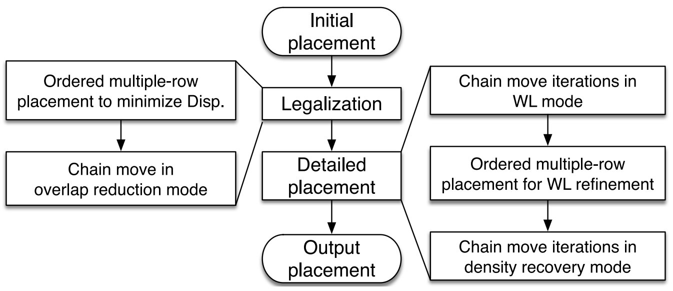

Initial placement
Ordered multiple-row placement to minimize Disp.
Legalization
Chain move in overlap reduction mode
Detailed placement
Output placement
Chain move iterations in WL mode
Ordered multiple-row placement for WL refinement
Chain move iterations in density recovery mode

Fig. 2. Overall flow of placement.

In the ICCAD 2013 placement contest, only cell area utilization is included in the computation of ABU. In advanced technology nodes, area utilization is not enough to model the congestion, because some large cells may contain very few pins, while some small cells may in the contrast involve a lot of interconnections. So we propose average pin utilization (APU) that captures the pin distribution of the layout. The pin density in each bin is the ratio of number of pins to the number of placement sites in the bin. Once the pin density map is obtained, the computation of APU penalty is the same as that of ABU, shown in the following equations:

$$\text{overflow}_\gamma^p = \max\left(0, \frac{\text{APU}_\gamma}{d_t^p} - 1\right) \tag{4a}$$

$$\text{APU} = \frac{\sum_{\gamma\in\Gamma} w_\gamma \cdot \text{overflow}_\gamma^p}{\sum_{\gamma\in\Gamma} w_\gamma}, \Gamma \in \{2, 5, 10, 20\} \tag{4b}$$

where d_t^p denotes target pin utilization and APU_γ denotes the average pin density of the top $\gamma\%$ bins of highest pin utilization.

With all the metrics defined, the multiple-row detailed placement (MrDP) problem is defined as follows.

Problem 1 (MrDP): Given an initial heterogeneous-sized standard cell placement plus a number of fixed macro blocks, either legal or not, we produce a legal placement solution with optimized wirelength and density, i.e., sHPWL and APU.

C. Overall Flow

The overall flow of our placement engine is shown in Fig. 2. Given the placement solution from global placement, we first check whether the placement is legal. If it is not legal, legalization is performed to remove overlaps and align power line of multiple-row cells. In this step, we first perform OMR placement to remove as much overlap as possible with minimum displacement. Then chain move algorithm (see Section III-A) in overlap reduction mode is performed to further remove rest overlaps. These two techniques are usually powerful enough to remove all the overlaps as long as the design has reasonable utilization. If there are still remaining overlaps, we search for nearest available locations for remaining cells that still contain overlaps, while this is never triggered in the experiment. Then we perform wirelength optimization to improve both wirelength and density until less than 1% cells are moved or maximum iteration is reached. We allow at most six iterations in the experiment. The OMR placement (see Sections III-B and III-C) is performed to further optimize wirelength. Before the final placement is produced, we refine density by invoking chain move algorithm in density recovery mode because wirelength optimization often pack cells together at the cost of density degradation.

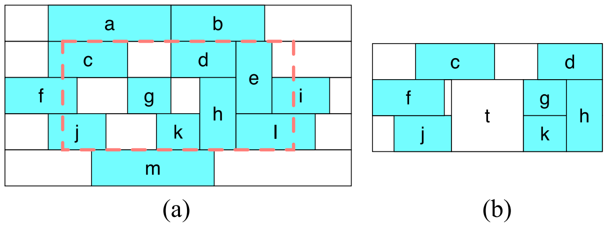


Fig. 3. Example of (a) placement with multiple-row height cells and (b) inserting another cell t by slightly shifting cell g and j.

III. Detailed Placement for Multiple-Row Cells

In this section, we will explain our placement algorithms, such as chain move and ODR placement in detail.

A. Chain Move Algorithm

One of the typical detailed placement approaches is to improve wirelength in a cell-by-cell manner; i.e., pick a cell and move to better position or try to swap with another cell for better wirelength [4], [18], [19]. It is proved to be very effective in the detailed placement for single-row cells. However, the situation changes when it comes to multiple-row height cells. Since a multiple-row height cell occupies the space of contiguous rows, it is more likely to involve overlaps with multiple cells, which results in the failure of position search with previous approach. Fig. 3(a) gives an example of placement which is difficult to insert another multiple-row height cell t into the dashed region without perturbing at least two cells. With slightly shifting cells g and j, shown in Fig. 3(b), cell t can be placed in the dashed region without overlap. Similar situation may also occur to very large single-row height cells which are difficult to be fit into dense regions without perturbation of multiple cells.

If it is able to allow the movement of multiple cells at a time, there will be more candidate positions for better placement quality. Inspired by density preserving refinement from [9] and gain map from [22] and [23], we develop an algorithm to allow other cells to move together when optimizing a target cell.

Definition 1 (Chain Move): Each chain move contains a set of movements for one or several cells.

A chain move involving multiple cells is usually triggered by the attempts of inserting a cell into a position resulting in overlaps with existing cells in that region, so the overlapped cells need to find new positions to resolve overlaps. If a cell is placed to a position without any overlap, there is only a single movement in the chain move.

Definition 2 (Cell Pool): It is a queue structure used for temporary storage of cells within a chain move.

In the example of Fig. 3, cell t overlaps with cells g and j when inserted to the dashed region, so cells g and j are added to the cell pool. In the following steps, cells in the cell pool are first popped out and placed until the cell pool goes empty, which indicates the end of a chain move.

Definition 3 (Scoreboard): It consists of an array of chain move entries with corresponding changes in wirelength cost for each chain move.

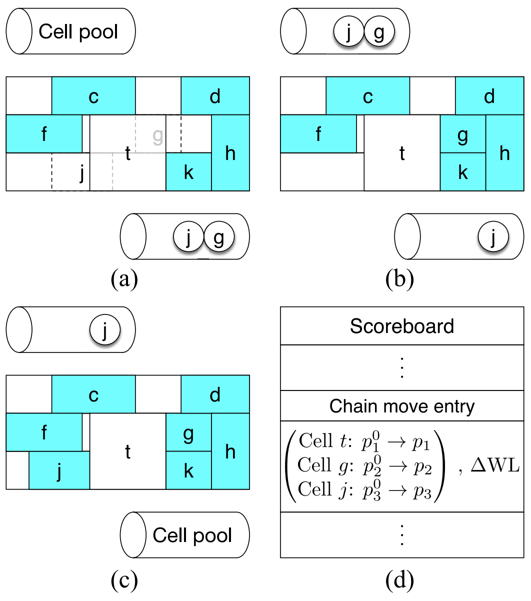

Cell pool
c
d
f
g
t
h
j
k
(a)
(b)
(c)
(d)
Scoreboard
Chain move entry
$\begin{pmatrix}\text{Cell } t\text{: } p_1^0 \to p_1 \\ \text{Cell } g\text{: } p_2^0 \to p_2 \\ \text{Cell } j\text{: } p_3^0 \to p_3\end{pmatrix}$, ΔWL

Fig. 4. Chain move example of (a) first movement: place cell t to p_1 from p_1^0 and push overlapped cell g and j to cell pool, (b) second movement: pop cell g from cell pool and place to p_2 from p_2^0, (c) third movement: pop cell j from cell pool and place to p_3 from p_3^0, and (d) corresponding chain move entry in scoreboard.

Algorithm 1 Chain Move Algorithm

```
Require: A set of placed cells $C$ in the layout.
Ensure: Move cells to minimize wirelength cost.
1: ReorderCells($C$);
2: Re-structure $C$ as a queue;
3: while $C$ is not empty or $Pool$ is not empty do
4:     if $Pool$ is not empty then
5:         $c_i \leftarrow Pool$.pop();
6:     else
7:         $c_i \leftarrow C$.pop();
8:         if $c_i$ has already been moved then
9:             Continue;
10:        end if
11:    end if
12:    $r_i \leftarrow$ ComputeSearchRegion($c_i$);
13:    $A_i \leftarrow$ collect candidate positions in $r_i$;
14:    $cost_b \leftarrow \infty$;
15:    for each $a_j \in A_i$ do
16:        $(cost_i, p_i, O_i) \leftarrow$ ComputeMoveCost($c_i$, $a_j$);
17:        if $cost_i < cost_b$ then
18:            $cost_b \leftarrow cost_i$; $p_b \leftarrow p_i$; $O_b \leftarrow O_i$;
19:        end if
20:    end for
21:    Move $c_i$ to $p_b$;
22:    $Pool$.push($O_b$);
23:    $Board$.last.append($c_i, p_i^0 \rightarrow p_b$);
24:    if $Pool$ is empty then
25:        Compute ΔWL for $Board$.last;
26:    end if
27: end while
28: BacktraceToBestEntry($C$, $Board$);
```

TABLE I
NOTATIONS USED IN CHAIN MOVE ALGORITHM

$Pool$	The cell pool.
$Board$	The scoreboard.
p_i^0	Initial position of Cell c_i.
p_i	Candidate position of cell c_i.
O_i	The set of cells overlapping with cell c_i at p_i.
$cost_i$	The cost of cell c_i at p_i.
$p_b, O_b, cost_b$	Correspond to best $p_i, O_i, cost_i$, respectively.

Since the positions of all cells are determined at the end of a chain move, we can compute accurate wirelength cost and record the differences with that at the beginning of the chain move. The scoreboard can help find a cumulatively good solution instead of that in a very greedy approach which usually requires improvements in each movement.

For the chain move example in Fig. 3, Fig. 4 gives the corresponding example of interaction between the cell pool and scoreboard. Here the horizontal cylinders on top of each Fig. 4(a)–(c) indicate the status of the cell pool before any movement, while the ones on the bottom indicate the status after the movements. At the beginning of the first movement, the cell pool is empty. Cell t is moved to position p_1 from p_1^0 but results in overlap with cells g and j during the first movement, so they are pushed into the cell pool. In the second movement, cell g is popped from the cell pool and moved to position p_2 from p_2^0 to resolve overlap. Similarly, the third movement places cell j to position p_3 from p_3^0. Fig. 4(d) shows the corresponding chain move entry in the scoreboard, which not only records each movement but also the change of wirelength cost before and after this chain move.

1) Overview of Chain Move Algorithm: The overview of the chain move algorithm is shown in Algorithm 1 and the notations are defined in Table I. In general each cell is only allowed to move once during one iteration. The function `ReorderCells` in line 1 of Algorithm 1 shuffles the cell sequence in C. Then cell set C is copied to a first-in-first-out queue structure and the main loop of chain move algorithm begins.

Within the loop, we first try to fetch a cell from the cell pool. If the cell pool is empty, we then obtain the first cell c_i in C. Then region r_i for cell c_i is computed for search of candidate positions, which is completed by function `ComputeSearchRegion`. The power line alignment constraints are considered during the selection of candidate positions.

For each candidate position a_j in A_i, the cost is computed by function `ComputeMoveCost` and the position with the best cost is applied to the cell from lines 15 to 28. When applying the best position, it is necessary to push all the overlapped cells in O_b to the cell pool and update the movement records in the scoreboard. If the cell pool goes to empty after a movement, which means the end of the chain move, we can now compute the accurate wirelength change and update the scoreboard. At the end of each pass, function `BacktraceToBestEntry` scans the scoreboard to find the best cumulative wirelength.

2) Max Prefix Sum of Wirelength Improvement: Like that in the well-known KL and FM partitioning algorithm [22], [23], we have a scoreboard that records the wirelength changes in each chain move, which helps find the maximum prefix sum of wirelength improvement by `BacktraceToBestEntry`. So the chain move scheme allows temporary degradation of wirelength as long as it eventually achieves better solutions, which can help find the best cumulative wirelength.

3) Constraints to Chain Move: There exist corner cases where a cell may fail to find any legal position in its search region. The corner case is likely to be triggered when all cells in a dense region have already been moved in this pass, because each cell is only allowed to move once in each pass. If such corner cases are triggered, we discard current chain and recover all the movements in this chain. Another corner case lies in the involvement of too many cells in a chain, which may result in the difficulty in searching for legal positions for the last cell. Therefore, we set an upper bound to the length of a chain to avoid long chains. Any chain exceeding the upper bound will trigger the discarding process. The maximum length of chain is set to 10 000, but it is never triggered in the experiment.

Lemma 1: If the placement is legal at the beginning of a chain move, the legality is maintained at the end of the chain move.

Proof: If the chain is discarded, all movements are recovered, so there is no perturbation to the placement. Otherwise, the chain ends because the cell pool goes empty, which means the last movement does not cause any overlap. So the placement is still legal at the end of the chain move. The maintenance of legality is very meaningful to avoid wirelength degradation from extra legalization effort. ■

4) Visiting Order of Cells: The visiting order of cells during each pass matters to the solution quality. If we keep a fixed order for each iteration, the wirelength saturates quickly and fails to descent further. So a suitable visiting order is essential to the solution quality under different objectives. Here we discuss the details about the function `ReorderCells` for different optimization objectives. In overlap reduction mode, multiple-row height cells and large cells have higher priority, because it is easier for small cells to find overlap-free positions and thus a legal placement can be found more efficiently. When it comes to wirelength minimization from a legal placement, those cells far away from their optimal regions are granted with high priority, because higher gain can be achieved by moving cells with longer distances.

5) Search Region Computation: We discuss the function `ComputeSearchRegion` here on search region computation. First we compute the optimal region as most previous global move algorithms do [18], but it is often congested. We extend the optimal region by mirroring the original position of the cell to the center of the optimal region and form a new box. Any bin intersecting with the search region will be considered for collection of candidate positions to the set A_i. We check bins from the ones close to the optimal region to farther ones. We observe that after several updates in lines 18–20 for each cell, the final solution quality converges. To save runtime we exit early from the loop after trying several positions for each cell in the experiment.

6) Move Cost Computation: Now we explain the function `ComputeMoveCost`. The objective of the placement includes wirelength and density. In addition, each movement may lead to overlapping cells that will be collected to the cell pool. So the cost consists of three parts: 1) wirelength cost; 2) density cost; and 3) overlap cost, shown as follows:

$$\text{cost} = \Delta\text{WL} \cdot (1 + \alpha \cdot c_d) + \beta \cdot c_{ov} \tag{5}$$

where ΔWL denotes the wirelength cost, c_d denotes density cost, and c_{ov} denotes the overlap cost. The weights α and β are set to 1.5 and 0.5 in the experiment.

Wirelength cost is in general defined as the HPWL change for the movement. However, if the cell is connected to some cells in the cell pool whose positions are not determined yet, such connections are ignored.

In the density cost, we consider both area density and pin density. In the placement that involves multiple-row height cells, the cells can be very large and result in the intersections with multiple bins. So the density increases in all bins are summed up for cost. Let c_{ad} denote the cost of area density and c_{pd} denote the cost of pin density. Let B be the set of bins intersected with the cell c_i at candidate position p_i and $d_a(b)$ and $d_p(b)$ denote the original area and pin density for bin b

$$c_{ad} = \sum_{b \in B} \sum_{\gamma \in \Gamma} w_\gamma \cdot f(d_a, \Delta d_a, \text{ABU}_\gamma) \tag{6a}$$

$$c_{pd} = \sum_{b \in B} \sum_{\gamma \in \Gamma} w_\gamma \cdot f(d_p, \Delta d_p, \text{APU}_\gamma) \tag{6b}$$

$$c_d = 0.5 \times \left(\frac{c_{ad}}{d_t^a} + \frac{c_{pd}}{d_t^p} \right) \tag{6c}$$

$$f(d, \Delta d, \overline{d}) = \begin{cases} \frac{\Delta d}{d}, & \text{if } d + \Delta d \geq \overline{d} \\ 0, & \text{otherwise} \end{cases} \tag{6d}$$

where Δd_a and Δd_p denote the area and pin density increase in each bin, and d_t^a and d_t^p denote the target area and pin density for the layout, respectively. Function f computes the density cost and the cost only happens when the new density exceeds the average density of the top γ% bins. Although the weights for c_{ad} and c_{pd} can be adjusted for different targets, we set them equal in the experiment for simplicity.

The overlap cost c_{ov} is defined as the total area of overlapped cells times the total number of pins divided by row height. As the overlapped cells need to be inserted to the cell pool which results in the inaccuracy of wirelength cost computation, fewer pins are preferred for less contribution to the wirelength cost.

There are some hard constraints for a candidate position that lead to invalidate this candidate. Each overlapped cell must be no larger than current cell; otherwise, it is even more difficult to find legal positions for those overlapped cells. The overlapped cells must not be moved yet in current pass, because each cell can only move once within each pass of iteration.

7) Various Optimization Modes: The chain move algorithm can be configured for various modes, such as overlap reduction, wirelength minimization and density recovery. For overlap reduction mode, the main difference lies in the function `BacktraceToBestEntry` which will not be called in overlap reduction mode, because we observe that applying all the chain moves removes most of the overlaps regardless of potential wirelength degradation. Empirically we often still get some wirelength improvements. In this mode, there is an additional part of displacement cost added to (5). The purpose of the displacement cost is to reduce the perturbation to the global placement solution.

In wirelength minimization mode, we also perform local clustering of horizontally abutting cells in every odd iteration. For any pair of single-row height cells c_i and c_j which horizontally abut to each other, if they share at least one net and either of them is located at the boundary of the bounding box of the shared net, we cluster them and merge their nets into the new cluster, as the bounding box of the shared net is likely to achieve further reduction by moving both cells together. The process starts from scanning cells from left to right in each placement row and an existing cluster is also allowed to merge with another cell to form a larger cluster. We avoid any cluster which involves more than five cells because large clusters are typically difficult to find available locations without large perturbation to other cells. The clustering scheme is performed in alternative iterations (e.g., every odd iteration) because we expect in every even iteration the flat chain moves to perturb the potential clustering solutions which avoids to fall into local optimum quickly. This is inspired by the coarsening and uncoarsening scheme in partitioning algorithms like hMetis [24]. After chain move iterations, we fix multiple-row height cells and perform conventional global move to single-row height cells for further wirelength improvements. This incremental step usually converges in one or two iterations.

In density recovery mode, we perform the chain move algorithm on cells in those densest bins (e.g., top 20% dense bins) and increase the weight of density cost in (5) (i.e., $\alpha = 10$ and $\beta = 2$). The function `ComputeSearchRegion` returns a large region centered by the current position of the cell instead of computed from its optimal region which is usually congested. In the function `BacktraceToBestEntry`, we allow small amount of wirelength degradation (e.g., 0.5% in the experiment) for density improvement.

B. Ordered Double-Row Placement

The ordered single-row placement has been well explored in detailed placement for wirelength minimization and legalization [17], [20], [25]–[28]. There are also many single-row algorithms designed for manufacturability compliance, such as multiple patterning lithography, FinFET process, and E-beam lithography [29]–[38]. The problem can be formulated into a dual min-cost flow problem that can be solved in $\mathcal{O}(n^2 \log m^2)$ time complexity for wirelength minimization [25], where n is the number of cells in a row and m is the number of nets involved. The runtime is reduced to $\mathcal{O}(m \log m)$ by the clumping algorithm from [26]. If each cell in a row has a maximum displacement M, the problem can be transferred to a shortest path problem and a DP-based algorithm is able to solve the problem in $\mathcal{O}(M^2 n)$ [20], [33]. It can be further improved to $\mathcal{O}(Mn)$ by exploiting the monotonicity and pruning the solution space [37]. However, most of these algorithms only focus on single-row placement and are not able to deal with multiple-row height cells. Here we define an ODR placement problem as follows.

Problem 2 (ODR Placement): Given two rows of cells that are ordered from left to right in each row, horizontally move the cells to optimize HPWL without ruining the order of cells in each row.

Please note that the two sequences of cells may contain multiple-row height cells, shown in Fig. 5. Here are several definitions to the cells in the double-row placement problem.

Definition 4 (Double-Row Region R_{dr}): The rectangular region defined by the target two rows to be solved.

The target rows to be solved by double-row placement form a rectangular box. The region defined by the other rows will be referred to as a region outside the double-row region, denoted by $\overline{R}_{dr}$.

Definition 5 (Splitting Cell): Any multiple-row height cell spans both rows in R_{dr}.

In Fig. 5, cells e and i cover both lower and upper row in R_{dr}, so they are considered as splitting cells.

Definition 6 (Crossing Cell): Any multiple-row cell spans only one of the two rows in R_{dr}.

Cells like g and j in Fig. 5 either take the lower or upper row in R_{dr}, and also intersect with $\overline{R}_{dr}$. They are considered as crossing cells.

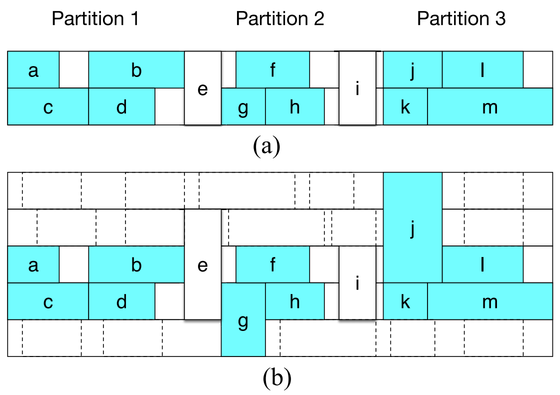


Fig. 5. Example of an (a) ideal case in ODR placement and a (b) general case with large splitting cells and crossing cells such as cells e and j.

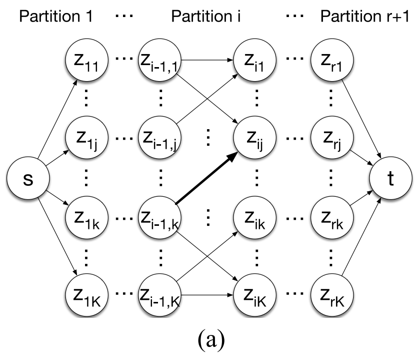
TABLE II
NOTATIONS IN ODR PLACEMENT

M	Maximum displacement for a cell.
d_i	The displacement of cell c_i, $-M \leq d_i \leq M$.
z_i	A splitting cell in the splitting cell set SC.
y_i	A crossing cell in the crossing cell set CC.
v_i	A single-row height cell or crossing cell in the lower row of a partition.
u_i	A single-row height cell or crossing cell in the upper row of a partition.
PC_i	The set of cells in the partition between splitting cell z_{i-1} and z_i.

There are several cases to this problem. The ideal case is that the double-row placement problem only consists of single-row height cells and double-row height splitting cells, which means all the cells will lie in R_{dr}, shown in Fig. 5(a). Two splitting cells e and i separate the each row into three parts, i.e., partitions 1–3. But this is not often true due to the existence of crossing cells and large splitting cells. Fig. 5(b) gives a general case for the double-row placement problem where some splitting cells and crossing cells span more than two rows. In this case where cells e, g, and j spread out of the rows, their movements must keep the order within the two rows and not cause any overlap in the other rows. We will first explain the algorithm with the ideal case in Fig. 5(a) and extend it to handle the general cases. For simplicity, we further assume in the ideal case, there is no inter-row connection between cells in the lower and upper row within each partition. The general double-row placement problem without ordering constraints is very difficult, since the general single-row placement problem is already known as $\mathcal{NP}$-hard [39].

1) Nested Shortest Path Problem: We first formulate the ODR placement problem into a nested shortest path problem with outer and inner level. Then we solve it with a nested DP algorithm. Table II gives the notations used in the ODR placement problem. We define the maximum displacement M such that each cell has $K = 2M + 1$ displacement values. Let z_{ij} denote the jth position for splitting cell z_i. Let r be the number of splitting cells in R_{dr}, b be the number of cells in the lower row of a partition, and t be the number of cells in the upper row of a partition.

The key observation to the ODR placement problem is the independence of subproblems within each partition providing the positions of splitting cells fixed. For instance, the subproblem for cells in partition 1 of Fig. 5(a) becomes



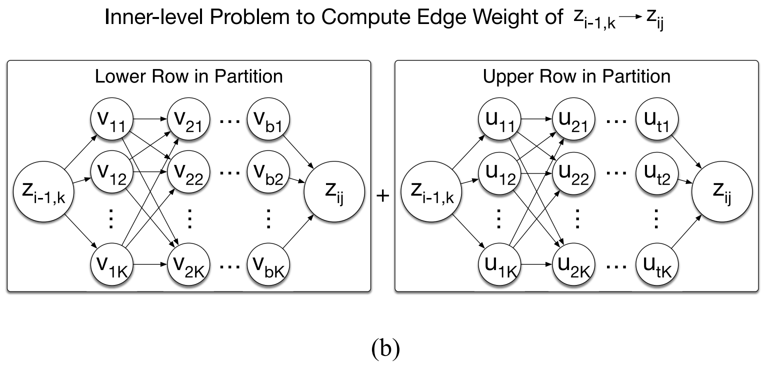

Inner-level Problem to Compute Edge Weight of $z_{i-1,k} \rightarrow z_{ij}$
Lower Row in Partition
Upper Row in Partition
(b)

Fig. 6. (a) Outer-level shortest path problem that solves the positions of splitting cell $z_1, z_2, \ldots, z_r$. The weights of edges in each partition need to be computed by solving the inner-level problems. (b) Inner-level problem computes the edge weight of $z_{i-1,k} \rightarrow z_{ij}$ by solving the shortest path problem of lower and upper row in the partition with given positions of the splitting cells $z_{i-1,k}$ and z_{ij}.

independent as long as the position of splitting cell *e* is determined. Similarly, the subproblem in partition 2 only relies on the positions of splitting cells *e* and *i*. Therefore, if we can determine the positions of the splitting cells, it is possible to solve the corresponding independent subproblem. With such observation, we formulate a nested shortest path problem shown in Fig. 6, where we solve the positions of all the splitting cells with an outer-level shortest path problem whose edge weights are determined by a set of inner-level problems.

Fig. 6(a) gives the graph representation of the outer-level shortest path algorithm where each node denotes a candidate position of a splitting cell. We need to find the shortest path from *s* to *t*. However, the weights of edges in Fig. 6(a) are not determined yet because the minimum placement cost for cells within each partition is still unknown. With the previous independence property, we can compute the weight of any edge $z_{i-1,k} \rightarrow z_{ij}$ by solving the inner-level problem shown in Fig. 6(b). The inner-level problem consists of two shortest path problems for the lower and upper row in the partition. These two shortest path problems are independent due to the assumption in ideal case that there is no inter-row connection in a partition. Node $z_{i-1,k}$ and z_{ij} serve as the starting and terminating node in the inner-level problem.

2) Nested Dynamic Programming: In general any algorithm that solves shortest path can be applied to the nested shortest path problem defined above. For efficiency, we adapt the DP algorithm in [37] to solve the nested shortest path problem in the ODR placement, which results in a nested DP scheme. Algorithm 2 gives the skeleton of the nested DP algorithm. To highlight the nesting scheme, we omit the details that are the same as the ordered single-row placement and only keep the simplified key steps. The algorithm calls the function `SolveOuterLevel` to solve the outer-level shortest path problem. The kernel procedure of `SolveOuterLevel` lies in the three loops from lines 7 to 15. The cost of each candidate position is evaluated in lines 10–12 where function `ComputeDPCost` computes the cost for z_{i-1} and z_{ij} themselves and function `SolveInnerLevel` solves the inner-level problem for cost in the partition. Within a partition, `SolveInnerLevel` computes the cost of lower and upper row separately with the cost function `ComputeDPCost` and return the total cost. Since the DP for the inner-level problem is the same as single-row version in the ideal case, the details are omitted.

Algorithm 2 ODR Placement

Require: Two ordered sequences of cells.
Ensure: Shift cells to minimize wirelength.

```
 1: ... // prepare data $SC$
 2: SolveOuterLevel($SC$);
 3: return
 4:
 5: function SolveOuterLevel($SC$)
 6:     ...
 7:     for each $z_i \in SC$, $i \leftarrow 2$ to $r$ do
 8:         for each $d_i \in [-M, M]$ do
 9:             for each $d_{i-1} \in [-M, M]$ do
10:                 $cost_i(d_i)$ ← ComputeDPCost($d_{i-1}, d_i$)
11:                         +SolveInnerLevel($d_{i-1}, d_i, PC_i$);
                                              ▷ process $cost_i(d_i)$ in DP
12:             end for
13:         end for
14:     end for
15:     ... // apply solution
16: end function
17:
18: function SolveInnerLevel($d_{i-1}, d_i, PC_i$)
19:     $cost_1$ ← solve DP for lower row in $PC_i$;
20:     $cost_2$ ← solve DP for upper row in $PC_i$;
21:     return $cost_1 + cost_2$;
22: end function
```

The wirelength cost computed in `ComputeDPCost` adopts the cost function defined in [18] for single-row placement. If a cell c_i connects to another cell c_j in the same row and c_j is on the left of c_i, we assume the position of c_j is on the left boundary of the row for wirelength cost computation; if c_j is on the right of c_i in the same row, the position of c_j is assumed to be the right boundary of the row. For any c_j in a different row to c_i, its actual position is used. This wirelength cost turns out to be equivalent to HPWL in single-row placement and the equivalence holds in the ideal case of double-row placement as well.

Lemma 2: Algorithm 2 gives optimal solution for the wirelength cost to the ODR placement under the ideal case.

The proof is omitted here due to page limit.

The runtime for Algorithm 2 turns out to be $\mathcal{O}(M^2 n)$ where n is the total number of cells in R_{dr}. Considering the $r+1$ partitions defined by r splitting cells, within each partition PC_i, the lower row contains b_i cells and the upper row contains t_i cells. Assume `ComputeDPCost` takes constant time and $n \gg r$. The DP scheme takes $\mathcal{O}(Mn)$ to solve single-row placement [37]. So solving partition PC_i for one time takes $\mathcal{O}(Mb_i)+\mathcal{O}(Mt_i)$ in `SolveInnerLevel`. The runtime complexity for Algorithm 2 can be computed as follows:

$$\begin{aligned}\text{complexity} &\approx \sum_{i=1}^{r+1} M \cdot (\mathcal{O}(Mb_i) + \mathcal{O}(Mt_i)) \\ &= \mathcal{O}\Big(M^2(n-r)\Big) \approx \mathcal{O}\Big(M^2 n\Big). \end{aligned} \quad (7)$$

3) Extension to General Cases: The potential overlaps to $\overline{R}_{dr}$ must be considered due to the existence of large splitting cells and crossing cells in a general case. During the ODR placement, any position of a cell overlapping with any placement site already taken by other cells in $\overline{R}_{dr}$ should be avoided; i.e., assign a very large cost to such positions. We can add a large penalty to a position in `ComputeDPCost` without losing the optimality since such penalty only depends on the position of the cell itself.

However, under a general case, the wirelength cost computed by `ComputeDPCost` in the inner-level problem is no longer always equivalent to HPWL. because a cell in the lower row of a partition may have connection with another cell in the upper row. Such inaccuracy from the wirelength cost usually comes from short inter-row connections, so the overhead is small. Besides wirelength, the nested DP scheme can also be adapted to support other objectives, such as displacement and local congestion.

Although ODR placement can minimize wirelength, it may squeeze the whitespaces in dense regions and result in congestion. To mitigate such side effects, we fix the cells in congested regions and only move cells in low-density regions. In general the algorithm can also be applied to resolve overlaps for legalization, but the computation effort becomes an issue for layouts with large amount of overlaps due to its quadratic relation with maximum displacement. Therefore, we adopt it as an incremental optimization technique for legal designs.

C. Ordered Multiple-Row Placement

Despite various algorithms designed for single-row placement mentioned in Section III-B, the dual network flow formulation [25] is flexible enough to handle multiple rows at the same time for total displacement minimization or wirelength minimization while it is not limited by any constraint from multiple-row height cells. The network flow can be solved by various algorithms for dual min-cost flow with proper graph transformation. Although there are brief theoretical derivations for this formulation [25], [40], its practical insight and details remain to be explored.

We extend the definition of OMR placement from ODR placement problem.

Problem 3 (OMR Placement): Given arbitrary number of rows of cells that are ordered from left to right in each row, horizontally move the cells to optimize total displacement or HPWL without ruining the order of cells in each row.

The formulation of OMR is quite different from ODR. First, supposing that OMR solves R rows simultaneously, it returns the optimal solution of cells within the entire region of R rows. If ODR is used to solve the same region, it needs to run $\lceil (R/2) \rceil$ times. In other words, ODR has to divide the region before solving any region with $R > 2$. When the objective includes wirelength which involves connections of cells in different rows, such division loses optimality even if ODR returns optimal solutions of every two rows. Second, current algorithm for ODR is realized by enumerating discrete displacement sites for each cell in a nested DP scheme. In spite of its flexibility in the objective, its runtime complexity is quadratically related to maximum displacement M, while in the network flow formulation of OMR, the maximum displacement M does not have to appear explicitly in the runtime complexity. We do not need to tradeoff M for runtime at the cost of quality degradation. Third, while ODR returns optimal solutions in ideal case, it loses optimality in general cases as mentioned in Section III-B3. The network flow algorithm can solve OMR optimally even for general cases. Although generally most multiple-row height cells are double-row height cells, OMR is expected to outperform ODR if OMR is affordable in terms of reasonable runtime.

1) Displacement Minimization: The problem to minimize total displacement can be written as the following mathematical program:

$$\mathcal{P}_m\colon \min \sum_{i\in N} \Big| x_i - x_i^0 \Big| \quad (8a)$$

$$\text{s.t. } x_i - x_j \le -w_i, \quad \forall (i,j) \in O \quad (8b)$$

$$l_i \le x_i \le u_i, \quad \forall i \in N \quad (8c)$$

where N denotes the set of cells, O denotes the set of cell pairs in which the order should be maintained without overlap, x_i denotes the horizontal position of cell i, w_i denotes its width, and x_i^0 denotes its original position. The objective in (8a) minimizes total displacement in $\mathcal{L}1$ norm. The constraint (8b) ensures overlap free between horizontally abutting cell i and j with w_i as the width of cell i. The constraint (8c) limits cell i to be within a movable range. We can further remove the absolute operation in the objective by introducing additional variable d_i^l and d_i^r for each cell and add constraints

$$d_i^l - x_i \le 0, \quad \forall i \in N \quad (9a)$$

$$d_i^l \le x_i^0, \quad \forall i \in N \quad (9b)$$

$$x_i - d_i^r \le 0, \quad \forall i \in N \quad (9c)$$

$$d_i^r \ge x_i^0, \quad \forall i \in N \quad (9d)$$

where (9a) and (9b) guarantee that d_i^l is no larger than the smaller one of x_i and x_i^0, and (9c) and (9d) guarantee that d_i^r is no smaller than the larger one of x_i and x_i^0. The objective in (8a) is changed to

$$\min \sum_{i\in N} d_i^r - d_i^l. \quad (10)$$

The bound constraints in (8c), (9b), and (9d) can be converted to differential constraints by introducing a single variable $\bar{x}$ and replace all x_i with $x_i' = x_i + \bar{x}$, all d_i^l with $d_i'^l = d_i^l + \bar{x}$, and all d_i^r with $d_i'^r = d_i^r + \bar{x}$. Then the problem $\mathcal{P}_m(x_i, d_i^l, d_i^r)$ is transformed to problem $\mathcal{P}_m'(x_i', d_i'^l, d_i'^r, \bar{x})$ with differential constraints only as follows:

$$\mathcal{P}_m'\colon \min \sum_{i\in N} d_i'^r - d_i'^l, \quad \forall i \in N \quad (11a)$$

$$\text{s.t. } d_i'^l - x_i' \le 0, \quad \forall i \in N \quad (11b)$$

$$d_i'^l - \bar{x} \le x_i^0, \quad \forall i \in N \quad (11c)$$

$$x_i' - d_i'^r \le 0, \quad \forall i \in N \quad (11d)$$

$$\bar{x} - d_i'^r \le -x_i^0, \quad \forall i \in N \quad (11e)$$

$$x_i' - x_j' \le -w_i, \quad \forall (i,j) \in O \quad (11f)$$

$$l_i \le x_i' - \bar{x} \le u_i, \quad \forall i \in N. \quad (11g)$$

Once problem $\mathcal{P}'_m$ is solved, the solutions to $\mathcal{P}_m$ can be easily derived by deducing $\bar{x}$.

2) Generalized Formulation: We generalize all the variables $x'_i, d'^l_i, d'^r_i, \bar{x}$ in (11a) to π_i, introduce b_i as the coefficient of each variable in the objective, and introduce c_{ij} as the right hand side for each differential constraint. Problem $\mathcal{P}'_m$ in (11a) can be transformed to problem $\mathcal{P}(\pi_i, \alpha_{ij})$ with additional slack variable α_{ij} for each constraint

$$\mathcal{P}\colon\ \min \sum_{i\in N} b_i\pi_i + \sum_{(i,j)\in E} u_{ij}\alpha_{ij} \quad (12a)$$

$$\text{s.t.}\ \ \pi_i - \pi_j - \alpha_{ij} \le c_{ij}, \quad \forall (i,j)\in E \quad (12b)$$

$$s\alpha_{ij} \ge 0, \quad \forall (i,j)\in E \quad (12c)$$

where N represents the set of variable π_i, E represents the set of differential constraints, and u_{ij} is relatively large compared with b_i. For example, to construct (12a) from the objective in (11a), the coefficients for d'^r_i are mapped to $b_i = 1$, while the coefficients for d'^l_i are mapped $b_i = -1$; to construct (12b) from (11f), we set $c_{ij} = -w_i$, and so forth.

While problem $\mathcal{P}$ matches problem $\mathcal{P}'_m$ exactly without variable α_{ij}, the reason of introducing variable α_{ij} lies in the fact that the input placement may not be legal in detailed placement which often results in infeasible models of problem $\mathcal{P}'_m$, it is more meaningful to optimize the objective while minimizing the violations to the constraints. Note that the infeasible model not only comes from regions with utilization larger than 100%, it may also come from the regions with utilization smaller than 100% due to the existence of *dead spaces* introduced by multiple-row height cells [12], [13]. Thus the objective here is to optimize total displacement or HPWL with minimum overlaps between cells. If problem $\mathcal{P}'_m$ is feasible, then $\alpha_{ij} = 0$ in the optimal solution of problem $\mathcal{P}$ due to large u_{ij} and the optimal solutions to both problems are equivalent; otherwise, problem $\mathcal{P}$ is still feasible and return a minimum objective $\sum_{i\in N} b_i\pi_i + \sum_{(i,j)\in E} u_{ij}\alpha_{ij}$ with some nonzero α_{ij} indicating the violations to some differential constraints. If u_{ij} is large enough, such violations can be minimized. For an extreme case, when u_{ij} goes to infinity, problem $\mathcal{P}$ goes unbounded when problem $\mathcal{P}'_m$ is infeasible. In the experiment, we give u_{ij} a large enough value such as twice of the width of a placement row.

The dual problem of problem $\mathcal{P}$ is associated to the min-cost flow problem as follows [40]:

$$\mathcal{D}\colon\ \min \sum_{(i,j)\in E} c_{ij}f_{ij} \quad (13a)$$

$$\text{s.t.}\ \sum_{j:(i,j)\in E} f_{ij} - \sum_{j:(j,i)\in E} f_{ji} = -b_i, \ \forall i \in N \quad (13b)$$

$$0 \le f_{ij} \le u_{ij}, \ \forall (i,j)\in E \quad (13c)$$

where f_{ij} denotes the flow on arc (i,j), c_{ij} denotes the cost of flow, u_{ij} denotes the flow capacity on an arc, and $-b_i$ denotes the supply of vertex i. Fig. 7(a) shows an example of mapping from problem $\mathcal{P}$ to a network flow graph where the variable π_i is associated with the mass balance constraint of vertex i and its solution can be obtained from the vertex potential.

An example of OMR placement and its corresponding network flow graph are shown in Fig. 7(b) and (c) with total displacement minimization. Each cell i introduces three vertices where vertices d^l_i and d^r_i associate with variables d'^l_i and d'^r_i in problem $\mathcal{P}'_m$ (11a). Vertex x_i associates with variable x'_i.

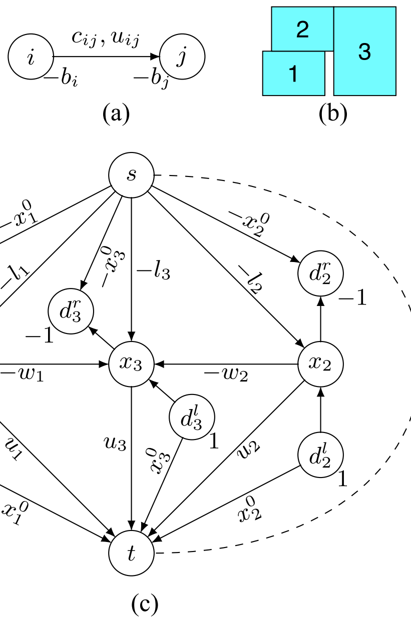


Fig. 7. (a) Mapping problem $\mathcal{P}$ to vertices and arcs in a network flow graph where each vertex has a supply value and each arc has a cost value and a capacity value. Example of (b) three cells for OMR placement problem $\mathcal{P}'_m$ in (11a) and (c) corresponding network flow graph with nonzero vertex supply values and nonzero arc cost values labeled, while arc capacity values are not labeled.

The additional vertex introduced by variable $\bar{x}$ is split into vertices s and t for typical representation of network flow graph. The dashed line between s and t means they can be either merged or kept separate for min-cost flow algorithm. Each differential constraint in (12b) corresponds to an arc with cost c_{ij} and capacity u_{ij}. The nonzero supply values are labeled next to each vertex and the overall supply is zero. Nonzero costs are labeled along with arcs. The capacity u_{ij} of each arc (i,j) is not shown in the figure for brevity. It can be an arbitrary large enough number to avoid violations of constraints in problem $\mathcal{P}$ as aforementioned. By solving the network flow, we can extract the potential of each vertex which is associated with the solution of π_i to problem $\mathcal{P}$.

3) Wirelength Minimization: As mentioned, the OMR placement problem with network flow formulation is also capable of minimizing HPWL

$$\mathcal{P}_h\colon\ \min \sum_{i\in E} r_i - l_i \quad (14a)$$

$$\text{s.t.}\ \ l_i - x_j \le o_j, \quad \forall i \in E, j \in E_i \quad (14b)$$

$$x_j - r_i \le -o_j, \quad \forall i \in E, j \in E_i$$

$$\text{(8b) and (8c)} \quad (14c)$$

where E denotes the set of interconnections and E_i denotes the set of cells in net i. Variables l_i and r_i denote the left and right boundary of the bounding box of net i, respectively. Variable o_j indicates the pin offset of cell j in net i. Similar transformation as above can be applied to construct an optimization problem with differential constraints only such that it can be transformed to the min-cost flow problem.

In general it is expensive to solve the OMR placement for full layout, but we can divide the layout into row chunks where each chunk takes R rows to trade performance for runtime. For multiple-row height cells at the boundary of each chunk, we treat them as fixed. Suppose there are $\hat{R}$ rows in the layout. Then we need to invoke the min-cost flow algorithm for $\lceil(\hat{R}/R)\rceil$ times. Problem $\mathcal{P}_h$ (14a) generalizes and extends the ODR placement problem in Section III-B. It is able to solve more than two rows simultaneously and its runtime complexity is correlated with number of cells and nets rather than maximum displacement M, which indicates potential tradeoffs under different configurations of R.

There are various min-cost flow algorithms like network simplex, cost scaling, capacity scaling, etc. [41], while not all of them support negative costs on arcs. To construct the network flow graph that is compatible to different algorithms, the negative costs can be removed by arc reversal [40] where we can flip the sign of arc cost by adjusting the supply values of its two vertices. With the flexibility of the network flow formulation, the technique can be applied to either legalization stage to remove as much overlap as possible, or post refinement stage to further improve wirelength. While the technique is not limited to constraints from multiple-row height cells, it might suffer from efficiency issues if simply applied to full layout. We demonstrate the performance, efficiency and various tradeoffs of the multiple-row placement in Section IV. In addition, both (11a) and (14a) describe linear programs, so we also compare the efficiency of network flow algorithms with LP algorithms. It needs to mention that since the row-based placement techniques do not change the vertical positions of cells, they follow the power line alignment constraints as long as there is no violation in the input.

IV. Experimental Results

Our algorithm was implemented in C++ and tested on an eight-core 3.40-GHz Linux server with 32-GB RAM. GUROBI [42] is used as the LP solver and LEMON [43] is used as the min-cost flow solver. Single thread is used in the experiment. We validate our algorithm on two sets of benchmarks. The first set of benchmarks are generated from ACM International Symposium on Physical Design (ISPD) 2005 placement benchmark suite by [11] with only single-row and double-row height cells. Double-row height cells are randomly generated from about 30% single-row height cells. The state-of-the-art wirelength-driven global placer POLAR [9] is used for global placement. We obtain the binary from [11] and all the results are collected from our machine. The second set of benchmarks are modified from ICCAD 2014 placement benchmark suite [44] in which we resize cells such as flip-flops to double-row height and some large cells such as NAND4_X4 and INV_X32 to three- and four-row height cells. We adopt the evaluation script from ICCAD 2013 placement contest to verify the legality, wirelength and density of our placement solution. The bin sizes are set to 9×9 row heights according to the evaluation script. The target pin density d_t^p for APU evaluation is set to the average pin density of top 60% densest bins. Benchmarks and programs are released at link (http://www.cerc.utexas.edu/utda/download/MrDP/index.html).

Table III shows the statistics of ISPD 2005 benchmarks and Table IV shows the comparison between our algorithm [11], [45]. The sizes of the designs vary from 200 K to 2 M with utilizations from 67.70% to 91.10%. The ratio of multiple-row height cells are shown as "DH." The wirelength for the input global placement solution is shown as "GP," which is not legalized yet. The results of our algorithm is shown as "MrDP." Runtime is shown as "CPU" in seconds.

TABLE III
ISPD 2005 Benchmark Suite [11]

Design	Size	DH	Util	Target Util
adaptec1	211K	30.18%	90.84%	91%
adaptec2	255K	30.16%	89.12%	90%
adaptec3	452K	30.11%	78.44%	80%
adaptec4	496K	30.19%	67.70%	75%
bigblue1	278K	30.14%	73.44%	80%
bigblue2	558K	32.90%	68.99%	75%
bigblue3	1097K	30.31%	91.10%	91%
bigblue4	2177K	30.26%	73.88%	75%

Since [11] only considers wirelength, we first compare wirelength in which MrDP achieves smaller HPWL in all benchmarks on an average of 1.2%. We can also see from the table that MrDP can achieve even more significant improvement in sHPWL, 3.7% on average, which indicates better cell density in the placement solution. The ABU penalty from MrDP is 20.2% smaller than that from [11] and APU penalty shows 13.4% improvement. Although MrDP is slightly slower than [11], even the largest benchmark with 2 million cells can be finished within 10 min, which is still affordable in placement.

Table V gives experimental results on modified ICCAD 2014 benchmarks. To the best of our knowledge, no published detailed placers are reported to explicitly handle such benchmarks with various multiple-row height cells yet. The ratio of multiple-row height cells varies from 17.17% to 41.09% for different benchmarks, shown as "MH." The average percentage of three-row height cells and four-row height cells is around 0.1%, which indicates most of the multiple-row height cells are double-row height cells. We keep the same target utilizations as the contest setting. The data under "initial" denotes the evaluation of initial solutions that still contain overlaps. We can see that MrDP achieves 3.2% improvement in HPWL and 4.7% improvement in sHPWL. The cell and pin density penalty also decrease by 42.6% and 20.0%, respectively, from initial placement, which is significantly improved from [45]. We ascribe the improvement in density to the OMR placement and density recovery mode of chain move, where the former achieves more wirelength reduction than ODR placement and thus the latter has more margin to smooth density with affordable wirelength increase.

A. HPWL and Runtime Breakdown

Fig. 8 shows the HPWL improvement and runtime breakdown of three benchmarks, *b19*, *leon2*, and *netcard*. The HPWL improvement in Fig. 8(a) is the cumulative normalized improvement after executing each step in the flow. "Chain move WL" denotes the chain move in wirelength minimization mode and "chain move density" denotes the density refinement step. It is shown generally chain move in wirelength minimization mode gives the most of wirelength improvement, which also takes most part of the overall runtime [around 60% in Fig. 8(b)], while OMR can further reduce the wirelength after the convergence of chain move with small runtime overhead [around 10% in Fig. 8(b)]. The performance of OMR varies from benchmark to benchmark; e.g., in benchmark *b19*, it improves the wirelength by around 1.5%, while in benchmark *netcard*, the improvement is only around 0.1%. The density refinement step slightly degrades wirelength for density improvement.

TABLE IV
COMPARISON OF OUR ALGORITHM WITH WU *et al.* [11]

Design	HPWL				sHPWL				ABU penalty			APU penalty			CPU		
	GP	[11]	[45]	MrDP	GP	[11]	[45]	MrDP	[11]	[45]	MrDP	[11]	[45]	MrDP	[11]	[45]	MrDP
adaptec1	95.57	91.35	91.03	91.00	134.45	96.22	96.88	96.66	0.0533	0.0642	0.0622	0.8943	0.7668	0.7724	38.3	44.2	43.9
adaptec2	105.75	105.66	104.07	104.14	121.10	107.40	105.68	104.94	0.0165	0.0154	0.0077	2.2661	2.0553	2.0383	42.5	48.6	49.7
adaptec3	241.83	242.13	237.69	237.94	305.53	273.94	267.20	265.51	0.1314	0.1242	0.1159	2.6111	2.2966	2.3002	82.8	84.7	87.9
adaptec4	206.81	208.92	204.94	205.12	279.16	253.97	240.62	238.33	0.2156	0.1741	0.1619	2.4462	2.0664	2.0571	83.6	88.6	92.7
bigblue1	116.95	113.09	112.48	112.68	134.39	133.34	127.03	124.28	0.1791	0.1293	0.1029	0.5442	0.3683	0.3625	36.9	52.6	56.0
bigblue2	159.59	160.86	158.11	158.15	230.82	197.11	190.54	189.58	0.2253	0.2051	0.1987	1.2189	1.0881	1.0916	78.3	101.1	101.0
bigblue3	413.75	418.69	412.01	411.73	499.20	431.31	428.86	428.17	0.0301	0.0409	0.0399	1.9053	1.7502	1.7494	224.9	264.9	264.6
bigblue4	881.32	882.51	876.84	877.40	1166.86	1099.14	1049.69	1040.15	0.2455	0.1971	0.1855	0.9599	0.7562	0.7521	322.3	438.1	478.0
avg.	277.69	277.90	274.65	274.77	358.94	324.05	313.31	310.95	0.1371	0.1188	0.1093	1.6057	1.3935	1.3904	113.7	140.4	146.7
ratio	1.000	1.001	0.989	0.989	1.000	0.903	0.873	0.866	1.000	0.867	0.798	1.000	0.868	0.866	1.000	1.235	1.291

TABLE V
EXPERIMENTAL RESULTS ON MODIFIED ICCAD 2014 BENCHMARKS

Design	Size	MH %	Util %	Target Util%	HPWL			sHPWL			ABU			APU			CPU	
					Initial	[45]	MrDP	Initial	[45]	MrDP	Initial	[45]	MrDP	Initial	[45]	MrDP	[45]	MrDP
vga_lcd	165K	21.99	54.98	70	4.19	4.02	4.02	4.39	4.20	4.15	0.0471	0.0444	0.0331	0.1310	0.1184	0.1090	27.3	26.7
b19	219K	27.93	52.56	76	3.32	3.17	3.14	3.47	3.27	3.20	0.0440	0.0295	0.0171	0.2750	0.2247	0.1889	31.9	34.3
leon3mp	650K	35.61	51.78	70	15.19	14.34	14.31	15.76	14.52	14.36	0.0377	0.0121	0.0040	0.1623	0.1107	0.1170	261.4	248.9
leon2	795K	41.09	59.82	70	31.97	31.32	31.22	33.88	32.94	32.53	0.0595	0.0517	0.0421	0.1087	0.0779	0.0661	405.8	393.1
dist	133K	26.34	65.23	75	5.06	4.81	4.82	5.06	4.81	4.82	0.0000	0.0000	0.0000	0.1704	0.1203	0.1199	17.0	17.8
mult	160K	14.81	60.14	65	2.95	2.76	2.76	3.08	2.84	2.84	0.0427	0.0300	0.0271	0.1224	0.1547	0.1583	20.7	20.9
netcard	961K	17.17	47.43	72	41.13	40.23	40.25	43.18	42.24	41.77	0.0498	0.0499	0.0379	0.1441	0.1364	0.1322	296.1	288.4
avg.	-	-	-	-	14.83	14.38	14.36	15.54	14.97	14.81	0.0401	0.0311	0.0230	0.1591	0.1347	0.1274	151.4	147.2
ratio	-	-	-	-	1.000	0.970	0.968	1.000	0.963	0.953	1.000	0.775	0.574	1.000	0.847	0.800	1.00	0.97

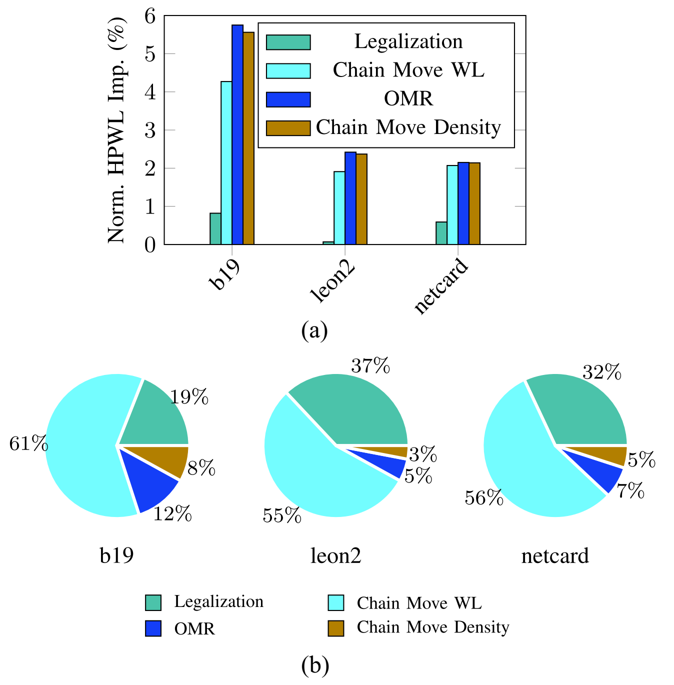


Fig. 8. HPWL and runtime breakdown of benchmarks *b19*, *leon2*, and *netcard*.

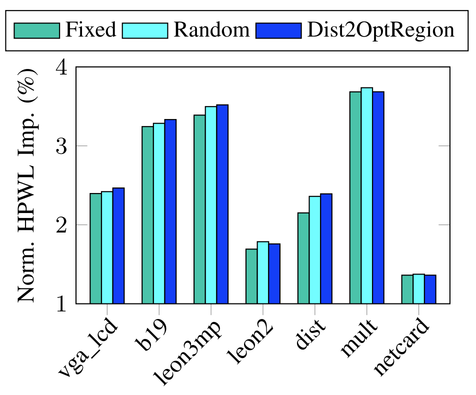


Fig. 9. Comparison between various ordering strategies in chain move on ICCAD 2014 benchmarks. "Fixed" means no shuffling during each iteration of chain moves. "Random" means the visiting order of cells is randomly shuffled during each iteration. "Dist2OptRegion" means cells are ordered in descending order of the distances to the optimal regions.

B. Visiting Order of Cells in Chain Move

Fig. 9 shows the comparison of various visiting order of cells in chain move mentioned in Section III-A4. We can see that it is not a good strategy to fix the visiting order of cells, while random shuffling or sorting by distances to optimal regions of cells gives better wirelength. The results indicate that it is better to perturb the visiting order of cells in each iteration for convergence to better wirelength.

C. Tradeoffs in Ordered Double-Row Placement

We also study the tradeoff between performance and runtime for different maximum displacement M in ODR in Fig. 10. With the increase of M, wirelength drops while the runtime rises quadratically. The wirelength starts to saturate after M goes larger than 8. To tradeoff runtime and performance, we set M to 8 placement sites in the experiment.

D. Tradeoffs in Ordered Multiple-Row Placement

Although ODR is able to improve wirelength efficiently, it is limited to solve two rows at a time. On the other hand, the network flow formulation in Section III-C is able to solve multiple rows simultaneously. We compare the wirelength and runtime between ODR and OMR in Fig. 11. Since the solution space of ODR is related to the maximum displacement M, we try various M values. The row chunk size R for OMR is set to 18. In the experiment, OMR can on average achieve 3.6× HPWL improvement than ODR with $M = 8$ and 3.4× HPWL improvement than that with $M = 16$, while the average runtime for OMR is comparable to ODR with $M = 16$.

Fig. 12 gives the tradeoffs between wirelength and runtime for R. With the increase of R, the wirelength drops while the

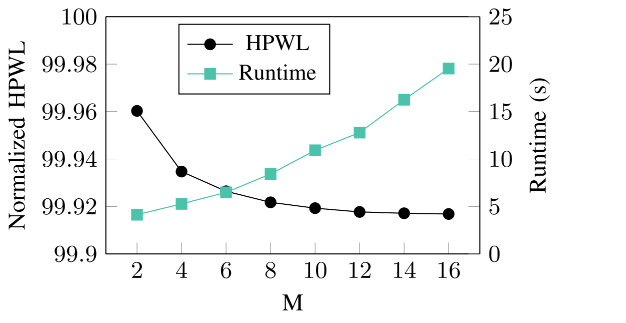


Fig. 10. HPWL and runtime tradeoffs for M in ODR on benchmark *leon2*.

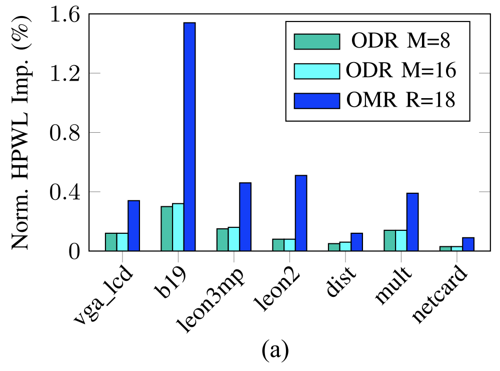
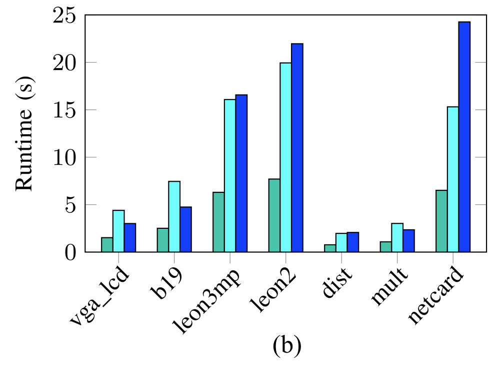



Fig. 11. Comparison between ODR (maximum displacement $M = 8$ and $M = 16$) and OMR ($R = 18$) on ICCAD 2014 benchmarks. (a) HPWL improvement. (b) Runtime.

runtime almost increase linearly. We choose $R = 18$ with affordable runtime and reasonable wirelength improvement. With the HPWL improvement from OMR, there is more margin for the follow-up density recovery step in Section III-A7 to improve density while allowing slight wirelength degradation, which explains the improvements of ABU and APU from [45] in Tables IV and V.

The comparison of runtime between various min-cost flow algorithms and LP is shown in Fig. 13. The efficiency of min-cost flow algorithms varies from problem to problem. Previous study shows that cost scaling algorithm in general is suitable to large graph with relatively low degree, while network simplex algorithm is suitable to small graph with high degree [41]. In our experiment, network simplex is the most efficient for OMR among all the algorithms including LP (on average 2.2× slower). The vertices with large degree are probably from vertices to denote left and right boundaries of nets which involve in a lot of cells. The capacity scaling algorithm turns out to be the slowest (on average 209.0× slower than network simplex) due to large number of arcs and large capacity values

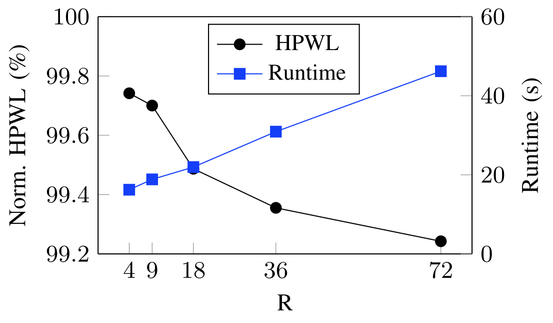


Fig. 12. Trend of HPWL and runtime with R for OMR based on the results of benchmark *leon2*.

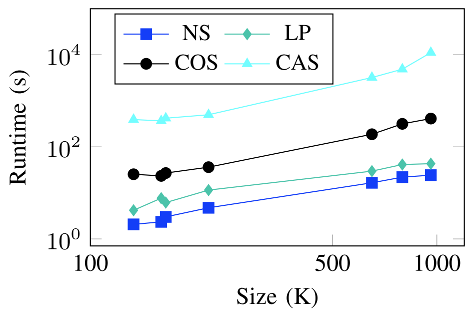


Fig. 13. Runtime comparison between various min-cost flow algorithms and LP to solve OMR with $R = 18$ on various sizes of ICCAD 2014 benchmarks. NS: network simplex; LP: linear programming; COS: cost scaling; and CAS: capacity scaling. Both axes are in log scale for easier analysis.

on arcs. Our experiments actually show almost linear correlation between the runtime of network simplex and the sizes of benchmarks. Therefore, we adopt network simplex algorithm to solve the min-cost flow problem.

It needs to mention that our objective aims at wirelength and density optimization under given maximum displacement, while the problem in [4] tries to remove overlaps and minimize total displacement. Our techniques often end up with better wirelength than [4], but larger total displacement, due to different objectives.

V. Conclusion

In this paper, we have addressed the placement challenges in advanced technology nodes and proposed a detailed placer for heterogeneous-sized cells to help resolve these challenges. Three major techniques have been introduced to generalize the optimization of both single-row height cells and multiple-row height cells, including a chain move scheme to find maximum prefix sum of wirelength improvement, a nested DP algorithm for double-row placement, and a network flow-based algorithm for multiple-row placement. Experimental results demonstrate our algorithm outperforms the most recent detailed placer for multiple-row height cells in both wirelength and density.

References

[1] M. P.-H. Lin, C.-C. Hsu, and Y.-T. Chang, "Recent research in clock power saving with multi-bit flip-flops," in *Proc. IEEE Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Seoul, South Korea, 2011, pp. 1–4.

[2] C.-C. Tsai, Y. Shi, G. Luo, and I. H.-R. Jiang, "FF-bond: Multi-bit flip-flop bonding at placement," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, Stateline, NV, USA, 2013, pp. 147–153.

[3] C.-C. Hsu, Y.-C. Chen, and M. P.-H. Lin, "In-placement clock-tree aware multi-bit flip-flop generation for power optimization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2013, pp. 592–598.

[4] W.-K. Chow, J. Kuang, X. He, W. Cai, and E. F. Y. Young, "Cell density-driven detailed placement with displacement constraint," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, Petaluma, CA, USA, 2014, pp. 3–10.

[5] T.-C. Chen, T.-C. Hsu, Z.-W. Jiang, and Y.-W. Chang, "NTUplace: A ratio partitioning based placement algorithm for large-scale mixed-size designs," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, San Francisco, CA, USA, 2005, pp. 236–238.

[6] N. Viswanathan, M. Pan, and C. Chu, "FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, Yokohama, Japan, 2007, pp. 135–140.

[7] M.-C. Kim, D.-J. Lee, and I. L. Markov, "SimPL: An effective placement algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 1, pp. 50–60, Jan. 2012.

[8] M.-C. Kim, N. Viswanathan, C. J. Alpert, I. L. Markov, and S. Ramji, "MAPLE: Multilevel adaptive placement for mixed-size designs," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, Napa, CA, USA, 2012, pp. 193–200.

[9] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev, "POLAR: A high performance mixed-size wirelengh-driven placer with density constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 3, pp. 447–459, Mar. 2015.

[10] J. Lu *et al.*, "ePlace-MS: Electrostatics-based placement for mixed-size circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 5, pp. 685–698, May 2015.

[11] G. Wu and C. Chu, "Detailed placement algorithm for VLSI design with double-row height standard cells," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 9, pp. 1569–1573, Sep. 2016.

[12] W.-K. Chow, C.-W. Pui, and E. F. Y. Young, "Legalization algorithm for multiple-row height standard cell design," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, 2016, pp. 1–6.

[13] C.-H. Wang *et al.*, "An effective legalization algorithm for mixed-cell-height standard cells," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, 2017, pp. 450–455.

[14] C.-Y. Hung, P.-Y. Chou, and W.-K. Mak, "Mixed-cell-height standard cell placement legalization," in *Proc. ACM Great Lakes Symp. VLSI (GLSVLSI)*, Banff, AB, Canada, 2017, pp. 149–154.

[15] J. Chen, Z. Zhu, W. Zhu, and Y.-W. Chang, "Toward optimal legalization for mixed-cell-height circuit designs," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, 2017, pp. 1–6.

[16] Y. Lin, B. Yu, and D. Z. Pan, "Detailed placement in advanced technology nodes: A survey," in *Proc. IEEE Int. Conf. Solid-State Integr. Circuit Technol. (ICSICT)*, Hangzhou, China, 2016, pp. 836–839.

[17] U. Brenner and J. Vygen, "Faster optimal single-row placement with fixed ordering," in *Proc. IEEE/ACM Design Autom. Test Eurpoe (DATE)*, Paris, France, 2000, pp. 117–121.

[18] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2005, pp. 48–55.

[19] S. Popovych *et al.*, "Density-aware detailed placement with instant legalization," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2014, pp. 1–6.

[20] T. Taghavi *et al.*, "New placement prediction and mitigation techniques for local routing congestion," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2010, pp. 621–624.

[21] M.-C. Kim, N. Viswanathan, Z. Li, and C. Alpert, "ICCAD-2013 CAD contest in placement finishing and benchmark suite," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2013, pp. 268–270.

[22] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, Feb. 1970.

[23] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Las Vegas, NV, USA, 1982, pp. 175–181.

[24] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Anaheim, CA, USA, 1997, pp. 526–529.

[25] J. Vygen, "Algorithms for detailed placement of standard cells," in *Proc. IEEE/ACM Design Autom. Test Eurpoe (DATE)*, Paris, France, 1998, pp. 321–324.

[26] A. B. Kahng, P. Tucker, and A. Zelikovsky, "Optimization of linear placements for wirelength minimization with free sites," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, Hong Kong, 1999, pp. 241–244.

[27] A. B. Kahng, I. L. Markov, and S. Reda, "On legalization of row-based placements," in *Proc. ACM Great Lakes Symp. VLSI (GLSVLSI)*, Boston, MA, USA, 2004, pp. 214–219.

[28] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Abacus: Fast legalization of standard cell circuits with minimal movement," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, Portland, OR, USA, 2008, pp. 47–53.

[29] B. Yu *et al.*, "Design for manufacturability and reliability in extreme-scaling VLSI," *Sci. China Inf. Sci.*, vol. 59, pp. 1–23, Jun. 2016.

[30] J.-R. Gao, B. Yu, R. Huang, and D. Z. Pan, "Self-aligned double patterning friendly configuration for standard cell library considering placement," in *Proc. SPIE*, vol. 8684. San Jose, CA, USA, 2013, Art. no. 868406.

[31] H. Tian, Y. Du, H. Zhang, Z. Xiao, and M. D. F. Wong, "Triple patterning aware detailed placement with constrained pattern assignment," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2014, pp. 116–123.

[32] J. Kuang, W.-K. Chow, and E. F. Y. Young, "Triple patterning lithography aware optimization for standard cell based design," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2014, pp. 108–115.

[33] B. Yu *et al.*, "Methodology for standard cell compliance and detailed placement for triple patterning lithography," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 5, pp. 726–739, May 2015.

[34] H.-A. Chien, Y.-H. Chen, S.-Y. Han, H.-Y. Lai, and T.-C. Wang, "On refining row-based detailed placement for triple patterning lithography," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 5, pp. 778–793, May 2015.

[35] Y. Lin, B. Yu, B. Xu, and D. Z. Pan, "Triple patterning aware detailed placement toward zero cross-row middle-of-line conflict," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 7, pp. 1140–1152, Jul. 2017.

[36] Y. Du and M. D. F. Wong, "Optimization of standard cell based detailed placement for 16 nm FinFET process," in *Proc. IEEE/ACM Design Autom. Test Europe (DATE)*, Dresden, Germany, 2014, pp. 1–6.

[37] Y. Lin *et al.*, "Stitch aware detailed placement for multiple E-beam lithography," *Integr. VLSI J.*, vol. 58, pp. 47–54, Jun. 2017.

[38] W. Ye *et al.*, "Placement mitigation techniques for power grid electromigration," in *Proc. IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, Taipei, Taiwan, 2017, pp. 1–6.

[39] S. Chowdhury, "Analytical approaches to the combinatorial optimization in linear placement problems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 8, no. 6, pp. 630–639, Jun. 1989.

[40] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Beijing, China: Pearson, 2005.

[41] Z. Király and P. Kovács, "Efficient implementations of minimum-cost flow algorithms," *CoRR*, vol. abs/1207.6381, 2012. [Online]. Available: http://arxiv.org/abs/1207.6381

[42] *Gurobi Optimizer Reference Manual*, Gurobi Optim. Inc., Houston, TX, USA, 2016. [Online]. Available: http://www.gurobi.com

[43] *LEMON*. Accessed: Sep. 30, 2015. [Online]. Available: http://lemon.cs.elte.hu/trac/lemon

[44] M.-C. Kim, J. Hu, and N. Viswanathan, "ICCAD-2014 CAD contest in incremental timing-driven placement and benchmark suite," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2014, pp. 361–366.

[45] Y. Lin *et al.*, "MrDP: Multiple-row detailed placement of heterogeneous-sized cells for advanced nodes," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, 2016, pp. 1–8.

Yibo Lin received the B.S. degree in microelectronics from Shanghai Jiaotong University, Shanghai, China, in 2013. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA.

Mr. Lin was a recipient of the Franco Cerrina Memorial Best Student Paper Award at the SPIE Advanced Lithography Conference 2016, the University Graduate Continuing Fellowship in 2017.

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 four best paper awards at the 2017 International Symposium on Physical Design, the 2016 SPIE Advanced Lithography Conference, the 2013 International Conference on Computer Aided Design, and the 2012 Asia and South Pacific Design Automation Conference 2012, and three ICCAD contest awards in 2012, 2013, and 2015. He has served in the Editorial Board of *Integration, the VLSI Journal* and *IET Cyber-Physical Systems: Theory and Applications*.

Xiaoqing Xu (S'15–M'17) received the B.S. degree in microelectronics from Peking University, Beijing, China, in 2012 and the M.S.E. and Ph.D. degrees in electrical and computer engineering from the University of Texas at Austin, Austin, TX, USA, in 2015 and 2017, respectively.

He is currently a Senior Research Engineer with ARM Research, Austin, TX, USA.

Dr. Xu was a recipient of numerous awards, including the Golden Medal at ACM Student Research Competition at ICCAD 2016, the University Graduate Continuing Fellowship in 2016, the SPIE BACUS Fellowship in 2016, the Best in Session Award at SRC TECHCON 2015, the William J. McCalla Best Paper Award at ICCAD 2013, and the CAD Contest Award at ICCAD 2013.

Jhih-Rong Gao received the B.S. and M.S. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2005 and 2007 respectively, and the Ph.D. degree in electrical and computer engineering from the University of Texas at Austin, Austin, TX, USA, in 2014.

She was an Research and Development Engineer with Synopsys Inc., Hsinchu, Taiwan, from 2007 to 2009. In 2014, she joined Cadence Design Systems Inc., Austin, TX, USA, where she is a Principle Software Engineer researching on improving the algorithms and interactions for placement, routing, and clock tree synthesis.

Dr. Gao was a recipient of the BACUS Photomask Scholarship from SPIE in 2013.

Natarajan Viswanathan received the Ph.D. degree in computer engineering from Iowa State University, Ames, IA, USA, in 2009.

From 2006 to 2016, he was with IBM Research and IBM Systems, Austin, TX, USA. architecting core design automation tools and methodologies used in the design of multiple generations of high-performance microprocessor and ASIC designs. He joined Cadence Design Systems Inc., Austin, TX, USA, in 2016, where he is a Software Architect researching on next-generation solutions for clocking. He has published over 30 refereed conference and journal papers and holds over 20 patent grants in the field of EDA.

Dr. Viswanathan was a recipient of the Best Paper Award at ISPD 2004, the best paper nomination at DAC 2007, two best paper nominations at ISPD 2012 for his work on IC placement, and the ACM SIGDA Technical Leadership Award for his work in organizing worldwide CAD contests. He was the Contest Chair for the ISPD 2011, DAC 2012, and ICCAD 2012 CAD contests on placement. He has served on the Technical Program Committee of major conferences, including DAC, ICCAD, and ISPD. Over the last two years, he has served as the Subcommittee Co-Chair for the back-end design and IP track at DAC.

Wen-Hao Liu received the B.S. and Ph.D. degrees in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2008 and 2013, respectively.

He is currently a Senior Principal Engineer with Cadence Design Systems Inc., Austin, TX, USA. He is the main developer of the next-generation global routing, clock routing, and Steiner tree generation engines used in Cadence's tools. His current research interests include routing, placement, and clock synthesis. He has published 27 papers and holds five patents in the above areas.

Dr. Liu has served on the Technical Program Committee of DAC, ISPD, and ASPDAC in the physical design tracks.

Zhuo Li (S'01–M'05–SM'09) received the B.S. and M.S. degrees in electrical engineering from Xi'an Jiaotong University, Xi'an, China, and the Ph.D. degree in computer engineering from Texas A&M University, College Station, TX, USA, in 2005.

He is currently with Cadence Design Systems, Austin, TX, USA, where he manages a team to deliver high performance clocking solutions for the Innovus Place&Route digital product.

Dr. Li was a recipient of IEEE CEDA Early Career Award in 2013, and was selected to participate in National Academy of Engineering's 21st Annual U.S. Frontiers of Engineering Symposium as one of 89 nations top engineering talent in both academia and industry for ages 30–45 in 2015. He is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. He currently serves as the Designer Track Chair for the IEEE/ACM Design Automation Conference.

Charles J. Alpert (F'05) received the bachelor's degree from Stanford University, Stanford, CA, USA, and the Doctorate degree in computer science from University of California at Los Angeles, Los Angeles, CA, USA, in 1996.

He is currently with Cadence Design Systems, Austin, TX, USA, where he manages the clock team for the Innovus Place&Route digital implementation product.

David Z. Pan (S'97–M'00–SM'06–F'14) is currently the Engineering Foundation Professor with the University of Texas at Austin, Austin, TX, USA. He has published over 280 refereed technical papers, and holds eight U.S. patents. He has graduated over 20 Ph.D. students who are currently holding key academic and industry positions. His current research interests include cross-layer nanometer IC design for manufacturability, reliability, security, physical design, analog design automation, and CAD for emerging technologies.

Prof. Pan was a recipient of a number of awards for his research contributions, including the SRC 2013 Technical Excellence Award, DAC Top 10 Author in Fifth Decade, ASP-DAC Frequently Cited Author Award, and 14 best paper awards. He has served as a Senior Associate Editor for *ACM Transactions on Design Automation of Electronic Systems*, an Associate Editor for a number of other journals. He has served in the Executive and Program Committees of many major conferences, including ASPDAC 2017 Program Chair and ICCAD 2018 Program Chair. He is a fellow of SPIE.