Enhanced Optimal Multi-Row Detailed Placement for Neighbor Diffusion Effect Mitigation in Sub-10nm VLSI

Changho Han, *Member, IEEE,* Andrew B. Kahng, *Fellow, IEEE,*
Lutong Wang, *Student Member, IEEE,* and Bangqi Xu, *Student Member, IEEE*

Abstract*—Layout-dependent effect (LDE) causes variation in device performance as well as mismatch in model-hardware correlation (MHC) in sub-10nm nodes. In order to effectively explore the power-performance envelope for IC design, cell libraries must provide cells with different diffusion heights, leading to neighbor diffusion effect (NDE) due to inter-cell diffusion height change (diffusion *steps*). Special filler cells can protect against *steps* to functional cells, but with non-trivial area overhead. In this work, we develop dynamic programming-based single-row and multi-row detailed placement optimizations that optimally[1] reduce inter-cell diffusion *steps* to mitigate the impacts of NDE. Compared to previous works, our algorithms are capable of exploring richer solution spaces as they support cell flipping, relocating and reordering across cell rows; we also consider cell displacement, flipping and wirelength costs. Notably, to our knowledge, our multi-row dynamic programming-based optimization algorithm is the first to optimally handle inter-row cell relocating and reordering. We also explore various metaheuristic configurations to further improve the solution quality. Last, we develop a timing-aware approach, which is capable of creating intentional *steps* that can potentially improve the drive strength of critical cells.

***Index Terms*—detailed placement, multi-row detailed placement, dynamic programming, neighbor diffusion effect, timing-aware.**

I. Introduction

In advanced technology nodes, device behavior no longer depends on independent geometrical parameters [6]. Due to aggressive device scaling, lithography limitations and process complexity, *layout-dependent effect* (LDE) arises from the proximity of devices, and significantly affects device performance. An important type of LDE is *neighbor diffusion effect* (NDE) [1], where the horizontal spacing between diffusion regions changes the performance of transistors. Figure 1(a) illustrates different diffusion spacing caused by diffusion height changes between four transistors. If the heights of neighboring diffusion regions are different, there is a diffusion *step*, e.g., transistor *T2* has a diffusion *step* to each of *T1* and *T3*.

More specifically, the drive strength (i.e., I_{on}) and the leakage power (i.e., I_{off}) of a transistor fin is a function of the horizontal spacing to the adjacent diffusion regions of the transistor fin. Since NDE changes the electrical characteristics of transistors, it affects the power, performance and area of designs [1]. For example, Figure 1(a) shows the transistor fins A and B with the spacings to their neighboring diffusion area, i.e., d_A and d_B, respectively. As d_A and d_B are different, I_{on} and I_{off} of the two transistor fins are different (e.g., $I_{off}(A) = f(d_A) \neq I_{off}(B) = f(d_B)$) due to the change in V_t [1]. For example, given a single inverter with a diffusion *step* next to the PFET and a diffusion *step* next to the NFET, the impacts to the two devices in combination result in higher leakage.

In this work, we use a bimodal assumption to simplify the NDE problem: for a given transistor, either of two leakage values holds, depending on whether the diffusion region on the nearest neighboring site of the transistor has *full height* (that is, same or larger height), or *less height*, compared to the transistor's diffusion height. The leakage difference for the above two cases is linear with #*steps*, e.g., a diffusion height difference of two steps results in 2× leakage difference compared to that of one step. In a conventional place-and-route flow, intra-cell NDE (i.e., NDE effect within a standard cell) is captured by library characterization since the diffusion shapes within a cell are pre-determined. However, it is difficult to capture inter-cell NDE since neighboring diffusion shapes are determined by detailed placement. Thus, in general, library characterization always assumes existence of a full-height neighboring diffusion region on standard cell boundaries, which causes miscorrelation between the model (i.e., library) and the hardware (i.e., actual diffusion shapes at standard cell boundaries and their device leakage impacts) in a design. Minimizing diffusion *steps* in detailed placement is a key idea toward reduction of model-hardware miscorrelation.

With aggressive device scaling, the diffusion *step* not only causes NDE, but also induces a increase in the process complexity due to the limited resolution of conventional 193i lithography. In advanced nodes, the diffusion shapes of transistors are merged and patterned as a single polygon; the transistors are then separated by using diffusion breaks (which are achieved by applying diffusion cuts) [24], as shown in Figure 1(a). Figure 1(b) illustrates the desired pattern of

C. Han is with Samsung Electronics Co., Ltd., Hwaseong-si, Gyeonggi-do, South Korea (email: changho1.han@samsung.com).

A. B. Kahng is with the Departments of Computer Science and Engineering, and of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA, 92093 USA (email: abk@ucsd.edu).

L. Wang and B. Xu are with the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA, 92093 USA (e-mail: {luw002, bangqixu}@ucsd.edu).

[1]The optimality is in terms of maximum diffusion *step* reduction, for given displacement range, reordering range and cell variants. Additionally, the above range definitions, including the definition of the ordering of cells, depend on assumptions described in Section IV and Section V.

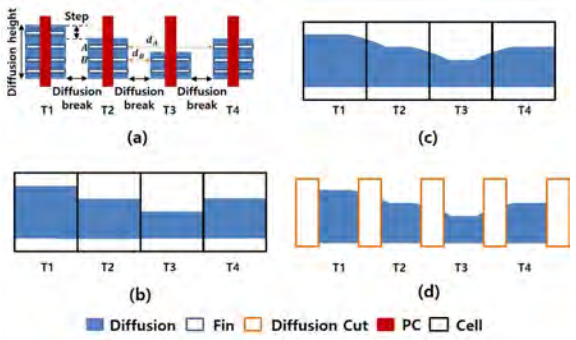

Fig. 1. (a) Diffusion *step* and fin spacing, (b) desired pattern, (c) actual diffusion region showing corner rounding, and (d) diffusion breaks (after diffusion cuts applied).

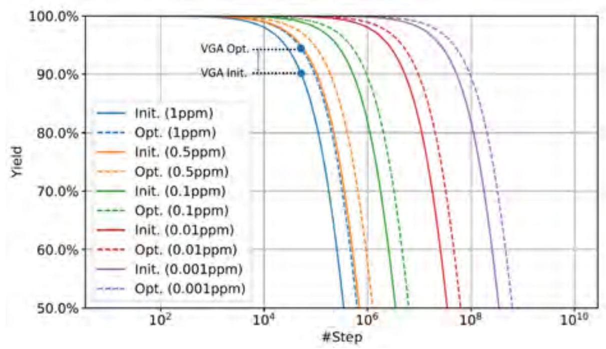

Fig. 2. Initial (Init.) and projected (Opt.) yield assuming 90% inter-cell *step* reduction for various base failure rate.

a single polygon to generate the diffusion regions of four transistors. The actual pattern of the polygon (showing corner rounding in lithography) is shown in Figure 1(c). Figure 1(d) illustrates the final printed diffusion layout with diffusion cuts. At the boundaries of diffusion where diffusion *steps* exist, fin shapes and diffusion shapes are distorted due to the corner rounding phenomena. A distorted and/or sharp-angled end of a fin may cause an increase in electrical field, resulting in gate oxide breakdown [20]. Further, such distorted diffusion shapes change the diffusion height and fin length, which can cause dramatic shifts in threshold voltage (Vt), or even device failure in sub-10*nm* nodes.[2] This Vt shift has negative impact on design performance and quality. For example, Vt variation can cause setup time and/or hold time violations in a design. As a result, the maximum frequency that the design can achieve is reduced, or the design can even fail with hold time violations due to ultra-low Vt which cannot be recovered.

For a motivating study, we define a (inter-cell NDE-induced) *cell failure* to occur if the boundary transistor has a >100mV Vt shift compared to the average Vt for all transistors. According to [28], the failure rate of a transistor with a diffusion *step* is twice as high as a transistor without a diffusion *step* (base failure rate). The solid lines in Figure 2 show the yield vs. (initial) number of diffusion *steps* (∼ #cells) with different base failure rates. We assume #*steps* is approximately proportional to #cells, which holds for testcases in Section VI. The dashed lines in Figure 2 show the projected yield for the same chip if we can reduce 90% of diffusion *steps*. In our preliminary study, more than 60% of standard cells (cell-boundary transistors) have inter-cell diffusion *steps*. For a relatively small design block VGA (85% utilization in an N7 (7*nm* design enablement), 69K cells and 50K diffusion *steps* initially), we assume a base failure rate of 1ppm and can achieve 3.6% yield improvement by removing 90% of diffusion *steps*. For a commercial design with multiple hundreds of millions of cells and diffusion *steps*, if we assume a more realistic 1ppb base failure rate, then we can achieve ∼3% yield improvement by removing 90% of diffusion *steps*.[3] In light of this, minimizing diffusion *steps* helps to recover the yield of designs by reducing Vt (and thus speed) variation of transistors.

Current limitations and our approach. In order to reduce diffusion *steps*, special non-functional *filler cells* are instantiated between functional cells [17] as we elaborate in Section III-A below. However, opportunities for *step*-reducing filler cell insertion are limited given a fixed layout, and this approach (effectively similar to cell padding) is expensive in terms of area. Other works [5] [16][21][26] propose graph-algorithmic or dynamic programming methods to resolve complex design rules in advanced nodes. However, the solution spaces considered are typically limited due to the assumption of (ordered)-single-row placement.[4] Recent works [15][23] on multi-row detailed placement involve heuristic approaches, and no advanced-node rules are considered. Han [7] propose an optimal single-row and double-row dynamic programming for detailed placement optimization, allowing cell reordering with support of double-height cells.

In this paper, we extend our previous single-row and double-row detailed placement framework [7] with HPWL-awareness and with multi-row detailed placement optimization. Our main contributions are summarized as follows.

- We extend the optimal single-row dynamic programming-based approach [7] to an HPWL-aware version. The proposed approach minimizes and balances diffusion *steps* and HPWL cost. Our proposed algorithm is capable of all types of cell movements – i.e., cell variants, relocating, and reordering (i.e., *P-reordering* with $P > 2$).
- We propose a new multi-row dynamic programming, with support of movable, and fully-reorderable, multi-height cells, including reordering between multi-height cells. Inter-row cell moving within each optimization window (in multiple of rows) is intrinsically supported that further improves solution quality.
- We propose metaheuristics to use both single-row HPWL-aware optimization and multi-row optimization to achieve better solution quality.
- We extend our formulation to a potential timing-aware optimization that leads to 6× increase in *intentional* steps around timing-critical cells to improve the timing performance.

[2]According to our collaborator [28], there can be >150mV Vt shift in the 10LPE node.

[3]Based on guidance from our collaborator [28], after scaling to account for our small testcase sizes, we assume a base failure rate of 1ppm for each *step* in our experiments with small design blocks. See Table IV in Section VI.

[4]Lin et al. [16] propose a *P-reordering* problem. However, only *2-reordering* (i.e., neighbor cell switching) is presented. We describe our methodology to handle the *P-reordering* problem in Section III.

- We improve the solution quality over [7] by achieving up to 98% inter-cell diffusion *step* reduction compared to 90% achieved in [7], while consuming similar runtime.

The remainder of this paper is organized as follows. Section II reviews related works. Section III describes the problem formulation and dynamic programming-based single-row detailed placement methodology. Section IV describes the double-row detailed placement flow. Section V describes the multi-row detailed placement flow. In Section VI, we describe our experimental setup and results. Section VII gives conclusions and directions for ongoing work.

II. Previous Work

We classify relevant previous works on detailed placement into three categories: (i) detailed placement for advanced nodes, (ii) mixed cell-height placement, and (iii) NDE-aware detailed placement.

Detailed placement for advanced nodes. To support complex design rules introduced in advanced nodes, the objectives of detailed placement have changed from classical objectives (e.g., wirelength reduction [9][10][11][13][14][19]) in recent years. The works of [16][21][26] resolve triple-patterning issues. Yu et al. [26] propose shortest path and dynamic programming algorithms to solve the ordered single row (OSR) placement. Tian et al. [21] develop a weighted partial MAX SAT approach to solve the OSR problem. Lin et al. [16] propose a local reordered single row refinement (LRSR) and implement a 2-reordering (i.e., neighboring cell switching) approach using a unified graph model. Du and Wong [5] apply a shortest-path algorithm supporting flipping and 2-reordering to address the drain-drain abutment problem in FinFET-based cell placement. The works of [3][8] propose mixed integer linear programming (MILP)-based methods to comply with drain-drain abutment, minimum implant area and minimum oxide jog length rules, and to increase vertical M1 connections.

Mixed cell-height placement. Wu et al. [23] propose a pairing technique to handle double-height cells for detailed placement. Their method simply groups or inflates cells so that all cells become double-height cells, after which a conventional detailed placer can be used. Recently, Lin et al. [15] have proposed a *chain move* scheme along with a nested dynamic programming-based approach to support multiple cell-height placement. They first perform chain moves to save wirelength cost. On top of this, dynamic programming is applied to solve the nested shortest path problem. Other techniques [4] are developed to support non-integer-ratio (e.g., mixture of 8T and 12T cells) mixed cell-height placement.

NDE-aware placement. Ou et al. [18] perform NDE-aware analog placement by modifying and integrating a compact model for NDE into an existing analog placement algorithm. Oh et al. [17] develop special filler cells to mitigate NDE.

Han et al. [7] (which this work builds on) propose to resolve the NDE problem in detailed placement stage. Inter-cell diffusion *steps* are minimized by trying to match the diffusion heights of neighboring cells. If two neighboring cells have different diffusion heights, special filler cells can be inserted to reduce diffusion *steps*. [7] proposes single-row and double-row dynamic programming optimizations that support cell relocating, reordering and flipping as well as double-height cells. They support reordering between single-height cells, and between a single-height cell and a double-height cell, but not between two double-height cells.

In summary, many works such as [5][16][21][26] propose graph or dynamic programming models to resolve complex design rules in advanced nodes. However, their solution spaces are limited by the assumption of (ordered)-single-row placement. Two recent works [15][23] on multi-row detailed placement give heuristic approaches, but no advanced node rules are considered. Our previous work [7] proposes dynamic programming-based methods to optimize single-row and double-row placements, systematically supporting cell reordering and double-height cells. However, the dynamic programming formulation cannot be extended to support more than two rows, and the formulation cannot support reordering between two double-height cells. Notably, our work advances over [7], and is distinguished from previous approaches, in several ways. (i) We formulate an optimal (HPWL-aware) single-row and multi-row dynamic programming-based approach to minimize a cost function that includes diffusion *steps*. (ii) We support a richer set of cell movements than in previous works – i.e., flipping, relocating *and* reordering – via a systematic methodology to handle *P-reordering* with $P > 2$. Specifically, our multi-row approach intrinsically supports *inter-row* cell relocation. (iii) Our formulation supports multi-height cells with movable, and fully-reorderable, multi-height cells.

III. Single-Row Optimization

In this section, we describe the problem statement and our dynamic programming formulation for single-row detailed placement.

Single-Row Optimization Problem. *Given an initial legalized single-row placement, perturb the placement to minimize inter-cell diffusion* steps.

Inputs: A legalized single-row placement, available cell variants, and cost function of a diffusion *step*.

Output: Optimized single-row detailed placement with minimized overall cost (including inter-cell diffusion *steps*).

Constraints: Maximum displacement range, maximum reordering range, availability of cell flipping.

A. Filler Cell and Step Costs

TABLE I
Cost for one diffusion *step*.

Spacing (sites)	0	1	2	3	4+
Cost	1	$+\infty$	1	1	0

Table I describes inter-cell diffusion *step* cost. For each pair of adjacent cells, if there are zero, two or three empty sites in between, the cost is equal to the number of inter-cell diffusion *steps*; if there are at least four empty sites in between, the cost is always zero. That is, with four or more empty sites we can always assume proper filler cell insertions resulting in no inter-cell diffusion *steps*. Figure 3 shows an example of filler cell insertion between two functional cells that have different diffusion heights at edges that face each other. If the two functional cells have fewer than four empty sites in between, filler cells can only match one of the diffusion heights. As a result, there always exists at least one diffusion *step* that affects one of the two functional cells. However, with a spacing of four or more sites, a legal diffusion height transition can always be achieved by one or more contiguous filler cell(s). Thus, the filler cell(s) can match both the diffusion heights of the two functional cells. In a relevant advanced technology, the minimum filler cell width is two placement sites due to process limitations. Therefore, adjacent functional cells must abut, or have at least two empty sites between them, in order to insert a filler cell [28]. In our implementation, we avoid single-site spacings by assigning infinite cost to such scenarios, as indicated in Table I. Even though our optimization does not explicitly allocate white space, the dynamic programming (presented later in Sections III, IV and V) itself can utilize / change the local white space distribution by cell relocating and cell reordering within specified ranges. Filler cell cost is explicitly included in our dynamic programming cost calculation, such that our optimization is aware of both whitespace and filler insertion as it trades off between (i) abutting two cells without filler insertion at the cost of diffusion *steps*, and (ii) leaving four placement sites for a proper filler insertion in an effort to minimize the diffusion *steps* between neighboring cells.

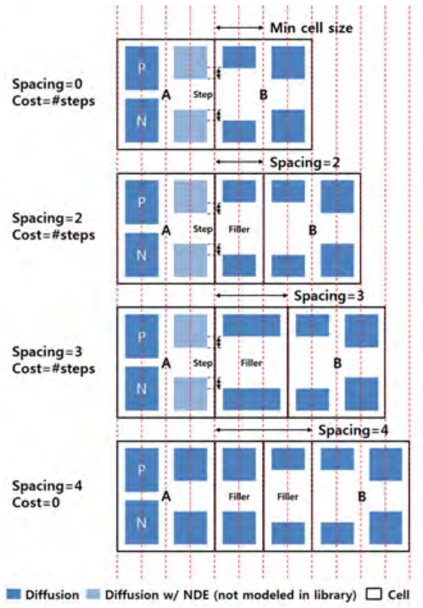

Fig. 3. Filler insertion between cell A and B, given different spacings.

B. Notations

Table II shows notations used in our formulation. For each cell c_k, **cell index** k is its (left-to-right) sequentially ordered position in the initial placement. Given a set of cells (C) in a row of an initial placement, the leftmost cell is c_1, and the rightmost cell is $c_{|C|}$.

For each c_k, we define **cell variants** (v) which correspond to different cell orientations and cell layouts with the same functionality. To minimize #diffusion *steps*, we can use several variants of a cell with the same functionality, for which layouts have different diffusion heights. In our experiments below, $v = 0$ indicates the cell orientation in the initial placement, and $v = 1$ indicates the flipped (i.e., mirrored about the y-axis) cell orientation. $w_{k,v}$ is the width of cell c_k with variant v, in units of placement sites. Flipping a cell does not change the set of sites that the cell occupies.

We define the **displacement range** $[-x_\Delta, x_\Delta]$ as the constraint that a cell cannot move more than x_Δ sites from its initial placement. We use x_k to denote the initial right x-coordinate of c_k, in units of placement sites. Thus, c_k can be placed with its right x-coordinate in the interval $[x_k - x_\Delta, x_k + x_\Delta]$. We use l to denote the **displacement** (in sites) from the initial cell placement (i.e., $l \in [-x_\Delta, x_\Delta]$). For the cells on the boundary of the die, we make sure that the displacement range will not extend beyond the die boundary.

We support cell reordering with a **reordering range** $[-r, r]$, i.e., given r, in the placement solution c_k can have a new sequentially ordered position within the range $k-r, k-r+1, \ldots, k+r$.

In our dynamic programming, we place one cell at a time from left to right, and the index i is used to indicate that i cells have been placed. Given a cell reordering range $[-r, r]$, cells c_k with $k < i - r$ are placed, $i - r \le k \le i + r$ may or may not be placed, and $k > i + r$ are not placed. For the $2r + 1$ cells such that $i - r \le k \le i + r$, we use a binary array s to denote the *placement status* of each cell. Here, s is a binary array of size $(2r + 1)$, i.e., $s \in \{0, 1\}^{2r+1}$. Each bit in the array indicates whether the corresponding cell is

TABLE II
NOTATIONS.

Notation	Meaning
C	set of cells in a window of initial placement
c_k	k^{th} cell in the left-to-right ordered initial placement i.e., k is the cell index
v	a cell variant
$w_{k,v}$	width of c_k with a variant v
$[-x_\Delta, x_\Delta]$	horizontal displacement range
x_k	absolute x-coordinate of c_k in the initial placement, in units of placement sites
l	displacement from the initial placement, in units of placement sites
$[-r, r]$	reordering range
i	number of placed cells
j	position shift from the initial placement
s	placement status array
$d[i][j][v][l][s]$	minimum cost when i cells are placed with case (j,v,l,s)
The notations below apply only to multi-row optimization	
$[-y_\Delta, y_\Delta]$	vertical displacement range
y_k	absolute y-coordinate of c_k in the initial placement, in units of rows
m	number of rows in an optimization window
b	row index in an optimization window
d_b	for the b^{th} row, d_b is the distance between the rightmost boundary of b^{th} row, and the rightmost boundary of all rows in the optimization window
D	distance array of d_b in an optimization window (i.e. $[d_0 \ldots d_{m-1}]$)
t_b	for b^{th} row, type of the rightmost cell (e.g., 2-fin, 3-fin or 4-fin)
T	type array of t_b in an optimization window (i.e., $[t_0 \ldots t_{m-1}]$)
$\{D, T\}$	boundary condition
$[D][T]$	forming boundary condition $\{D, T\}$
$d[i][j][v][l][s]$ $[D][T]$	minimum cost when i cells are placed, forming boundary condition $\{D, T\}$

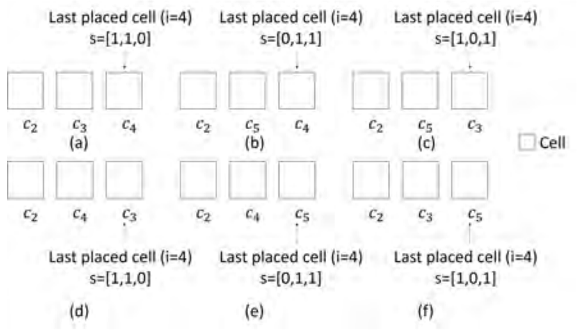

Fig. 4. Illustration of six placement solutions with three legal states given $i = 4$ and $r = 1$.

placed or not. For example, if we have six cells c_1 to c_6, $i = 4$ and $r = 1$, then s captures the placement status of the $(2 \cdot 1 + 1 = 3)$ cells c_3, c_4 and c_5. $s = [0, 1, 1]$ means that c_3 is not placed, while c_4 and c_5 are placed. Figure 4 illustrates six placement solutions with three legal states when $i = 4$. In this example, c_1 and c_2 must be placed and c_6 must not be placed. We note that the indices of s correspond to k (position in the initial placement), but not the final position. For example, $s[0]$ always represents the status for c_3, and $s[2]$ always represents the status for c_5, regardless of the actual sequence of positions, as shown in Figure 4(b). Also, when we have placed i cells, since cells with index $k < i - r$ must be placed, we must have placed $i - (i - r - 1) = r + 1$ cells that have cell index $i - r \leq k \leq i + r$. Thus, at all times, a legal status array s has exactly $r + 1$ elements equal to 1. In the above example, s always has $1 + 1 = 2$ elements equal to 1.

Given i, to identify the last placed cell c_k (that is, the i^{th} cell to have been placed), we define the **position shift** as j, where $k = i + j$. For example, in Figure 4(c), given $i = 4$, the position shift $j = -1$ tells that the last placed cell is c_3, since $3 = 4 + (-1)$.

At the heart of our dynamic programming recurrence, we use $d[i][j][v][l][D][T][s]$ to represent the minimum cost when i cells have been placed. Note that in single-row case, the dimensions of D and T are both zero. Therefore, the dynamic programming array can be reduced to $d[i][j][v][l][s]$. From this array, we can obtain the last placed cell c_k, where $k = i + j$. We can also tell the variant v in use, the displacement l, and the status s for cell c_k. We define the above as *case* (j, v, l, s), with i implicitly given, for simplicity. Therefore, we complete the row placement once we reach $i = |C|$, and we obtain the optimal solution by finding the minimum cost among all cases of $i = |C|$. In our implementation, we store a pointer for each entry in the DP array so that the optimized placement can be traced back from $d[|C|][j][v][l][s]$ all the way to $d[0][j][v][l][s]$.

C. Dynamic Programming Formulation

Algorithm 1 describes our dynamic programming (DP) procedure for single-row placement in detail. Line 2 initializes the DP solution array. Lines 3–13 describe the main algorithm. Starting with placing the first cell, the algorithm incrementally adds (places) cells next to the current partial placement solution. Procedure $getNext()$ returns a list of legal $next$ cells and the respective status of each of these cells. Along with legal (j', s') from Line 5, Line 6 checks all possible cases (v', l') considering placement legality and displacement constraints, as shown in Equation (1). Lines 7–9 update the minimum cost for the case (j', v', l', s') when we place the $i' = (i+1)^{st}$ cell. In Lines 14–17, we obtain the minimum cost among all legal cases when $i = |C|$, and Line 18 returns the minimum cost for the current row.

$$x_{i+j} + l + w_{i+j,v} \leq x_{i'+j'} + l' \tag{1}$$

The function $cost(^{i',j',v',l'}_{i,j,v,l})$ calculates the cost as a weighted sum of (i) diffusion *step* cost, (ii) displacement cost, and (iii) cell variant cost, as shown in Equation (2). The diffusion *step* cost is calculated as total #inter-cell diffusion *steps* between the i^{th} and $(i')^{th}$ placed cells. The displacement cost is equal to the absolute value of l'. In this work, we assume that the given initial placement solution has adequate quality in terms of various metrics, including but not limited to pin accessibility, global routability, etc. Thus, we simplify other optimization objectives as one "displacement minimization" objective. As noted above, in this work we assume two cell variants: original orientation and flipped orientation. We set the variant cost to one if a cell is flipped ($v' = 1$), and zero otherwise. Two weighting factors α and β (β can be seen as supplementing α by capturing an equivalence between cell flipping and displacement) are used to balance the three cost terms. We describe experiments regarding the impact of weighting factors in Section VI.

$$cost(^{i',j',v',l'}_{i,j,v,l}) = cost_{step} + \alpha \cdot cost_{disp} + \alpha \cdot \beta \cdot cost_{var} \tag{2}$$

Algorithm 2 details our methodology to obtain $next$ status. That is, given the binary status array for i, we construct the status array for $i' = i + 1$. Line 2 initializes the list of next available $(cellIndex, status)$ combinations. In Line 3, we first shift the status array for i one bit to the left to obtain the cell placement status for $i' = i+1$. Then, Lines 4–9 check whether cell $c_{i'-r}$ must be placed as the $(i')^{th}$ cell. If we do not place $c_{i'-r}$ as the $(i')^{th}$ cell, then cell $c_{i'-r}$ will be placed out of its reordering range. Thus, we set $s[-r] = 1$ and return so that we make sure to choose $c_{i'-r}$ as the $(i')^{th}$ cell. Lines 10–16 check whether any binary indicator $s[m]$ is equal to zero. If so, $c_{i'+m}$ could be the next legally placed cell. In such a case, we add $(m, nextStatus)$ to the list.

D. HPWL-Aware Optimization

We mitigate the wirelength impact of single-row *step* optimization by modifying the cost function. Specifically, we add a ΔHPWL cost component to the function $cost(^{i',j',v',l'}_{i\ ,j\ ,v\ ,l})$, as shown in Equation (3).

$$cost(^{i',j',v',l'}_{i,j,v,l}) = cost_{step} + \alpha \cdot cost_{disp} + \alpha \cdot \beta \cdot cost_{var} + \gamma \cdot cost_{\Delta HPWL} \tag{3}$$

We calculate the $cost_{\Delta HPWL}$ by summing up the ΔHPWL contribution of cell c_k over all nets incident to c_k, in the same way as in [13]. $cost_{\Delta HPWL}$ captures the impact of a cell's placement on bounding box sizes of incident nets. We use a new weighting factor γ to balance the four cost terms. We describe experiments regarding the impact of weighting factors in Section VI.

Algorithm 1 Dynamic programming (single-row)

```
1: Initialize for all legal cases $(j, v, l, s)$
2:     $d[0][j][v][l][s] \leftarrow 0$, $d[i][j][v][l][s] \leftarrow +\infty$, $(0 < i \le |C|)$
3: for all $i = 0$ to $|C| - 1$ do
4:     for all $d[i][j][v][l][s] \ne +\infty$ do
5:         for all $(j', s') \in$ getNext($s$) do
6:             for all $(v', l')$ do
7:                 $i' = i + 1$
8:                 $t \leftarrow d[i][j][v][l][s] + cost(^{i',j',v',l'}_{i,\ j,\ v,\ l})$
9:                 $d[i'][j'][v'][l'][s'] \leftarrow \min(d[i'][j'][v'][l'][s'], t)$
10:            end for
11:        end for
12:    end for
13: end for
14: $finalCost \leftarrow \infty$
15: for all $(j, v, l, s)$, $i = |C|$ do
16:     $finalCost \leftarrow \min(d[|C|][j][v][l][s], finalCost)$
17: end for
18: Return $finalCost$
```

Algorithm 2 Procedure $getNext$ (single-row)

```
1: Inputs: $s$
2: Initialize $nextList \leftarrow \emptyset$
3: $s \leftarrow$ shiftLeft1Bit($s$)
4: if $s[-r] = 0$ then
5:     $s[-r] \leftarrow 1$
6:     $nextStatus \leftarrow s$
7:     $nextList \leftarrow nextList \cup \{(-r, nextStatus)\}$
8:     Return $nextList$
9: end if
10: for all $m \in [-r, r]$ do
11:     if $s[m] = 0$ then
12:         $nextStatus \leftarrow s$
13:         $nextStatus[m] \leftarrow 1$
14:         $nextList \leftarrow nextList \cup \{(m, nextStatus)\}$
15:     end if
16: end for
17: Return $nextList$
```

IV. Double-Row Optimization

In this section, we briefly revisit the *double-row detailed placement* optimization presented in [7]. We give the problem statement addressed in [7], and explain limitations in double-row optimization compared to the multi-row optimization that we describe below in Section V.

Double-Row Optimization Problem. *Given an initial legalized double-row placement with double-height cells, perturb the placement within each row to minimize inter-cell diffusion* steps.

Inputs: Legalized double-row placement, available cell variants, and cost function of a diffusion *step*.

Output: Optimized double-row detailed placement with minimized overall cost (including inter-cell diffusion *steps*).

Constraints: Maximum displacement range, maximum reordering range, availability of cell flipping.

We note the following two limitations with regard to this problem statement.

Limitation 1. *Single-height cells cannot move across rows.*

In double-row optimization, the double-height cell effectively breaks the two rows into separate optimization regions, wherein we invoke single-row optimization separately for each of the two rows. Thus, in the double-row optimization of [7], single-height cells cannot be relocated to the other row.

Limitation 2. *The relative positions among double-height cells are fixed.*

For two double-height cells A and B, if A is initially to the left of B ($x_A < x_B$), then we require that in our final placement, c_A remains to the left of c_B. We note that we still allow reordering between a single-height cell and a double-height cell (thus, the double-height cells are *partially reorderable*) so as to maximize the *steps* reduction.

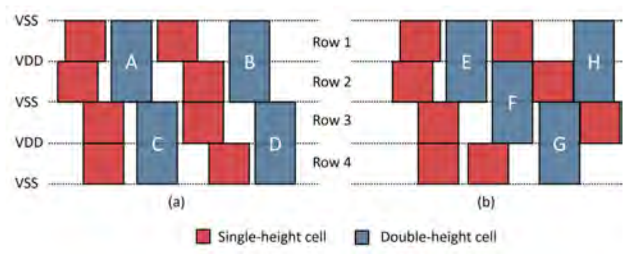

Fig. 5. Illustrations of double-height cells in placement rows. (a) Separable pairs of cell rows, reflecting power rail design of double-height cells in current N10 libraries. (b) Non-separable pairs of cell rows.

In the next section, we describe a more generic multi-row optimization without the above limitations. Due to the above limitations compared to the multi-row optimization described in Section V, as well as the page limit, we omit further details. The complete double-row formulation can be found in [7].

V. Multi-Row Optimization

In this section, we generalize from the single-row dynamic programming, and describe our approach for *multi-row detailed placement*, with support of fully-reorderable multi-height cells and inter-row cell relocating.

Multi-Row Optimization Problem. *Given an initial legalized multi-row placement, perturb the placement across the multiple rows to minimize inter-cell diffusion* steps.

Inputs: Legalized multi-row placement, available cell variants, and yield cost function.

Output: Optimized multi-row detailed placement with minimized overall cost (including inter-cell diffusion *steps*)

Constraints: Maximum horizontal displacement range, maximum vertical displacement range, maximum reordering range and availability of cell flipping.

A. Preliminaries

Similar to double-row optimization, we optimize m consecutive rows together (as a single optimization *window*) in multi-row optimization. In an optimization window, we move the cells according to our algorithm assuming that cells outside the window are fixed. Different windows are optimized separately. However, compared to the double-row optimization in Section IV, we do not require the relative positions among double-height cells to be fixed. Instead, a double-height cell can be reordered with another double-height cell as long as they are within the reordering range. Moreover, in contrast to Section IV's double-row optimization, where a cell cannot move outside its original cell row, here we allow a cell to move freely within a given vertical displacement range (in units of placement rows), enabling larger solution space to minimize diffusion *steps*.

In single-row and double-row optimization, where only intra-row relocating and reordering are allowed, the initial cell ordering (c_k in Table II) is defined within each row from the initial (input) placement. To enable a unified multi-row reordering range, with support of inter-row relocating and reordering, we redefine the original cell ordering as follows:

Definition. *Given an m-row initial (input) placement, cells in all m rows are left-to-right ordered according to their rightmost boundary, in a unified one-dimensional array, e.g., c_1, c_2, ..., c_k. If cells in the initial placement have the same x-coordinate for their right boundary, we break ties using y-coordinate of their lower boundary.*

Figure 6 shows an example of sequential cell ordering for a two-row initial placement. We note that cells c_4 and c_5 could have their positions exchanged in the ordering, regardless of their left boundary. However, as mentioned, in our implementation tie-breaking is by descending order of y-coordinate.

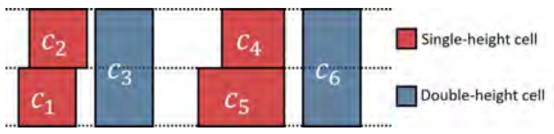

Fig. 6. An example of multi-row cell ordering. Cells are sequentially ordered (c_1 to c_6) according to the x-coordinate of their right boundary. Cells c_4 and c_5 have the same right boundary x-coordinate, and thus could be switched in the ordering.

With the above redefined cell ordering, reordering range works the same way as in Section III. The new sequentially ordered position is determined by the new x-coordinate (in the final solution) of the right boundary of each cell. The difference between the original and the new sequentially ordered position should be always within the reordering range. In the multi-row optimization, given the above redefined cell ordering, our dynamic programming still seeks to place one cell at a time, from left to right. The left-to-right placement procedure then induces the following assumption:

Assumption. *The x-coordinate of the right boundary of the $(i+1)^{st}$ cell must be greater than or equal to the right boundary of the partial placement consisting of i cells (i.e., placement boundary).*

Given the definition, the assumption does not reduce the solution space. For example, in Figure 7, assuming a partial placement of c_2 and c_1, if the 3^{rd} cell to be placed is c_3, and we would like its right boundary to be to the left of the placement boundary, then we can always get to such a partial placement solution from a partial placement of c_2 and c_3, followed by placement of c_1.

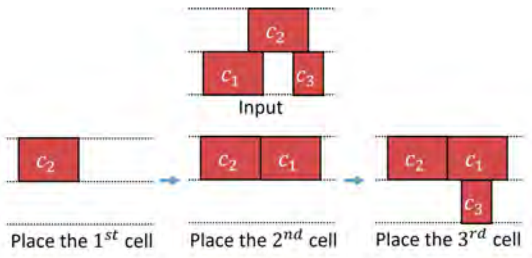

Fig. 7. Illustration of the **Assumption**.

B. Formulation

Given the above assumption, our approach will find an optimal placement solution for a given optimization window of m rows containing multi-height cells. We illustrate the multi-row dynamic programming-based detailed placement in Figure 8(a). We use type array $T = \{t_0, ..., t_{m-1}\}$ to describe the type, i.e., 2-fin, 3-fin or 4-fin configuration, of the rightmost cell in each row. Initially, each entry of T is an initial virtual cell, indicating that the placement boundary for all rows is the left boundary of the die, and that there will be no diffusion *step* penalty applied to any type of cell immediately to the right of this boundary. We also use distance array $D = \{d_0, ..., d_{m-1}\}$ to describe the shape of the placeable region as shown in Figure 8(b). The subproblems solved in the DP are of form: place $|C| - i$ cells in the placeable region defined by a partial placement with i cells.

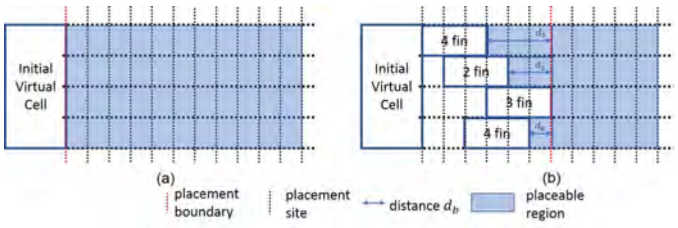

Fig. 8. Illustration of DP in multi-row placement with $m = 4$.

We give a precise description of our multi-row dynamic programming in Algorithm 3. Note that the numbers of entries of distance array D and cell type array T are both $m-1$ because the distance from the last placed cell to the placement boundary is always zero, and the cell type of the last placed cell can be retrieved by cell variant v. Lines 1-3 initialize the DP solution array. Lines 4-15 describe the main algorithm. Compared to single-row dynamic programming, we have one more iteration over all placement rows in an optimization window, subject to the maximum vertical displacement range constraint. Effectively, the multi-row DP array is different from single-row DP array in that it is capable of storing multiple intermediate placement solutions given the same cell ordering and horizontal displacement, as long as these solutions have different type (T) or distance (D) arrays. Also, Line 11 updates distance array D and cell type array T according to the choice of placement row b'. Lines 16-19 obtain the optimal solution among all legal cases when $i = |C|$, and Line 20 returns the optimal solution for the current optimization window.

Multi-row optimization is not capable of being aware of HPWL change in y direction and across different optimization windows. Therefore, to prevent HPWL degradation, we add additional displacement costs if a cell is moved out of the original HPWL bounding box, with penalty coefficient $\gamma_{penalty}$, as shown in Equation (4).[5] The term $cost_{hpwl}$ is calculated as the distance between the current cell and the original HPWL bounding box, in units of placement sites.

$$\begin{aligned} cost = cost_{step} + \alpha \cdot cost_{disp} + \gamma_{penalty} \cdot cost_{hpwl} \\ + \alpha \cdot \beta \cdot cost_{var} \end{aligned} \tag{4}$$

[5]We pre-calculate all net bounding boxes (one-time effort) and only apply the HPWL penalty if a cell is placed outside of its nets' bounding boxes.

Algorithm 3 Dynamic programming (multi-row)

```
1: **Initialize costs for all legal cases** $(j, v, l, b, D, T, s)$
2:     $d[0][j][v][l][b][D][T][s] \leftarrow 0,$
3:     $d[i][j][v][l][b][D][T][s] \leftarrow +\infty, (0 < i \leq |C|)$
4: **for all** $i = 0$ to $|C| - 1$ **do**
5:     **for all** $d[i][j][v][l][b][D][T][s] \neq \infty$ **do**
6:         **for all** $(j', s') \in$ getNext$(s)$ **do**
7:             **for all** $(v', l', b')$ **do**
8:                 $i' = i + 1$
9:                 $t \leftarrow d[i][j][v][l][b][D][T][s] + cost(^{j',v',l',b'}_{i,j,v,l,b,D,T})$
10:                $d[i'][j'][v'][l'][b'][D'][T'][s'] \leftarrow$
                     $\min(d[i'][j'][v'][l'][b'][D'][T'][s'], t)$
11:                $Update(D, T)$
12:            **end for**
13:        **end for**
14:    **end for**
15: **end for**
16: $finalCost \leftarrow \infty$
17: **for all** $(j, v, l, b, D, T, s)$ when $i = |C|$ **do**
18:     $finalCost \leftarrow \min(d[|C|][j][v][l][b][D][T][s], finalCost)$
19: **end for**
20: **Return** $finalCost$
```

VI. Experiments

We implement our dynamic programming in C++ with OpenAccess 2.2.43 [31] to support LEF/DEF [30], and with OpenMP [33] to enable thread-level parallelism. We perform experiments in an N7 FinFET technology with multi-height triple-Vt libraries from a leading technology consortium. The fin height information is not disclosed in our enablement. Therefore, following guidance from [28], we randomly assign fin heights (2, 3, or 4 fins) to each cell with 1:3:6 ratio for 2, 3 and 4 fins, respectively, as our default fin height assignment methodology to match industrial designs at advanced nodes. For example, a double-height cell will have four random fin heights, i.e., for its left and right boundaries on the first row, and its left and right boundaries on the second row. Section VI-C further discusses the impact of alternative fin height assignment methods.

We generate the bimodal leakage values from the NDE-oblivious standard-cell Liberty file as follows [28]. Since NDE only affects the boundary transistors for each cell, given a leakage value of each standard cell from the Liberty file, we first approximate the boundary transistor leakage value by dividing the state-independent cell leakage by the cell width (in units of contacted-poly pitch), e.g., if a cell (width = 3) has a leakage value of three, then the boundary transistors have a leakage value of one. Then, for each diffusion *step*, 52% of boundary transistor leakage value is added to the cell leakage. In the above example, the cell has a new leakage value of 3.52 (resp. 4.04) when there exists one *step* (resp. two *steps*).

We apply our detailed placement optimization to Arm Cortex-M0 and four design blocks (AES, JPEG, VGA and MPEG) from OpenCores [32]. Design information is summarized in Table III. We synthesize designs using *Synopsys Design Compiler L-2016.03-SP4* [34], and perform place-and-route using *Cadence Innovus Implementation System v15.2* [29]. We also apply our detailed placement optimization to winning solutions from the ICCAD-2017 multi-deck standard cell legalization contest [2]. All experiments are performed with 8 threads on a 2.6GHz Intel Xeon server.

In the following, we show (i) the scalability and sensitivity, i.e., impact of cell displacement range x_Δ, reordering range r, enabling of cell flipping f, and #rows per window m for the multi-row implementation on runtime and quality of results (QoR in terms of *step* reduction); (ii) impact of the weighting factors, i.e., weighting factor α for cell displacement, weighting factor β for cell flipping, and weighting factor γ for HPWL on QoR; (iii) metaheuristics by combining single-row HPWL-aware and multi-row optimization; (iv) our main results with single-row, double-row and multi-row optimization for five design blocks and three fin height assignment methodologies; (v) performance improvement using *intentional steps*; and (vi) our results with multi-row optimization for ICCAD-2017 benchmark [2].

TABLE III
Design information.

design	#inst	clkp
AES	∼12K	500ps
M0	∼10K	500ps
JPEG	∼54K	500ps
VGA	∼69K	500ps
MPEG	∼14K	500ps

A. Scalability/Sensitivity Study

In this subsection, we compare the impact of reordering range and displacement range on the single-row (SR), double-row (DR) and multi-row (MR) optimization. By default, we use $m = 2$ in MR optimization (see Figure 10 and discussion below). Following results of [7], cell flipping is enabled by default for maximum *step* reduction.

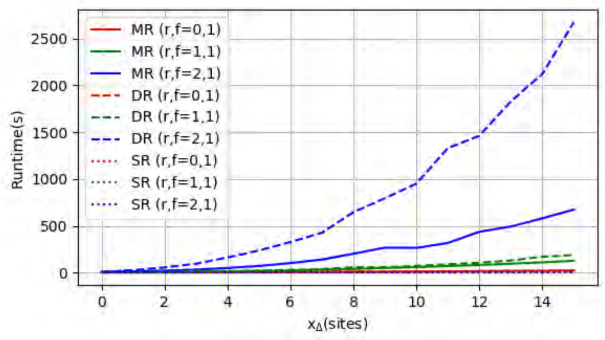

Fig. 9. Sensitivity of runtime to (x_Δ, r, f) parameters.

To assess the scalability of our approach, we sweep (x_Δ, r), i.e., maximum allowed cell displacement x_Δ (in placement sites) and maximum allowed one-sided reordering r, and study the impact on runtime. In this experiment, we sweep x_Δ from 0 to 15, and r from 0 to 2. A cell can freely move across 31 placement sites, and can have up to 5 different positions in a placement window, if we set $x_\Delta = 15$ and $r = 2$. We set $(\alpha, \beta) = (0, 0)$ as these parameters do not have any impact on the complexity of our formulation. We use design block AES for this study.[6]

Our study results are shown in Figure 9. We find that the runtime generally grows quadratically with the number of available placement sites per each cell. However, for cell

[6]To investigate the stability of our sensitivity studies and observations, we also use (i) an alternative AES design implementation with slightly different layout, and (ii) design block CORTEXM0DS. Results for (i) and (ii) are consistent with the results that we report here.

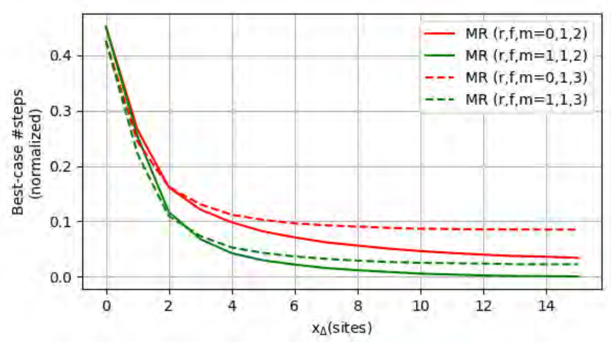

Fig. 10. Sensitivity of #*steps* to m in MR optimization.

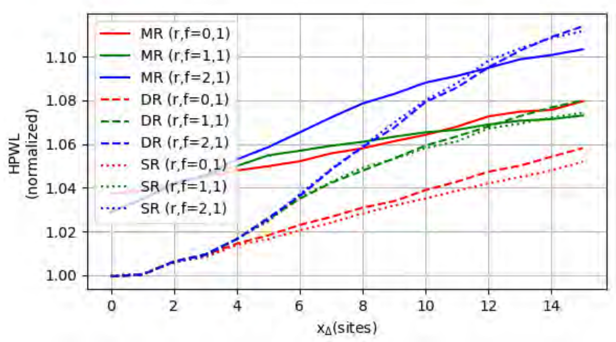

Fig. 12. Sensitivity of HPWL to (x_Δ, r, f) parameters.

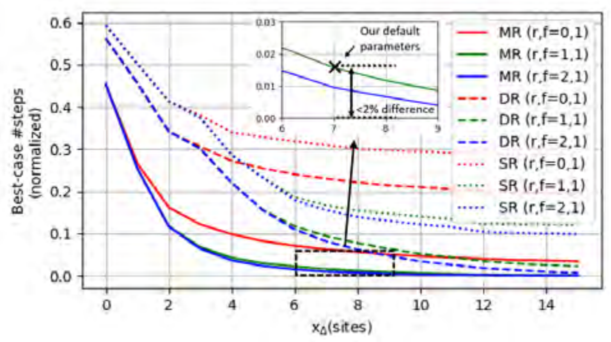

Fig. 11. Sensitivity of #*steps* to (x_Δ, r, f) parameters.

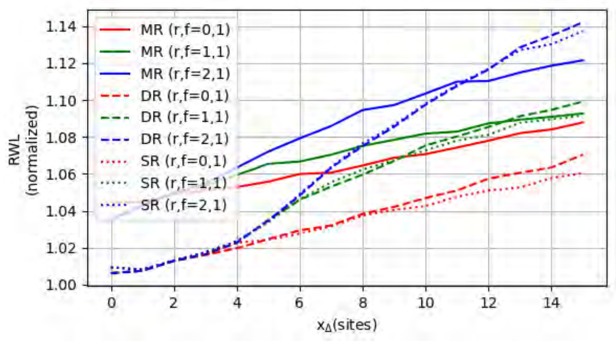

Fig. 13. Sensitivity of RWL to (x_Δ, r, f) parameters.

reordering, there is a dramatic increase in runtime as r goes up, e.g., we observe 12× runtime increase going from $r = 1$ to $r = 2$.

Also, compared to DR [7], our new MR implementation with $m = 2$ rows per window is much more efficient in terms of runtime. To investigate the impact of m (#rows in a window) in MR, we compare the sensitivity of #*steps* in Figure 10 for $m = 2$ and $m = 3$. Runs with $m = 4$ are not feasible due to much larger memory consumption. We find that $m = 2$ actually gives better #*steps* than $m = 3$ using our N7 library, because all multi-height cells have VSS power rails for their cell boundaries, such that all multi-height cells are aligned per two cell rows. Given the above observation, we use $m = 2$ for MR in all of the following experiments.

To assess the sensitivity to (x_Δ, r), Figures 11, 12 and 13 show #diffusion *steps*, HPWL and RWL respectively, as we sweep (x_Δ, r). Since our algorithm only optimizes #diffusion *steps* when $(\alpha, \beta) = (0, 0)$, here we see HPWL and RWL that correspond to a best-case (minimized) #*steps* normalized to initial design.

We see from Figure 11 that SR can only reduce #*steps* by up to 80%, while DR and MR are able to reduce #*steps* by up to 99% given larger displacement range. Also, MR is consistently better than DR, especially given a smaller displacement range. Along with the runtime benefit of MR, we believe that the new MR implementation surpasses both the solution quality and the runtime efficiency of DR [7].

Moreover, for $f = 1$, there is only ∼0.6% benefit of using $r = 2$ over $r = 1$, at the cost of 12× the runtime; this suggests that $r \geq 2$ may not offer significant benefit in reducing #*steps*. In Figure 12 and Figure 13, HPWL and RWL increase linearly as x_Δ goes up. Based on these studies, to balance solution quality and runtime we apply $(x_\Delta, r) = (7, 1)$ in all of the following experiments.

B. Study of Weighting Factors

In the following subsection, our default flow is MR optimization, with two rows per window. We investigate impacts of the weighting factors $(\alpha, \gamma_{penalty})$ for cell displacement and HPWL penalty $(\gamma_{penalty})$ on HPWL and #*steps*. We sweep α and $\gamma_{penalty}$ from 0 to 1. We perform this experiment using design block AES. The results are shown in Figure 14. We can see that a non-zero displacement weight (α) and a non-zero HPWL penalty $(\gamma_{penalty})$ save HPWL while preserving most of the *step* reduction benefits. Therefore, we apply $\alpha = 0.01$ and $\gamma_{penalty} = 0.00001$ in all following experiments.

For the single-row optimization, we also study the impact of the HPWL weighting factor γ on HPWL and #*steps*. We sweep γ from 0.00001 to 1 with a step size of 10×. We perform this experiment using design block AES, with results shown in Figure 15. The tradeoff between HPWL and #*step* is clear when γ is in the range of $[0.00001, 0.01]$. We use $\gamma = 0.0001$ for the HPWL-aware single-row optimization.

C. Main Results

We apply our multi-row dynamic programming-based optimization to all our design blocks using the aforementioned parameter settings, i.e., $(x_\Delta, r, f) = (7, 1, 1)$ and $(\alpha, \beta) = (0.01, 1)$. Table IV shows the *step* reduction, runtime and estimated yield improvement for all five design blocks using multi-row optimization. We also report the impact on other metrics, i.e., routed wirelength (RWL), worst negative slack (WNS) and leakage power as reported by the place-and-route tool [29].

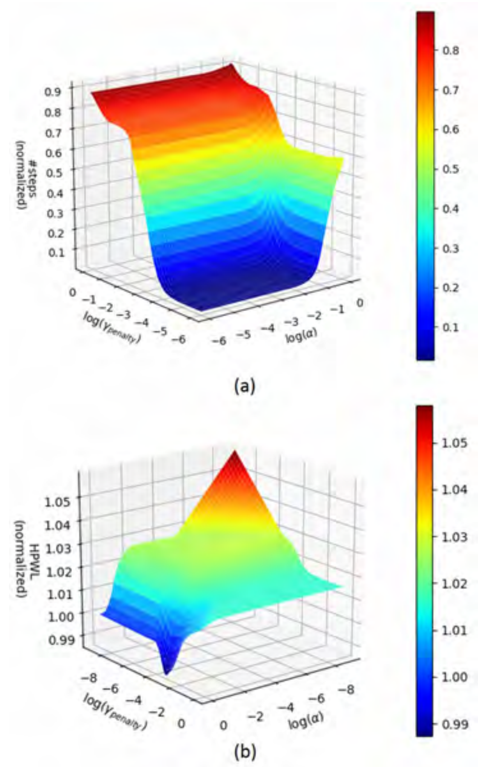

Fig. 14. Impacts of weighting factors $(\alpha, \gamma_{penalty})$ on the tradeoff between HPWL and #*steps*.

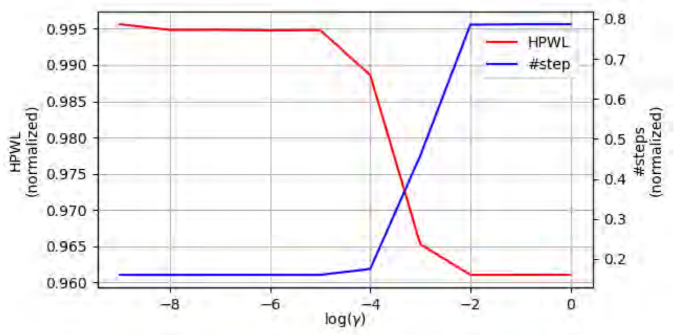

Fig. 15. Impact of weighting factor γ on the tradeoff between HPWL and #*steps*.

We also investigate the impact of fin height assignment methodologies. We apply three methodologies – (i) *rand* randomly assigns fin heights according to probability ratio 1:3:6 for 2, 3, and 4 fins, respectively (See Section VI above); (ii) *Vt* assigns fin heights according to their Vt property, with HVT (resp. NVT and LVT) cells having probability ratio 1:1:0 (resp. 1:1:1 and 0:1:1) for 2, 3, and 4 fins; (iii) *drive* assigns fin height according to their drive strength, with X0 (resp. X1 and others) cells having probability ratio 1:1:0 (resp. 1:1:1 and 0:1:1) for 2, 3, and 4 fins. The three methodologies generate different fin height distributions, and thus help confirm the robustness of our optimization in broader scenarios. The results are shown in Table IV. For all designs with the default (*rand*) random fin height distribution, we achieve up to 98.1% reduction in #*steps* at the cost of around 3.5% RWL increase. The results also show that our optimization has negligible impact on WNS and that we can slightly improve the leakage. In addition, we perform a preliminary yield estimation assuming 2ppm failure rate for each *step*, and 1ppm failure rate after we remove the *step* (recall Footnote 3). Based on this assumption, we can see a yield improvement of up to 4.56% for a design block of 69K instances. We note that the yield improvement is expected to grow markedly with the die size. A larger design of millions of instances may see more benefits.

For Vt and drive distribution, the results show similar *step* reduction percentage, demonstrating the robustness of our optimization. Figure 16 shows the layouts of placements before and after MR optimization.

We also investigate the improvement achieved by our multi-row optimization over single-row, double-row optimization and previous works. We compare multi-row (MR) optimization to (i) single-row (SR) optimization (also to match [5][16]), (ii) ordered double-row (ODR) optimization (to match [15]), and (iii) double-row (DR) optimization. For (i), we use the proposed methodology in Section III and fix the locations of all multi-height cells. We note that our SR implementation is equivalent to [5][16], supporting neighboring cell swapping and cell flipping with the adaptation of NDE. In SR, we use the same displacement range and reordering range as in DR, while using the default HPWL weighting factor $\gamma = 0.0001$ (HPWL weighting factor is not considered in the work of [7]). For (ii), we simply run our DR optimization with zero reordering range to achieve an ODR equivalent to [15]. For (iii), we use the proposed methodology in Section IV. The comparisons of #*steps*, routed wirelength (RWL) and runtime are shown in Tables V, VI and VII, respectively. For design blocks with fewer double-height cells, SR performance is competitive with that of ODR. However, for design blocks with more double-height cells, ODR is significantly better (up to 21% more *step* reduction) than SR due to movable double-height cells. The results show that DR effectively reduces the diffusion *steps* by around half compared to SR, and by around 40% compared to ODR. On average, DR has 11.6% more *step* reduction than ODR, and 17.7% more than SR, with respect to the initial number of diffusion *steps*. This suggests the importance of supporting movable and reorderable double-height cells, as there will be substantial benefits.

D. Metaheuristics

We have also explored several metaheuristics to assess (i) the *step* reduction achievable by invoking multiple optimization iterations, as well as (ii) potential improved tradeoffs between *step* reduction and degradation from initial placement (in terms of HPWL). First, we investigate the maximum *step* reduction versus the number of iterations. To explore the maximum benefits of *step* reduction, we invoke the multi-row optimization several times. Since the multi-row optimization is for every two rows, e.g., row 1 and 2 in a window, row 3 and 4 in the next window, we can shift the window by one row and run again if we can further improve the solution quality. In our experiments, we alternatively align / unalign the optimization window with double-height cells, with aligned window in the first iteration to encourage the movement of double-height cells. We show the normalized number of *diffusion steps* and

TABLE IV
EXPERIMENTAL RESULTS FOR ALL DESIGN BLOCKS USING MULTI-ROW OPTIMIZATION.

Design	**Type**	**Fin Height Distribution**			***#steps***		**RWL** (μm)		**WNS** (ns)		**Leakage** (mW)		**Runtime**	**Est. Yield**
		2 fin%	3 fin%	4 fin%	Init	Final (Δ%)	Init	Final (Δ%)	Init	Final	Init	Final (Δ%)	(sec)	**Impr.** %
	rand	10.0	30.4	59.6	7973	152 (-98.1%)	31873	32995 (+3.5%)	-0.013	-0.021	16.1	15.8 (-2.1%)	162.1	+0.71
AES	Vt	48.3	47.8	3.9	6816	143 (-97.9%)	31874	32944 (+3.4%)	-0.013	-0.020	16.6	15.8 (-4.9%)	81.5	+0.66
	drive	47.5	46.6	5.9	7215	236 (-96.7%)	31874	32888 (+3.2%)	-0.013	-0.018	16.1	15.8 (-2.0%)	109.9	+0.69
	rand	10.1	30.4	59.4	6588	243 (-96.3%)	27670	28728 (+3.8%)	-0.043	-0.070	18.9	18.6 (-1.9%)	174.4	+0.22
M0	Vt	49.3	48.6	2.1	5379	152 (-97.2%)	27674	28588 (+3.3%)	-0.043	-0.111	19.5	18.6 (-4.5%)	74.1	+0.52
	drive	46.3	45.6	8.0	6211	398 (-93.6%)	27669	28718 (+3.8%)	-0.043	-0.051	19.1	18.6 (-2.6%)	64.5	+0.58
	rand	10.0	30.0	60.0	34760	656 (-98.1%)	101000	107699 (+6.6%)	-0.319	-0.278	96.3	94.3 (-2.1%)	776.5	+3.50
JPEG	Vt	48.2	48.6	3.3	29452	387 (-98.7%)	100997	106972 (+5.9%)	-0.319	-0.274	98.8	94.4 (-4.4%)	403.2	+2.78
	drive	44.0	44.5	11.5	36173	1291 (-96.4%)	101003	108103 (+7.0%)	-0.323	-0.290	97.2	94.4 (-2.9%)	398.2	+3.30
	rand	10.0	30.1	60.0	50766	6179 (-87.8%)	208155	217492 (+4.5%)	-0.137	-0.080	208.3	205.1 (-1.5%)	713.3	+4.56
VGA	Vt	48.8	49.6	1.6	40743	3685 (-91.0%)	208155	216603 (+4.1%)	-0.137	-0.069	213.4	205.5 (-3.7%)	536.8	+3.48
	drive	42.1	42.8	15.1	57273	10871 (-81.0%)	208155	217664 (+4.6%)	-0.137	-0.129	208.2	205.1 (-1.5%)	491.1	+4.24
	rand	9.9	30.5	59.6	9994	1367 (-86.3%)	38896	40594 (+4.4%)	-0.005	-0.018	33.2	33.1 (-0.2%)	137.3	+0.87
MPEG	Vt	49.6	49.4	1.0	7824	753 (-90.4%)	38882	40383 (+3.9%)	-0.011	-0.026	33.2	33.1 (-0.3%)	68.6	+0.70
	drive	43.1	43.1	13.8	10931	2145 (-80.4%)	38901	40649 (+4.5%)	-0.005	-0.030	33.2	33.1 (-0.3%)	99.5	+0.86

TABLE V
COMPARISON OF DIFFUSION *steps* WITH SR (TO MATCH [5][16]), ODR (TO MATCH [15]) DR, MR AND METAHEURISTICS (META). DH%:= % OF DOUBLE-HEIGHT CELLS.

Design	DH%	Init	**SR (to match [5][16])**	**ODR (to match [15])**	**DR**	**MR**	**Meta**
AES	4.3%	7973	1395 (-82.5%)	1869 (-76.6%)	750 (-90.6%)	152 (-98.1%)	131 (-98.4%)
M0	8.4%	6588	1672 (-74.6%)	1742 (-73.6%)	842 (-87.2%)	243 (-96.3%)	179 (-97.3%)
JPEG	8.3%	34760	9731 (-72.0%)	8341 (-76.0%)	4555 (-86.9%)	656 (-98.1%)	473 (-98.6%)
VGA	24.8%	50766	27170 (-46.5%)	16405 (-67.7%)	11816 (-76.7%)	6179 (-87.8%)	5652 (-88.9%)
MPEG	23.0%	9994	5101 (-49.0%)	3444 (-65.5%)	2402 (-76.0%)	1367 (-86.3%)	1215 (-87.8%)
Avg.	–	-0.00%	-64.9%	-71.9%	-83.5%	-93.3%	-94.2%

TABLE VI
COMPARISON OF ROUTED WIRELENGTH (RWL) WITH SR, ODR, DR, MR AND METAHEURISTICS (META).

Design	Init	**SR**	**ODR**	**DR**	**MR**	**Meta**
AES	31873	32517 (+2.02%)	32637 (+2.40%)	32898 (+3.22%)	32995 (+3.52%)	33065 (+3.74%)
M0	27670	28201 (+1.92%)	28271 (+2.17%)	28470 (+2.89%)	28728 (+3.82%)	28805 (+4.10%)
JPEG	101000	104562 (+3.53%)	104657 (+3.62%)	105550 (+4.50%)	107699 (+6.63%)	108173 (+7.10%)
VGA	208155	212186 (+1.94%)	212905 (+2.28%)	214169 (+2.89%)	217492 (+4.49%)	216856 (+4.18%)
MPEG	38896	39640 (+1.91%)	39799 (+2.32%)	39950 (+2.71%)	40594 (+4.37%)	40512 (+4.15%)
Avg.	+0.00%	+2.26%	+2.56%	+3.24%	+4.57%	+4.66%

TABLE VII
COMPARISON OF RUNTIME (SECONDS) WITH SR, ODR, DR, MR AND METAHEURISTICS (META).

Design	**SR**	**ODR**	**DR**	**MR**	**Meta**
AES	32	8	59	162	348
M0	22	8	51	174	214
JPEG	325	50	344	776	2153
VGA	493	51	386	713	1658
MPEG	30	11	86	137	234

HPWL versus the number of optimization iterations (up to 8) in Figure 17. Compared to one iteration, the second iteration removes 45 out of 152 remaining steps after the first iteration, while the remaining six iterations only reduce 13 more *steps*, at the cost of increased HPWL.

Given the above observation, we seek to obtain a better tradeoff between *step* reduction and HPWL. Since our multi-row optimization is not HPWL-aware, we propose to invoke both single-row and multi-row optimization with a total "budget" of four iterations, to find the best four-iteration sequence. We explore all possible optimization sequences comprised of the following three configurations – (A) single-row HPWL-aware; (B) multi-row aligned with double-height cells; and (C) multi-row unaligned with double-height cells. We report the optimized number of *steps*, along with HPWL, in Figure 18. We can see that the configuration for the first iteration largely determines the optimized number of *steps*. The first iteration should be (B) to obtain better *step* reduction. Also, the optimization should finish with (A) for better HPWL. We report the metaheuristic results in Tables V, VI and VII.

E. Performance Improvement Using Intentional Steps

Similar in spirit to [12], we explore the possibility of improving design performance with *intentional steps* – i.e., using filler cells that create an *intentional step* to the neighboring timing-critical functional cell so as to improve the timing

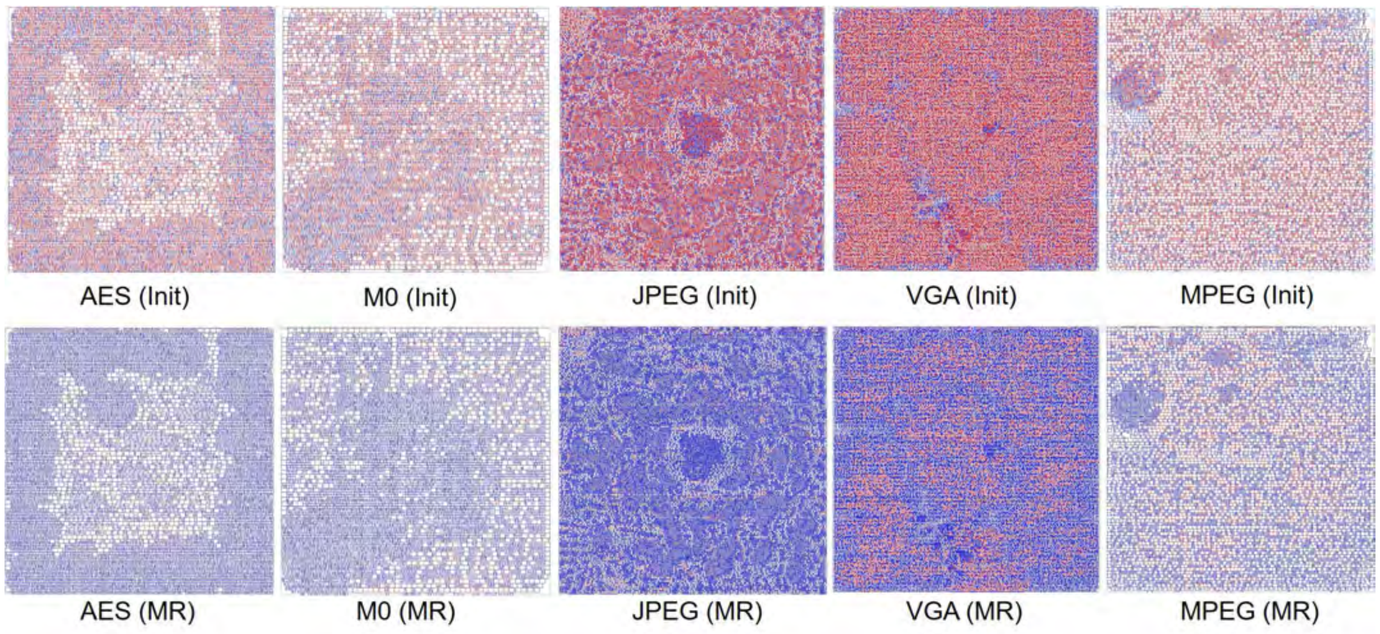

Fig. 16. Layouts of placements before (Init) and after (MR) our MR optimization. Red color indicates cell instances with diffusion *steps* and blue color indicates cell instances without diffusion *steps*.

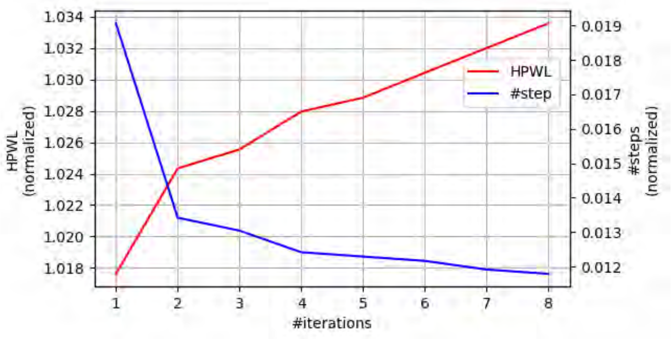

Fig. 17. #*steps* (normalized) and HPWL (normalized) vs. #iterations in metaheuristic optimization.

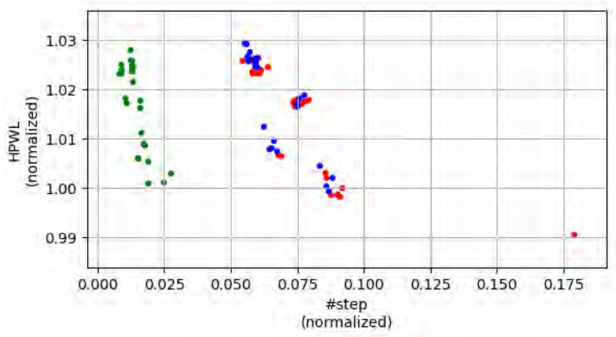

Fig. 18. #steps vs. HPWL in metaheuristic optimization. Red (resp. green and blue) dots represent metaheuristic iterations that start with configuration A (resp. configuration B and configuration C).

of that functional cell.[7] In the cost function, we use a third weighting factor δ to represent the benefit of an *intentional step* to a timing-critical cell. We sweep δ from 0 to -2 with a step size of -0.1. We select 5% of all cells as timing-critical cells and perform optimization using all design blocks. The results are shown in Figure 19. We use $orig.opt$ to represent the results with $\delta = 0$, and $time.opt$ to represent the results with $\delta = -0.3$. Compared to $\delta = 0$, we achieve up to $5\times$ increase in #*filler-induced steps* incident to timing-critical cells when $\delta = -0.3$, at the cost of slightly increased #*non-filler-induced steps* to non-timing-critical cells. This translates to up to 2.13 *steps* per timing-critical cell after $time.opt$, compared to 0.42 *steps* after $orig.opt$. Overall, we can still decrease total *steps* by more than 70%, showing the effectiveness of our algorithm. We note that as we add more *intentional steps* to timing-critical cells, we leave a smaller solution space for non-timing-critical cells. Thus, $time.opt$ generates more *steps* to non-timing-critical cells. We furthermore observe that as δ decreases, the #*intentional steps* that we can achieve approaches a limit, as shown in Figure 20. This may help set expectations for benefits that might be derived from a more comprehensive, timing-aware flow (which we leave for future work).

[7]An *intentional inter-cell step* may increase/decrease the drive strength of the function cell. E.g., a *step* adjacent to a PFET may decrease the drive strength while a *step* adjacent to an NFET may increase the drive strength. Here, instead of using a filler cell to match diffusion heights for both the NFET and the PFET of the function cell (to reduce #*steps*), we create a filler-induced *intentional step* by matching the diffusion height for only the PFET, thus increasing the drive strength for the NFET. We note that exact timing and power impacts and tradeoffs will vary with STI processes.

F. ICCAD-2017 Benchmark Results

We apply our multi-row dynamic programming-based optimization to winning solutions from the ICCAD-2017 contest [2] only considering row and site alignments, but not

TABLE VIII
DESIGN INFORMATION AND EXPERIMENT RESULTS FOR ICCAD-2017 BENCHMARK [2]. DISTRIBUTION OF SINGLE-HEIGHT, DOUBLE-HEIGHT, TRIPLE-HEIGHT AND QUADRUPLE-HEIGHT CELLS ARE SHOWN IN COLUMNS 1×H, 2×H, 3×H AND 4×H, RESPECTIVELY.

design	**#inst**	**cell types %**				***#steps***		**Runtime**
		1×H	2×H	3×H	4×H	Init	Final (Δ%)	(sec)
des_perf_b_md1	~11K	94.80	5.20	0.00	0.00	57806	3781 (-93.46%)	361.3
des_perf_b_md2	~11K	90.47	6.02	2.01	1.50	70733	7494 (-89.41%)	232.8
edit_dist_1_md1	~13K	90.31	6.12	2.04	1.53	74351	6019 (-91.90%)	420.9
edit_dist_a_md2	~13K	90.31	6.12	2.04	1.53	76657	8074 (-89.47%)	417.8
fft_2_md2	~ 3K	89.62	6.56	2.18	1.64	22040	3789 (-82.81%)	53.2
fft_a_md2	~ 3K	89.57	6.59	2.19	1.65	10960	606 (-94.47%)	136.4
fft_a_md3	~ 3K	93.42	2.19	2.19	2.19	11631	372 (-96.80%)	78.1
pci_bridge32_a_md1	~ 3K	90.39	6.07	2.02	1.52	17284	1429 (-91.73%)	83.8
des_perf_1	~11K	100.00	0.00	0.00	0.00	73202	3516 (-95.20%)	488.7
des_perf_a_md1	~11K	95.66	4.34	0.00	0.00	64624	3060 (-95.26%)	307.3
des_perf_a_md2	~11K	96.99	1.00	1.00	1.00	64346	4793 (-92.55%)	315.9
edit_dist_a_md3	~13K	93.88	2.04	2.04	2.04	78560	11100 (-85.87%)	258.9
pci_bridge32_a_md2	~ 3K	85.51	7.08	4.05	3.37	21435	6235 (-70.91%)	71.2
pci_bridge32_b_md1	~ 3K	90.39	6.07	2.02	1.52	14988	1070 (-92.86%)	68.1
pci_bridge32_b_md2	~ 3K	96.97	1.01	1.01	1.01	13812	488 (-96.47%)	135.0
pci_bridge32_b_md3	~ 3K	94.94	1.01	2.02	2.02	14929	1193 (-92.01%)	84.2

considering constraints, including maximum cell movement, cell edge spacing, pin access, pin shorts and fence regions from the contest. The input legalized placements for all benchmark testcases are from the first-place team's solutions in ICCAD-2017 contest, except pci_bridge32_a_md1 and pci_bridge32_a_md2, for which we use the second-place team's solutions (because the first-place team's solutions for these two testcases have cells placed outside of the die boundary). We keep the same P/G alignment as in the input placement. We apply *rand* fin height assignment methodology with the above-mentioned 1:3:6 ratio for 2, 3 and 4 fins, respectively. The results are shown in Table VIII. For all ICCAD-2017 benchmark testcases, we achieve up to 96.8% reduction in #*steps*.

VII. CONCLUSIONS

In this work, we have presented an *optimal* dynamic programming-based single-/double-row detailed placement methodology to minimize diffusion *steps* in sub-10nm VLSI, for improved yield and mitigation of NDE. Our work achieves several improvements as compared to previous works: (i) optimal dynamic programming with support of a richer set of cell movements, i.e., flipping, relocating and enhanced reordering; (ii) optimal double-row dynamic programming *with support of movable and reorderable double-height cells*; and (iii) a novel performance improvement technique using *intentional steps*. The proposed techniques achieve up to 98% reduction of inter-cell diffusion *steps*, with scalable runtime and high die utilization in an N7 node enablement. Open directions for future research include (i) a more comprehensive timing-aware optimization flow that is capable of addressing the NDE while minimizing placement disturbance; (ii) extension to other layout-dependent effect (LDE)-aware optimizations [25][27]; (iii) integration with other detailed placement objectives; and (iv) speedup techniques with regard to the reordering range.

ACKNOWLEDGMENTS

We thank Kwangsoo Han and Hyein Lee for their contribution in the work of [7].

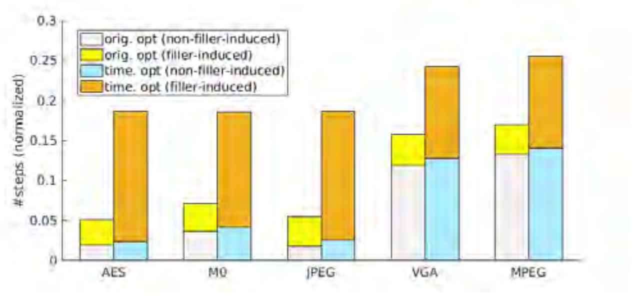

Fig. 19. Comparison of #filler-induced *steps* and total #*steps* for all design blocks before ($orig.opt$, $\delta = 0$) and after ($time.opt$, $\delta = -0.3$) using *intentional steps*.

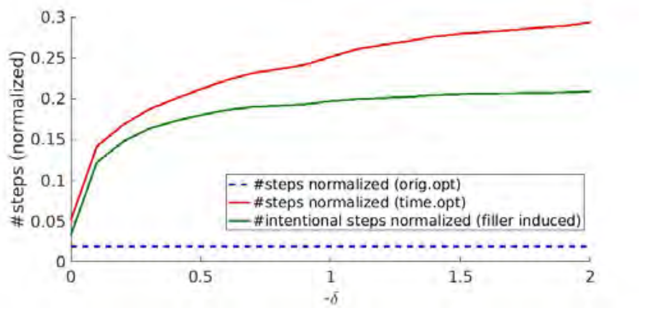

Fig. 20. Sensitivity of filler-induced *steps* to δ. Testcase: AES.

REFERENCES

[1] D. C. Chen, G. S. Lin, T. H. Lee, R. Lee, Y. C. Liu, M. F. Wang, Y. C. Cheng and D. Y. Wu, "Compact Modeling Solution of Layout Dependent Effect for FinFET Technology", *in Proc. ICMTS*, 2015, pp. 110-115.

[2] N. K. Darav, I. S. Bustany, A. Kennings and R. Mamidi, "ICCAD-2017 CAD Contest in Multi-Deck Standard Cell Legalization and Benchmarks", *in Proc. ICCAD*, 2017, pp. 867-871.

[3] P. Debacker, K. Han, A. B. Kahng, H. Lee, P. Raghavan and L. Wang, "Vertical M1 Routing-Aware Detailed Placement for Congestion and Wirelength Reduction in Sub-10nm Nodes", *in Proc. DAC*, 2017, pp. 51:1-51:6.

[4] S. Dobre, A. B. Kahng and J. Li, "Mixed Cell-Height Implementation for Improved Design Quality in Advanced Nodes", *in Proc. ICCAD*, 2015, pp. 854-860.

[5] Y. Du and M. D. F. Wong, "Optimization of Standard Cell Based

Detailed Placement for 16nm FinFET Process", *in Proc. DATE*, 2014, pp. 1-6.
[6] J. V. Faricelli, "Layout-Dependent Proximity Effects in Deep Nanoscale CMOS", *in Proc. CICC*, 2010, pp. 1-8.
[7] C. Han, K. Han, A. B. Kahng, H. Lee, L. Wang and B. Xu, "Optimal Multi-Row Detailed Placement for Yield and Model-Hardware Correlation Improvements in Sub-10nm VLSI", *in Proc. ICCAD*, 2017, pp. 667-674.
[8] K. Han, A. B. Kahng and H. Lee, "Scalable Detailed Placement Legalization for Complex Sub-14nm Constraints", *in Proc. ICCAD*, 2015, pp. 867-873.
[9] D. Hill, "Method and System for High Speed Detailed Placement of Cells Within an Integrated Circuit Design", *US Patent 6370673*, 2002.
[10] S.-W. Hur and J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement", *in Proc. ICCAD*, 2000, pp. 165-170.
[11] A. B. Kahng, I. L. Markov and S. Reda, "On Legalization of Row-Based Placements", *in Proc. GLSVLSI*, 2004, pp. 214-219.
[12] A. B. Kahng, P. Sharma and R. O. Topaloglu, "Exploiting STI Stress for Performance", *in Proc. ICCAD*, 2007, pp. 83-90.
[13] A. B. Kahng, P. Tucker and A. Zelikovsky, "Optimization of Linear Placements for Wirelength Minimization with Free Sites", *in Proc. ASP-DAC*, 1999, pp. 241-244.
[14] S. Li and C.-K. Koh, "Mixed Integer Programming Models for Detailed Placement", *in Proc. ISPD*, 2012, pp. 87-94.
[15] Y. Lin, B. Yu, X. Xu, J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, C. J. Alpert and D. Z. Pan, "MrDP: Multiple-row Detailed Placement of Heterogeneous-sized Cells for Advanced Nodes", *in Proc. ICCAD*, 2016, pp. 1-8.
[16] Y. Lin, B. Yu, B. Xu and D. Z. Pan, "Triple Patterning Aware Detailed Placement Toward Zero Cross-Row Middle-of-Line Conflict", *in Proc. ICCAD*, 2015, pp. 396-403.
[17] S.-K. Oh, "Standard Cell Library, Method of Using the Same, and Method of Designing Semiconductor Integrated Circuit", *US Patent App*, US20160055283.
[18] H.-C. Ou, K.-H. Tseng, J.-Y. Liu, I.-P. Wu and Y.-W. Chang, "Layout-Dependent-Effects-Aware Analytical Analog Placement", *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 8, pp. 1243-1254, Aug. 2016.
[19] M. Pan, N. Viswanathan and C. Chu, "An Efficient and Effective Detailed Placement Algorithm", *in Proc. ICCAD*, 2005, pp. 48-55.
[20] M. Tarabbia, A. Mittal and N. Hindawy, "Forming FinFET Cell with Fin Tip and Resulting Device", *US Patent App*, US20150137203.
[21] H. Tian, Y. Du, H. Zhang, Z. Xiao and M. D. F. Wong, "Triple Patterning Aware Detailed Placement with Constrained Pattern Assignment", *in Proc. ICCAD*, 2014, pp. 116-123.
[22] C.-H. Wang, Y.-Y. Wu, J. Chen, Y.-W. Chang, S.-Y. Kuo, W. Zhu and G. Fan, "An Effective Legalization Algorithm for Mixed-Cell-Height Standard Cells", *in Proc. ASP-DAC*, 2017, pp. 450-455.
[23] 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. 8, pp. 1569-1573, Sep. 2016.
[24] R. Xie, K.-Y. Lim, M. G. Sung and R. R.-H. Kim, "Methods of Forming Single and Double Diffusion Breaks on Integrated Circuit Products Comprised of FinFET Devices and The Resulting Products", *US Patent, US9412616*, 2016.
[25] S. Yang, Y. Liu, M. Cai, J. Bao, P. Feng, X. Chen, L. Ge, J. Yuan, J. Choi, P. Liu, Y. Suh, H. Wang, J. Deng, Y. Gao, J. Yang, X.-Y. Wang, D. Yang, J. Zhu, P. Penzes, SC Song, C. Park, S. Kim, J. Kim, S. Kang, E. Terzioglu, K. Rim and PR. C. Chidambaram, "10nm High Performance Mobile SoC Design and Technology Co- Developed for Performance, Power and Area Scaling", *in Proc. VLSI Technology*, pp. T70-T71.
[26] B. Yu, X. Xu, J.-R. Gao, Y. Lin, Z. Lee, C. J. Alpert and D. Z. Pan, "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.
[27] P. Zhao, S. M. Pandey, E. Banghart, X. He, R. Asra, V. Mahajan, H. Zhang, B. Zhu, K. Yamada, L. Cao, P. Balasubramaniam, M. Joshi, M. Eller, F. Benistant and S. Samavedam, "Influence of Stress Induced CT Local Layout Effect (LLE) on 14nm FinFET", *in Proc. VLSI Technology*, pp. T228-T229.
[28] Model-Hardware Correlation Team, *Samsung Electronics Co., Ltd.*, Nov. 2016.
[29] Cadence Innovus User Guide, http://www.cadence.com
[30] LEF/DEF reference 5.7. http://www.si2.org/openeda.si2.org/projects/lefdef
[31] Si2 OpenAccess. http://www.si2.org/?page=69
[32] OpenCores: Open Source IP-Cores, http://www.opencores.org
[33] OpenMP Architecture Review Board, "OpenMP Application Program Interface, Version 4.0".
[34] Synopsys Design Compiler User Guide, http://www.synopsys.com

Changho Han received his B.S. degree in electrical and electronic engineering from KAIST, Daejeon, South Korea, in 2001. He is a principal engineer leading model-hardware correlation team in Samsung Electronics.

Andrew B. Kahng is a professor at the Computer Science Engineering Department and Electrical and the Computer Engineering Department of the University of California at San Diego. His interests include IC physical design, the design-manufacturing interface, combinatorial optimization, and technology roadmapping. He received the Ph.D. degree in Computer Science from the University of California at San Diego.

Lutong Wang received the B.S. degree in microelectronics from Tsinghua University, Beijing, China, in 2014 and the M.S. degree in electrical and computer engineering from the University of California at San Diego, La Jolla, in 2016. He is currently pursuing the Ph.D. degree at the University of California at San Diego, La Jolla. His research interests include physical design implementation and DFM methodologies.

Bangqi Xu received the B.S. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA in 2015 and the M.S. degree in electrical and computer engineering from the University of California at San Diego, La Jolla, in 2017. He is currently pursuing the Ph.D. degree at the University of California at San Diego, La Jolla. His current research interests include detailed placement, PDN optimization and machine learning.