Improvement of Mixed Track-Height Standard-Cell Placement

Andrew B. Kahng[†], Seokhyeong Kang[‡*] and Minhyuk Kweon[‡]
[†]Department of Electrical and Computer Engineering, University of California San Diego
[‡]Department of Electrical Engineering, Pohang University of Science and Technology
[*]shkang@postech.ac.kr

***Abstract*—In sub-5nm nodes, track-height of standard cells must be aggressively scaled down while preserving design PPA. This requirement brings the challenge of placing a set of cells that have mixed track-heights, subject to the constraint that cells with the same height must be placed together in an "island" of cell rows. We apply integer linear programming (ILP) to solve the row assignment problem and improve the runtime of ILP with clustering, using a cost function that combines half-perimeter wirelength and displacement from a starting unconstrained placement. Considering the row assignment solution, we define fence-regions, which enable an existing place-and-route (P&R) tool to place the cells while considering the row-island constraints. Experimental results show that our proposed method can on average reduce final-routed wirelength by 8.5% and total power by 3.3%, with worst negative slack and total negative slack reductions of 24.0% and 13.0%, compared with the previous state-of-the-art method [10].**

I. Introduction

In sub-5nm nodes, it is necessary to scale down the height of standard cells to enhance density, while preserving the power and performance, along with area (i.e., PPA) benefits. Simply reducing the contacted poly pitch and local metal pitch to decrease standard-cell area has encountered limitations [1]. Reduced fin count and track-height in standard cells has been the main path to improve layout density and energy efficiency [1], [12].[1] Recent works propose the use of diverse standard-cell heights – that are not integer multiples of a single row height – within a single P&R block [1], [11], [14].

Employing different track-heights of cells together has emerged as a means to optimize the design [8], [9]. Specifically, the integration of non-integer multiple track-heights of standard cells has advantages of area, since this enables reduced redundant cell area. Thus, *mixed track-height design* leads to a more extensive design solution space, allowing for further optimization of power and cell area [4].

Dobre *et al.* [4] introduced the first mixed track-height strategy by partitioning the region into subregions in which track-heights are specified (Fig. 1(a)). They employed dynamic programming to divide the design area and perform timing-aware placement legalization, achieving reduced area and power while maintaining performance. Chen *et al.* [2] performed region-based global placement by introducing non-smooth density constraints and a pseudo-net which connects same track-height cells to comprehend mixed track-height cells, resulting in shorter wirelength.

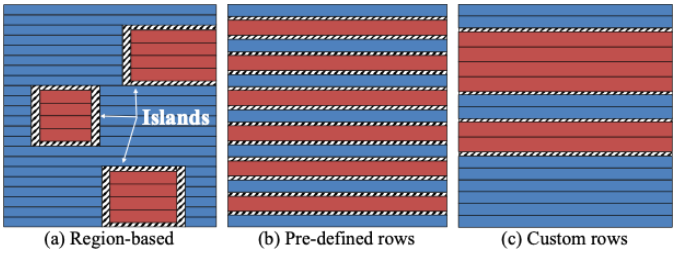


Fig. 1. Mixed track-height strategies. Short and tall cell rows are colored in blue and red, respectively. Hatched rectangles are the breaker cells.

There are two challenges in mixed track-height designs of this style. First, the insertion of breaker cells [2], [4], [10], both horizontally and vertically between cells of different track-heights, leads to a considerable expansion of the design area. Second, integrating different track-height cells within the same row results in misaligned power rails, making the designs more complex. To address these difficulties, a region-based placement strategy (Fig. 1(a)) is therefore adopted [7].

Another mixed track-height strategy makes the islands into wide and thin cell rows, and adheres to the *row-constraint* by placing cells of the same track-height within the corresponding row. For instance, the TSMC N3E node employs pre-determined alternating rows of two different track-heights (Fig. 1(b)); this FinFlex^{TM} approach enables increase of design flexibility within these pre-determined rows [15]. However, in theory, customizing the track-height of each row during the placement stage allows less space taken up by breaker cells and flexible designs without the constraint of having predetermined rows (Fig. 1(c)).

The work by Lin *et al.* [10] is the first attempt to place mixed track-height cells with customized row-constraint. The authors demonstrate that row-constraint placement reduces wirelength and power compared to the region-based placement. Starting with an unconstrained initial placement of cells, they employ k-means clustering of cell y-coordinates to assign a particular track-height to each given row, then move the cells to fit into rows with corresponding track-heights. However, this method is tested on a limited number of cases, and the resulting displacement can notably increase final wirelength.

In this paper, we follow the motivation of [10], and study the **row-constraint placement problem** (RCPP) using mixed track-height cells. Solving the **row assignment problem**

This work was partially supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT). (No. RS-2023-00222085, Development of memory module and memory compiler for non-volatile PIM). Partial support from DARPA HR0011-18-2-0032 (OpenROAD) and Google is gratefully acknowledged.

[1]Degrees of freedom include both the number of fins per device, and the fin height and fin pitch, in the most advanced FinFET technologies [5].

(RAP) well is the key to achieving a high-quality row-constraint placement, and can lead to significantly reduced wirelength. We solve the RAP using an optimal integer linear program (ILP) and expedite the runtime of ILP using 2-D k-means clustering. Using the results of the RAP, we enable an existing P&R tool to place the mixed track-height cells under the row-constraint. Furthermore, we conduct a comprehensive evaluation of our method to assess its effectiveness. The main contributions of this paper are:

- We introduce a *row-constraint placement* method using mixed track-height cells. The method employs ILP to solve a RAP, optimizing the displacement and wirelength.
- We apply 2-D k-means clustering to group the cells before solving the ILP, resulting in significant runtime reduction.
- We also demonstrate how to apply *fence-regions* in an existing commercial P&R tool to achieve a final placement consistent with the row assignment solution.
- For OpenCores testcases, we reduce routed wirelength by 8.5% and total power by 3.3% on average, with no loss of timing, compared to [10].

II. Problem Statement

The **row-constraint placement problem (RCPP)** is to optimize the unconstrained initial placement under row-constraint and other layout constraints. In this paper, we use short 6-track cells (6T) and tall 7.5-track cells (7.5T) of ASAP7 [3], and set 7.5T cells as minority cells.[2] A modified Library Exchange Format (mLEF) technique [4], [10] is applied to mixed track-height cells (6T and 7.5T cells) to produce the unconstrained initial placement with same height of cells.

- Input: unconstrained initial placement design synthesized with mixed track-height cells.
- Output: row-constraint placement solution with mixed track-height rows.
- Objective: minimize the wirelength and power while preserving timing metrics.

RCPP has several constraints. All cells must be placed in the sites of the rows with no overlap (layout constraint). The design has rows of mixed track-heights, and each cell must be placed in a row that has the same track-height (row constraint). Between rows that have differing track-heights, vertical spacing is needed. All track-height rows are paired in twos due to the N-well sharing rule, with each pair of rows having the same track-height. Thus, the layout has an even total number of cell rows.

To solve the RCPP, the positions of the majority and minority rows must be determined. We introduce the **row assignment problem (RAP)**, which decides the track-height of each row in the design. Our proposed method solves the RAP by determining the position of the minority rows, such that the cell displacement and half-perimeter wirelength (HPWL) are minimized. Here, the term "row" denotes a consecutive pair of rows to satisfy the manufacturing (N-well sharing) constraint. Notations used in our work are summarized in Table I.

[2]Typically, no more than 30% of a well-optimized netlist will have high-drive instances (larger row height), i.e., minority instances.

TABLE I
NOTATIONS USED IN OUR WORK

Notation	Definition
N_{minC}	The number of minority cells
N_{minR}	The number of minority rows
$\mathcal{C}$	A set of clusters containing minority cells
$\mathcal{R}$	A set of rows in a design
$N_{\mathcal{C}}$	The number of clusters in set $\mathcal{C}$
$N_{\mathcal{R}}$	The number of rows in set $\mathcal{R}$
s	Clustering resolution
x_{cr}	Boolean indicator variable of positioning cluster c into row r
f_{cr}	Cost function contribution of positioning cluster c into row r
$y(\cdot)$	y coordinate function
$w(\cdot)$	Width function
α	The weight of the cost function

III. Proposed Method

A. Overall method

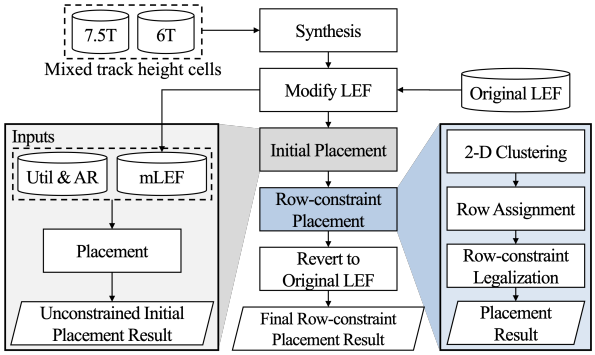


Fig. 2. Overall flow of producing row-constraint placement. The proposed method, row-constraint placement, is colored in blue. The netlist is synthesized using mixed track-height cells. Modified LEF (mLEF) enables an existing P&R tool to generate an initial placement with prescribed utilization (Util) and aspect ratio (AR). Row-constraint placement is conducted by clustering at the pre-processing stage, solving RAP, and placing the cells with row-constraint legalization. Reverting back to the original LEF produces the final row-constraint placement result.

Our proposed method proceeds in five main steps (Fig. 2). (i) First, logic synthesis generates the gate-level netlist that contains both 6T and 7.5T cells. (ii) Then, we create mLEF [10] files for all standard cells by converting their geometries, while preserving individual cell areas. The purpose of mLEF is to transform different track-heights into the same cell height, which enables the existing P&R tool to (iii) generate the unconstrained initial placement. The cell height of mLEF is determined by considering the ratio of different track-height cells in the design [10] and manufacturing grid.

The generated unconstrained initial placement is used as the input for (iv) our proposed row-constraint placement method (Fig. 2, blue-colored box). Row-constraint placement first solves the RAP, then conducts row-constraint legalization to place the cells under the row-constraint. The RAP is solved using ILP, which decides the position of both majority and minority rows. Our approach to address the RAP is accelerated by clustering minority cells to significantly reduce the runtime of ILP. We then use the fence-region capability (e.g., *createInstGroup -fence*) of the existing P&R tool to obtain the placement solution under the row-constraint. The existing P&R tool conducts incremental placement from the initial solution by considering the minority rows as 'fence-regions'. (v) Then, the final row-constraint placement result is produced after reverting the mLEF cells into the original cells.

B. Clustering

To decrease the runtime of ILP in Section III-C, we conduct clustering before solving the RAP. The runtime of ILP depends on the number of x_{cr} variables, which is (# of cells)$\times\%min\times N_{\mathcal{R}}$ initially. When the design is large or $\%min$ is large, the runtime of ILP can increase significantly due to the massive number of x_{cr} variables. Therefore, we seek to reduce the number of variables by using a clustering technique.

We use a *clustering resolution* parameter, s, which determines $N_{\mathcal{C}}$ by multiplying the number of minority cells and s. A high s yields numerous small clusters and a fine-grained clustering solution, whereas a low s yields a small number of large clusters and a coarse-grained clustering solution. $N_{\mathcal{C}}$ cannot exceed the number of minority cells, so $0 < s < 1$.

We use 2-D k-means clustering to cluster the minority cells into $N_{\mathcal{C}}$ clusters. The initial centroids of the clusters are determined using $p \times p$ grid points, where $p = \lceil\sqrt{N_{\mathcal{C}}}\rceil$. We then exclude $(p^2 - N_{\mathcal{C}})$ points from the outer region of the grid. The clustering starts with the positions of the minority cells in the initial placement.Clustering reduces the number of x_{cr} variables to $N_{\mathcal{C}} \times N_{\mathcal{R}}$, which can be determined by adjusting s, and thereby significantly reduces runtime of solving ILP.

C. Row Assignment

We treat RAP as an optimization problem and employ ILP to minimize displacement and HPWL. Following the ILP solution, we determine the positions of minority rows, and concurrently allocate clusters – each containing several minority cells – into the specific minority rows.

The boolean variable (x_{cr}) indicates whether the minority cells within the cluster c are assigned to the row r. A given row r is automatically set to be a minority row if $\max_{c\in\mathcal{C}} x_{cr} = 1$; in other words, if any cluster c is assigned to the row r, then r becomes a minority row. All remaining rows become majority rows. We design the f_{cr} to consider both y-displacement and the ΔHPWL.

We state the optimization problem using the boolean variables and cost function of our ILP formulation. For the clusters $\mathcal{C} = \{c_1, c_2, ..., c_{N_{\mathcal{C}}}\}$ and all rows $\mathcal{R} = \{r_1, r_2, ..., r_{N_{\mathcal{R}}}\}$:

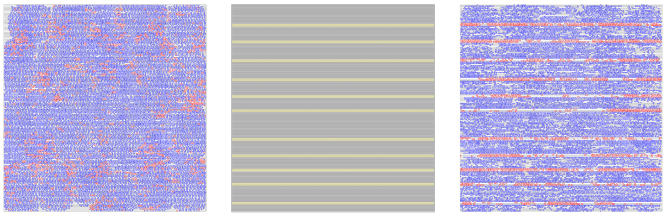
$$\min \sum_{c\in\mathcal{C}, r\in\mathcal{R}} f_{cr}x_{cr} \text{ for } x_{cr} = \begin{cases} 1 \text{ if } c \text{ is assigned to } r \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

$$f_{cr} = \alpha \cdot \text{Disp}(c, r) + (1 - \alpha)(\Delta\text{HPWL}(c, r)) \tag{2}$$

In our cost function (Eq. (2)), the displacement Disp(c, r) is calculated by summing up the y-displacements $|y(r) - y(\text{cell})|$ of each minority cell within the cluster c, and the ΔHPWL(c, r) accumulates the HPWL difference when a minority cell within the cluster c is moved vertically to the minority row r, while keeping the same x-coordinate. The parameter α weights y-displacement relative to ΔHPWL.

$$\sum_{r\in\mathcal{R}} x_{cr} = 1 \ \forall c \in \mathcal{C} \tag{3}$$

$$\sum_{c\in\mathcal{C}} w(c)x_{cr} \leq w(r) \ \forall r \in \mathcal{R} \tag{4}$$



(a) (b) (c)

Fig. 3. (a) Initial unconstrained placement input. (b) Fence-regions (colored in yellow) derived from the row assignment solution. (c) Final row-constraint placement result. Testcase: *aes_cipher_top, 360ps* with 60% utilization. Blue = majority cells (6T cells); Red = minority cells (7.5T cells).

$$\sum_{r\in\mathcal{R}} \max_{c\in\mathcal{C}} x_{cr} = N_{minR} \tag{5}$$

The unique assignment constraint (Eq. (3)) ensures that every cluster must be assigned exactly once. The row capacity constraint (Eq. (4)) specifies that the sum of cluster widths assigned to a row should not exceed the row's width. The width of the cluster is calculated by summing the width of minority cells within it. To help avoid undue wirelength increase due to minority cell relocation, the width of a minority cell is treated as the width of the original cell. Eq. (5) restricts the number of minority rows (N_{minR}).

The cost function of ILP includes both displacement and HPWL to prevent relocating the minority cells in ways that might increase wirelength. This approach helps ensure that the ILP produces a high-quality row assignment, given the capacity of each row, without an iterative process. Moreover, minority cell clustering contributes to speeding up the ILP.

D. Row-Constraint Legalization

Row-constraint legalization can be treated as a fence-region aware placement. By considering the minority rows as the fence-region, we enable the existing P&R tool to place the cells under the row-constraint while fully exploiting our row assignment solution. Starting from the initial unconstrained placement (Fig. 3(a)), we perform row-constraint legalization. We create a union of fence-regions (Fig. 3(b)) based on the positions of the minority rows in the row assignment solution. All minority cells are placed by the P&R tool within the fence-regions, while we prevent local resynthesis operations or buffering that can conflict with the row-constraint. Then the existing P&R tool can honor the row-constraint while placing the cells (Fig. 3(c)). With this method, we can freely assign all minority cells into the union of fence-regions.

IV. Experimental Setup and Results

In this section, we present our experimental setup of placing the mixed track-height cells. To show the efficiency of our method we run, and then compare the results of, five placement flows. Our results include post-placement, post-route results and runtime comparisons.

A. Experimental setup

We use ASAP7 [3] 7.5T (version 28) and 6T (version 26) cells for the mixed track-height cells in our design. Both RVT and LVT cells of each track-height are used together

TABLE II
SPECIFICATIONS OF 26 TESTCASES FROM NINE OPENCORES CIRCUITS

Bench name	Clock (ps)	# of cells	7.5T (%)	# of nets
aes_cipher_top	300	14040	28.13	14302
	320	13792	18.74	14054
	340	13031	13.94	13293
	360	12799	10.05	13061
	400	12419	5.27	12681
ldpc_decoder_802_3an	300	43299	23.79	45350
	350	42584	8.61	42584
	400	43706	3.62	45757
jpeg_encoder	300	50136	15.46	50158
	350	49449	10.70	49471
	400	47329	4.31	48129
fpu	4000	37739	17.50	37809
	4500	34945	10.36	35015
point_scalar_mult	200	55630	7.92	56172
	250	51556	4.87	52098
des3	210	57532	24.44	57766
	220	57851	21.27	58085
	230	57613	15.44	57847
	250	56653	10.17	56887
	290	55390	4.95	55624
vga_enh_top	270	73790	8.27	73879
	290	73516	3.80	73605
swerv	130	94333	9.07	95111
	550	89682	4.67	90460
nova	300	174267	9.75	174418
	500	155536	5.59	155687

TABLE III
COMPARISON OF FIVE PLACEMENT FLOWS

Flows	(1)	(2)	(3)	(4)	(5)
Row Assignment	None	Previous [10]	Previous [10]	Ours	Ours
Legalization	None	Previous [10]	Ours	Previous [10]	Ours

and typical corner is used. For evaluation, 26 testcases from nine OpenCores circuits [16] are used (Table II). During the synthesis stage, we change the clock period for each circuit to make various testcases with different percentages of minority cells. We synthesize all designs by using *Synopsys Design Compiler O-2018.06* [17].

We generate and apply mLEF [10] files to ensure that different track-height cells have the same height. In this way, the mLEF enables the commercial tool (*Cadence Innovus v21.16-s078_1* [18]) to produce an unconstrained initial placement. For all testcases, utilization is set to 60% and aspect ratio of the initial placement is 1.0. Our proposed method and methods related to mLEF are implemented with C++ and *CPLEX 22.1.1* [19]. *Cadence Innovus v21.16-s078_1* [18] is used to conduct row-constraint legalization and to finish routing after placement. In our row-constraint legalization, we set *dont_touch* to all instances to prevent buffering or local resynthesis operations that can conflict with the row-constraint. Our benchmarks and scripts will be open-sourced [6]; all data is available at [20].

We compare our five placement flows in Table III. The initial unconstrained placement, Flow (1), does not consider row-constraint. The unconstrained placement is invalid, because it uses mLEF cells, but is included as a standard baseline. In general, row-constraint placement imposes a harsh constraint compared to unconstrained placement, resulting in PPA overhead. Flows (2) and (3) use the reimplemented state-of-the-art row-constraint approach [10][3] to compare with Flows (4) and (5) which use our proposed row assignment method. Flows (2) and (4) use the same legalization, which modifies the Abacus method [13] under row-constraint [10], whereas Flows (3) and (5) use the proposed row-constraint legalization. For fairness, we set N_{minR} to match the result from the Flow (2). All flows start from the same initial unconstrained placement that uses mLEF cells, but the final placement results, except for Flow (1), have mixed track-height cells.

[3]No code or executable was available from authors of [10].

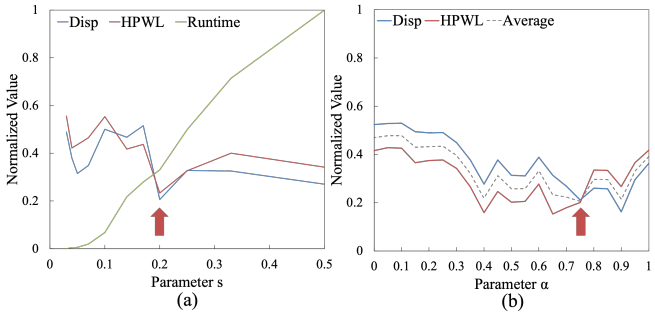


Fig. 4. (a) Normalized displacement, HPWL and ILP runtime of sweeping s. (b) Normalized displacement and HPWL of sweeping α.

B. Experimental results

1) Parameter determination. By sweeping the parameters and observing the quality of results (QoR) in the post-placement stage, we determine the parameters α and s based on the minimal displacement, HPWL and runtime of solving ILP as shown by the red arrows in Fig. 4. We use 14 testcases among Table II covering all circuits and various 7.5T% values. The QoR and runtime are 0-1 normalized for each testcase (i.e., scaled within the range of 0 to 1), then we average them over all testcases.

We observe that s affects the QoR more than α does. Therefore, we first tune s which determines the number of clusters, then tune α which weights the displacement relative to HPWL in ILP. We choose $s = 0.2$, which makes displacement and HPWL drop with the least runtime. We select $\alpha = 0.75$, which reduces both displacement and HPWL.

2) Post-placement results. We compare the five placement flows according to total displacement and HPWL of the design at the post-placement stage (Table IV). Total displacement is the sum of each instance's delta distance from the initial unconstrained placement (i.e., Flow (1)). Smaller displacement gives a similar result to initial placement, whereas smaller HPWL tends to result in a shorter routed wirelength. Therefore, small displacement and small HPWL[4] are preferred at the post-placement stage.

The proposed row-constraint legalization places all instances while being aware of fence-regions, but does not consider the initial placement. Thus, comparing displacement between flows that use the proposed row-constraint legalization is meaningless. However, we compared Flows (2) and (4), which have the same legalization method, to test the effect of

[4]Due to the multiple pins in a single cell, the HPWL increment relative to the unconstrained placement can be larger than the total displacement. Moreover, there can be HPWL distortion between mLEF cells and mixed track-height cells.

TABLE IV
POST-PLACEMENT RESULTS OF FIVE PLACEMENT FLOWS. BEST DISPLACEMENT AND HPWL AMONG FLOWS (2)[10], (3), (4) AND (5) ARE IN BOLD.

Testcase	Displacement (10^5um)				HPWL (10^5um)					Total Runtime (min)			
	(2)[10]	(3)	(4)[Ours]	(5)[Ours]	(1)	(2)[10]	(3)	(4)[Ours]	(5)[Ours]	(2)[10]	(3)	(4)[Ours]	(5)[Ours]
aes_300	0.63	2.81	**0.36**	0.92	1.60	2.29	2.11	1.83	**1.83**	0.2	3.2	1.4	4.8
aes_320	0.25	0.90	**0.24**	0.87	1.55	1.81	1.86	**1.79**	1.82	0.2	2.7	0.5	2.1
aes_340	**0.20**	0.81	0.21	0.83	1.49	1.69	1.73	**1.69**	1.73	0.2	2.7	0.3	2.0
aes_360	0.22	0.72	**0.16**	0.66	1.44	1.64	1.60	1.60	**1.58**	0.2	2.6	0.3	1.7
aes_400	**0.11**	0.63	0.12	0.61	1.38	1.54	**1.51**	1.57	1.51	0.2	2.4	0.2	1.7
ldpc_300	7.01	17.22	**2.55**	6.88	18.97	24.06	**19.74**	20.18	20.08	1.6	9.1	41.8	45.9
ldpc_350	0.84	5.70	**0.81**	5.60	17.33	18.14	18.16	**18.07**	18.11	2.1	6.6	5.8	9.7
ldpc_400	0.55	5.30	**0.55**	5.47	16.48	**17.06**	17.63	17.09	17.67	2.2	6.3	2.4	5.7
jpeg_300	4.21	24.95	**2.31**	4.37	6.40	11.41	10.36	7.46	**7.37**	2.9	7.5	14.8	16.0
jpeg_350	1.02	4.78	**0.96**	4.56	5.30	6.35	6.15	6.09	**5.95**	3.1	5.5	12.3	13.2
jpeg_400	**0.53**	4.24	0.54	4.22	4.72	5.51	4.87	5.46	**4.82**	2.9	5.5	4.2	4.8
fpu_4000	2.00	12.14	**0.94**	3.71	3.33	5.94	8.18	4.07	**3.91**	1.4	10.9	11.1	14.0
fpu_4500	0.55	2.95	**0.51**	2.85	2.78	3.50	3.48	3.48	**3.34**	1.5	5.5	3.8	6.1
point_200	1.44	6.37	**1.09**	5.99	7.14	8.95	8.99	**8.78**	8.99	3.4	6.0	10.5	13.6
point_250	0.96	5.49	**0.82**	5.21	6.44	7.86	7.63	7.72	**7.56**	3.2	7.3	4.0	6.3
des3_210	2.39	11.23	**1.58**	4.37	5.16	7.11	6.76	**6.14**	6.21	2.6	7.0	44.5	47.0
des3_220	2.17	4.95	**1.84**	4.53	5.47	6.77	6.63	6.48	**6.44**	2.8	6.4	42.0	43.7
des3_230	1.86	4.81	**1.12**	4.42	5.34	6.66	7.15	**6.14**	6.27	3.0	6.4	30.7	32.4
des3_250	0.99	4.70	**0.77**	4.09	4.88	6.16	6.48	5.91	**5.84**	3.2	5.8	15.7	17.0
des3_290	**0.72**	5.53	0.73	5.55	5.06	5.89	5.80	5.86	**5.78**	3.5	6.2	7.0	8.6
vga_270	1.74	9.49	**1.55**	9.44	8.06	10.15	11.63	**10.01**	11.54	6.2	11.7	21.9	25.7
vga_290	1.54	1.74	**1.32**	10.22	8.64	10.04	10.15	9.98	**8.72**	6.9	11.2	7.1	8.9
swerv_130	6.25	37.51	**2.76**	9.44	17.01	23.45	24.53	**19.40**	20.16	9.5	15.2	32.8	34.5
swerv_550	1.60	8.88	**1.60**	8.76	16.63	**19.10**	20.19	19.42	20.09	9.0	12.6	15.6	17.4
nova_300	6.22	74.74	**5.36**	84.01	18.89	27.72	31.82	**27.44**	29.31	36.9	28.0	98.0	82.3
nova_500	**5.10**	26.82	5.48	36.45	20.12	**24.27**	24.86	24.77	24.96	36.4	25.8	61.7	50.6
Normalized	1.000	5.285	**0.818**	4.731	0.804	1.000	1.014	0.938	**0.937**	1.000	4.638	5.109	7.612

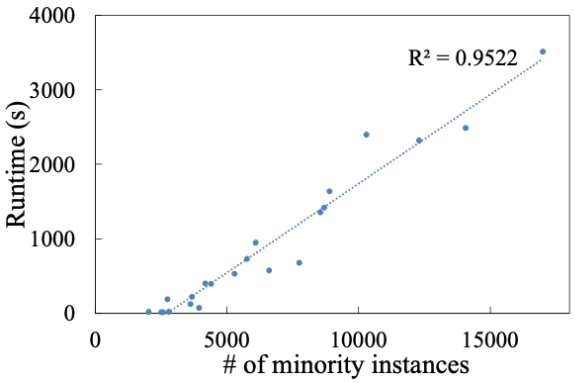


Fig. 5. ILP runtime of Flow (5), plotted against # of minority instances.

our row determination. Our proposed method to solve RAP reduces the displacement by 18.2% on average.

Compared to the previous state-of-the-art row-constraint placement (Flow (2)), our final proposed Flow (5) reduces HPWL by 6.3% on average. To assess the impact of our ILP row assignment solver, we compare the methods under the same legalization: the previous work's legalization, and our proposed row-constraint legalization. The comparison indicates that, on average, our row assignment reduces HPWL by 6.2% from Flow (2) to Flow (4) and 7.0% from Flow (3) to Flow (5).

3) Runtime comparison. We compare the total placement runtimes in Table IV. Due to the nature of ILP, the runtime of Flows (4) and (5) increase $\times$5.109 and $\times$7.612 on average compared to Flow (2), respectively.

We profile the runtime of each stage in Flow (5). We categorize the 26 testcases into three sets considering the number of minority instances: *small* (<3,000 minority instances; seven testcases), *medium* (3,000 to 5,000 minority instances; seven testcases), and *large* (>5,000 minority instances; 12 testcases). On average, the time for solving RAP is 4.95%, 30.57% and 72.60% for the *small*, *medium* and *large* sets, respectively, whereas the time for legalization is 95.04%, 69.41% and 27.37% for the same sets. The *small* set used a significant portion of time for legalization, whereas the *large* set mainly consumed time for solving RAP with ILP.

We plot the ILP runtime against the number of minority instances to show the scalability of our method to solve RAP (Fig. 5). The line of best fit shows a strong linear correlation with runtime. The proportion of time for solving RAP increases as the number of minority instances increases.

4) Clustering impact on ILP performance. We analyze how clustering decreases the ILP runtime at the post-placement stage. We compare the ILP-solving flow without clustering versus Flow (4) under the same legalization; the latter achieves 91.0% ILP runtime reduction with 5.2% and 1.0% displacement and HPWL overheads, respectively. Similar impact is seen from binding two adjacent cells together ($s = 0.5$), which yields 69.5% ILP runtime reduction with 0.4% and 0.2% displacement and HPWL overheads, respectively. Thus, clustering improves the runtime of ILP with minimal performance degradation.

5) Post-route results. We compare the post-route results for four flows: Flows (1), (2), (4) and (5) (Table V). We compare the post-route metrics of total routed wirelength[5], total power, worst negative slack (WNS), and total negative slack (TNS).

Flow (4) reduces routed wirelength by 7.6%, total power by 2.5%, WNS by 12.4% and TNS by 4.3% on average. This shows that our proposed row assignment using ILP is effective

[5]For the rank correlation of Flow (1), (2), (4) and (5) between Table IV (HPWL) and Table V (WL), 147 comparisons match among 156 comparisons.

TABLE V
POST-ROUTE RESULTS OF FOUR PLACEMENT FLOWS. BEST RESULTS AMONG FLOWS (2) [10], (4) AND (5) ARE IN BOLD.

Testcase	Wirelength (10^5um)				Total Power (mW)				WNS (ns)				TNS (ns)			
	(1)	(2)[10]	(4)[Ours]	(5)[Ours]	(1)	(2)[10]	(4)[Ours]	(5)[Ours]	(1)	(2)[10]	(4)[Ours]	(5)[Ours]	(1)	(2)[10]	(4)[Ours]	(5)[Ours]
aes_300	1.8	3.0	2.2	**2.2**	20.3	22.3	21.1	**21.1**	-0.163	-0.637	**-0.169**	-0.228	-34.7	-66.4	**-30.4**	-46.9
aes_320	1.8	2.2	**2.1**	2.2	19.2	20.1	**20.0**	20.0	-0.265	-0.288	-0.515	**-0.273**	-45.2	-56.0	-108.8	**-49.4**
aes_340	1.7	2.0	**2.0**	2.1	15.6	16.2	**16.2**	16.3	-0.480	**-0.439**	-0.481	-0.536	-98.3	**-108.0**	-108.1	-116.9
aes_360	1.7	2.0	**1.9**	1.9	13.7	14.2	**14.1**	14.1	-0.566	-0.475	-0.526	**-0.422**	-137.5	-122.2	-125.4	**-102.8**
aes_400	1.6	1.9	1.9	**1.8**	11.0	11.4	11.4	**11.4**	-0.403	**-0.477**	-0.599	-0.483	-104.0	-106.0	-141.2	**-100.5**
ldpc_300	26.1	32.5	27.8	**27.6**	309.9	367.3	322.3	**320.4**	-0.844	-1.785	-0.929	**-0.802**	-1235.4	-1590.2	-1327.3	**-1250.8**
ldpc_350	24.6	25.7	25.6	**25.6**	246.2	254.3	253.9	**253.0**	-0.846	-1.085	-1.021	**-0.817**	-1235.5	-1323.2	-1295.9	**-1226.9**
ldpc_400	23.7	**24.6**	24.6	25.3	197.5	**202.5**	202.5	205.6	-0.920	-0.926	-0.902	**-0.898**	-1279.5	-1334.7	-1342.7	**-1308.8**
jpeg_300	8.2	18.7	9.6	**9.6**	103.5	145.5	**107.8**	108.0	-1.384	-4.628	-0.930	**-0.897**	-1198.9	-2156.9	-1212.2	**-1197.3**
jpeg_350	6.6	7.9	7.6	**7.6**	82.7	86.3	85.4	**85.3**	-0.507	**-0.587**	-0.603	-0.644	-623.3	**-719.9**	-752.2	-777.8
jpeg_400	6.2	7.2	7.2	**6.7**	67.3	69.5	69.4	**68.3**	-0.601	-0.611	**-0.483**	-0.706	-1067.5	-1152.7	**-1034.5**	-1293.5
fpu_4000	4.2	8.8	5.2	**4.9**	5.0	5.8	5.2	**5.1**	-1.130	-6.572	**-1.402**	-1.435	-33.9	-304.8	**-43.2**	-44.9
fpu_4500	3.5	4.5	4.5	**4.3**	3.6	3.8	3.8	**3.8**	-1.186	-1.825	**-1.752**	-1.795	-35.6	-57.4	**-56.0**	-56.9
point_200	9.4	11.9	**11.5**	11.7	110.4	118.9	**117.4**	118.4	-0.527	-0.670	-0.643	**-0.603**	-2155.6	-2627.2	**-2577.0**	-2658.5
point_250	8.4	10.2	10.0	**9.9**	85.3	89.5	89.2	**88.8**	-0.721	-0.735	-0.704	**-0.641**	-2004.9	**-2259.6**	-2380.0	-2279.1
des3_210	6.3	8.9	**7.5**	7.5	199.4	211.9	204.5	**204.4**	-0.464	-0.954	-0.370	**-0.325**	-245.6	-298.0	-259.4	**-237.7**
des3_220	6.6	8.1	7.8	**7.7**	187.7	194.2	192.7	**192.2**	-0.248	-0.344	-0.379	**-0.248**	-197.0	-244.6	-257.8	**-213.3**
des3_230	6.4	7.9	**7.4**	7.5	171.8	178.3	**175.8**	176.0	-0.249	-0.291	-0.268	**-0.179**	-141.6	-195.2	-171.3	**-159.6**
des3_250	5.9	7.4	7.1	**7.0**	150.5	155.8	154.9	**154.2**	-0.158	-0.189	-0.172	**-0.152**	-129.8	-182.5	-170.2	**-152.4**
des3_290	6.2	7.1	7.1	**7.0**	124.9	127.9	127.9	**127.4**	-0.195	-0.222	-0.219	**-0.204**	-157.0	-206.2	-201.4	**-187.4**
vga_270	11.8	14.6	**14.5**	16.0	81.1	82.0	**82.0**	82.4	-2.589	-2.507	**-2.350**	-2.452	-7233.3	-9392.8	-9532.2	**-8826.0**
vga_290	12.4	14.4	14.4	**13.0**	75.0	75.6	75.6	**74.9**	-2.229	-2.709	-2.721	**-2.614**	-11327.3	-14743.0	-13591.9	**-11947.7**
swerv_130	21.5	33.2	**24.7**	25.4	129.4	146.1	**136.8**	139.8	-1.071	-4.510	**-1.108**	-1.499	-5201.0	-7155.9	-5769.0	**-5684.1**
swerv_550	21.2	**24.2**	24.6	25.1	30.6	**31.9**	32.1	32.0	-0.942	-1.315	**-1.045**	-1.269	-2125.7	**-2768.2**	-2924.1	-3109.9
nova_300	27.2	42.3	42.6	**38.9**	113.9	135.8	138.4	**122.2**	-2.144	-5.431	-6.055	**-2.278**	-34862.9	-52663.8	-60730.0	**-41825.1**
nova_500	31.1	37.9	38.9	**36.7**	66.5	71.0	72.7	**69.1**	-1.642	-3.114	-4.070	**-2.128**	-29789.6	-38588.3	-38212.1	**-33427.5**
Normalized	0.785	1	0.924	**0.915**	0.934	1	0.975	**0.967**	0.723	1	0.876	**0.760**	0.773	1	0.957	**0.870**

compared to [10]. Our final proposed Flow (5) shows the effective reduction of post-route metrics over Flow (2): i.e., reduction of routed wirelength by 8.5%, total power by 3.3%, WNS by 24.0% and TNS by 13.0%, on average.

6) Comparison with unconstrained placement. Row constraint placement design naturally brings overhead by requiring cells to be moved from an optimized initial solution. However, there are hidden additional area costs in unconstrained placement due to using mLEF instead of mixed track-height cells. As a result, directly comparing the row-constraint placement to unconstrained mLEF placement may be misleading. Nevertheless, it remains meaningful to reduce the overhead in row-constraint design compared with previous work [10].

Row-constraint placement incurs an overhead compared with Flow (1). At the post-placement stage, Flows (2) and (5) have an HPWL overhead of 26.6% and 17.2% on average, respectively. At the post-route stage, Flow (2) has 31.9% increase in routed wirelength and 7.6% increase in total power, while Flow (5) has 17.0% increase in routed wirelength and 3.6% increase in total power. Our final proposed Flow (5) has less overhead compared with Flow (2).

V. CONCLUSIONS

We have studied the RCPP in the context of mixed track-height design and proposed a novel flow to solve this problem. The *row assignment problem* is solved by using ILP under a row-constraint. A pre-processing stage that uses 2-D k-means clustering improves the runtime of ILP. *Row-constraint legalization* achieves a final row-constraint placement with an existing P&R tool, by applying fence-region constraints derived from the RAP solution. Our methods can significantly reduce routed wirelength and total power without significant timing degradation, compared to [10] (i.e., Flow (2)).

Additional practical ways to place the mixed track-height cells are emerging (e.g., [15]), so a future research direction might be to swap the track-heights of the cells and to place the mixed track-height cells on pre-determined alternating rows or other row patterns.

REFERENCES

[1] M. G. Bardon *et al.*, "Extreme scaling enabled by 5 tracks cells: Holistic design-device co-optimization for FinFETs and lateral nanowires," *IEEE IEDM*, 2016.
[2] J. Chen *et al.*, "An Efficient EPIST Algorithm for Global Placement with Non-Integer Multiple-Height Cells," *Proc. DAC*, 2020.
[3] L. T. Clark *et al.*, "ASAP: A 7-nm finFET predictive process design kit," *Microelectronics J.* 53, 2016.
[4] S. A. Dobre *et al.*, "Design Implementation With Noninteger Multiple-Height Cells for Improved Design Quality in Advanced Nodes," *IEEE TCAD* 37(4), 2018.
[5] M. Hatamian and P. Penzes, "Non-integer height standard cell library," *US Patent 8788998*, 2014.
[6] D. Junkin, "Supporting the Scientific Method for the Next Generation of Innovators", July 2022. https://open-source-eda-birds-of-a-feather.github.io/doc/slides/BOAF-Junkin-DAC-Presentation.pdf
[7] A. B. Kahng, "Advancing Placement," *Proc. ISPD*, 2021.
[8] S.-Y. Lee *et al.*, "RL-Legalizer: Reinforcement Learning-based Cell Priority Optimization in Mixed-Height Standard Cell Legalization," *Proc. DATE*, 2023.
[9] Y. Lin *et al.*, "MrDP: Multiple-Row Detailed Placement of Heterogeneous-Sized Cells for Advanced Nodes," *IEEE TCAD*, 37(6), 2018.
[10] Z.-Y. Lin and Y.-W. Chang, "A Row-Based Algorithm for Non-Integer Multiple-Cell-Height Placement," *Proc. ICCAD*, 2021.
[11] S. Ma *et al.*, "Low track height standard cell design in IN7 using scaling boosters," *Proc. SPIE*, vol. 10148, 2017.
[12] S. M. Y. Sherazi *et al.*, "Track height reduction for standard-cell in below 5nm node: How low can you go?," *Proc. SPIE Design Process Technol. Cooptim. Manuf. XII*, vol. 10588, 2018.
[13] P. Spindler *et al.*, "Abacus: Fast Legalization of Standard Cell Circuits with Minimal Movement," *Proc. ISPD*, 2008.
[14] T.-H. Wang *et al.*, "Six-track Standard Cell Libraries with Fin Depopulation, Contact over Active Gate, and Narrower Diffusion Break in 7nm Technology," *IEEE ISQED*, 2021.
[15] S.-Y. Wu *et al.*, "A 3nm CMOS FinFlex Platform Technology with Enhanced Power Efficiency and Performance for Mobile SoC and High Performance Computing Applications," *IEEE IEDM*, 2022.
[16] *OpenCores*. https://opencores.org/
[17] *Design Compiler User Guide vO-2018.06*, Synopsys, 2018.
[18] *Innovus User Guide v21.16*, Cadence Design Syst., 2021.
[19] *User's Manual for CPLEX v22.1.1*, IBM, 2022.
[20] https://github.com/minhyuk-kweon/DATE-2024.git