An Effective Legalization Algorithm for Mixed-Cell-Height Standard Cells

Chao-Hung Wang[1], Yen-Yi Wu[1], Jianli Chen[3], Yao-Wen Chang[1,2], Sy-Yen Kuo[1,2], Wenxing Zhu[3], and Genghua Fan[3]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan
[3]Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University, Fuzhou 350002, China
r03943174@ntu.edu.tw; yenyi@eda.ee.ntu.edu.tw; jlchen@fzu.edu.cn; {ywchang, sykuo}@ntu.edu.tw; {wxzhu, fan}@fzu.edu.cn;

***Abstract*—For circuit designs in advanced technologies, standard-cell libraries consist of cells with different heights; for example, the number of fins determines the height of cells in the FinFET technology. Cells of larger heights give higher drive strengths, but consume larger areas and power. Such mixed cell heights incur new, complicated challenges for layout designs, due mainly to the heterogeneity in cell dimensions and thus their larger solution spaces. There is not much published work on layout designs with mixed-height standard cells. This paper addresses the legalization problem of mixed-height standard cells, which intends to place cells without any overlap and with minimized displacement. We first study the properties of Abacus, generally considered the best legalization method for traditional single-row-height standard cells but criticized not suitable for handling the new challenge, analyze the capability and insufficiencies of Abacus for tackling the new problem, and remedy Abacuss insufficiencies and extend its advantages to develop an effective and efficient algorithm for the addressed problem. For example, dead spaces become a critical issue in mixed-cell-height legalization, which cannot be handled well with an Abacus variant alone. We thus derive a dead-space-aware objective function and an optimization scheme to handle this issue. Experimental results show that our algorithm can achieve the best wirelength among all published methods in reasonable running time, e.g., about 50% smaller wirelength increase than a state-of-the-art work.**

I. Introduction

Standard cells are one of the most popular styles in VLSI designs. Traditionally, a standard-cell library contains a variety of cells with the same height [8], [11] for easier design and optimization. For VLSI designs in advanced technologies, however, standard-cell libraries may consist of cells with different heights; for example, the number of fins determines the height of cells in the FinFET technology [4]. Cells of larger heights provide higher drive strengths and larger routing spaces, but consume larger areas and power; in contrast, cells of smaller heights have smaller areas and power consumptions, but offer weaker drive strengths and less routability. Because of the different properties and advantages with the two types of cells, designs with mixed cell heights become popular in advanced technologies [4]. For simple standard cells like inverters, the single-row-height structure may be sufficient. For some complex standard cells like flip-flops, on the other hand, double-row-height or even triple-row-height structure would be better than the single-row-height one because of the better routability and drive strength provided by a multi-row-height structure [1]. To satisfy multi-objective concerns in standard-cell designs (such as area, power, and routability), it is desirable to consider designs with mixed cell heights. However, such mixed cell heights would incur new, complicated challenges for layout designs, due mainly to the heterogeneity in their cell dimensions and thus their larger solution spaces.

Placement is a key step to VLSI layout design. Modern placement (especially, the dominant analytical placement) typically consists of three major stages: global placement, legalization, and detailed placement. Global placement computes the best position for each cell, ignoring the requirement of non-overlaps among cells. Legalization places cells into rows and remove cell overlaps such that cell displacements (or wirelength increase) are minimized, and detailed placement further refines the solution quality. Among these three stages, global placement does not handle the cell overlap issue, and detailed placement focuses more on local solution refinement; therefore, their corresponding problems for the designs with mixed cell heights are more similar to that with a single row height alone. In contrast, the legalization problem in the multi-row-height structure becomes much more complicated than that in the traditional single-row-height structure. The reasons are three-fold:

- Legalization of multi-row-height cells in one row would inevitably affect the configuration and feasibility in adjacent rows, which may further cause some domino effects on other distant rows.
- Legalization of multi-row-height cells need to additionally consider the cell boundary to power-rail alignment issue. For an odd-row-height cell, such alignment can be achieved also by vertical cell flipping, while there are two types of an even-row-height cell with either VDD or VSS running along its top and bottom boundaries. See Figure 1 for cells of different row heights and their alignments with the VDD/VSS power rails.
- The difference in the cell heights would inevitably induce more dead spaces in a legalized placement, which would in turn degrade the placement quality.

As a result, legalization becomes a crucial stage for dealing with this new, complicated challenge of placing standard cells with mixed cell heights to their desired locations.

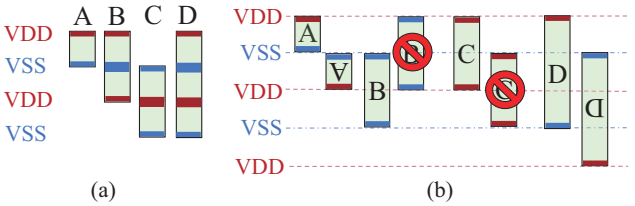


Fig. 1: Standard cells with different cell heights and power-rail alignment for mixed-cell-height placement.

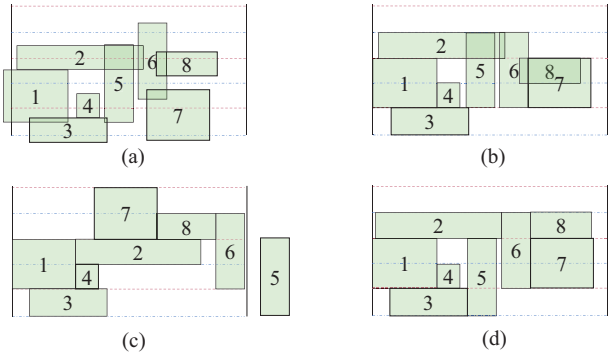


Fig. 2: (a) A given global placement result, where the cells are sorted by their x coordinates. (b) The solution obtained from [10] with cell overlaps. (c) A possible situation generated from [3], where Cell 5 cannot fit into the region based on the scenario described in the context. (d) A desired legalization solution obtained from our proposed algorithm.

There are many works on traditional legalization for the single-row-height standard cells. Among them, Tetris [6] and Abacus [10] are two most popular methods. Tetris first determines the cell legalization sequence by the x coordinate, and then greedily place cells row-by-row at the position with the smallest displacement within a given range. Tetris is very efficient, but Abacus typically can achieve better quality; especially, Abacus guarantees to achieve the optimal solution with minimum cell displacements for a row of cells with a predefined cell ordering. For each row, Abacus first assigns cells into rows, clusters overlapped cells inside a row, and then shift the whole cluster of cells with minimized displacement (or wirelength) to remove the overlaps. With multi-row-height cells, however, shifting cells in a row may incur overlaps in another row. See Figure 2(b) for Abacus's legalization on the instance shown in Figure 2(a), where cell overlaps remain after the legalization.

Not much work on legalization with mixed-height standard cells is reported in the literature. As mentioned above, unfortunately, most existing legalization methods cannot be directly applied to the mixed-cell-height legalization problem. (For example, see the problem shown in Figure 2(b).) The work [9] legalizes mixed single-row-height and double-row-height cells. It inflates or matches all the single-row-height cells into double-row-height cells, then transforms the mixed-cell-height problem into a single-

This work was partially supported by AnaGlobe, MediaTek, RealTek, TSMC, MOST of Taiwan under Grant NSC 102-2221-E-002-235-MY3, NSC 102-2923-E-002-006-MY3, MOST 103-2221-E-002-259-MY3, MOST 103-2812-8-002-003, MOST 104-2221-E-002-132-MY3, and NTU under Grant NTU-ERP-105R8951.

row-height one, and solves the transformed problem with a traditional legalization method. However, this work considers only single-row-height and double-row-height cells and does not handle the aforementioned power-rail alignment issue. Further, inflating single-row-height cells into double-row-height ones would inevitably incur significant dead spaces, thus degrading the solution quality.

The state-of-the-art work [3] addresses the general mixed-cell-height legalization problem, based on a greedy approach. According to the descriptions in [3], it first chooses an arbitrary cell, and places the cell into a local region that can accommodate this cell, if feasible. This method is fast. However, randomly choosing cells could change the original cell ordering, or even move cells to other random regions, which may significantly degrade the good quality obtained during global placement because the cell ordering often reflects the connectivity among cells, a phenomenon typically honored in global placement. Further, its selection tends to be local, possibly leading to a solution with limited quality. Further, to place multiple-row-height cells, it should find consecutive vacant rows to do so. However, finding consecutive vacant rows in a window could be hard, because of the limited space of the chosen windows. See Figure 2(c) for a possible case with such problems because of its randomness and local decisions below: Suppose that the cell ordering is $< c_2, c_1, c_7, c_8, c_6, c_3, c_4, c_5 >$. After legalizing c_2, the cell c_1 is to be legalized in the selected local region that contains the bottom four rows; it will be placed at middle two rows. Then, c_7, c_8, c_6, c_3, and c_4 are legalized one by one. Finally, because of the limited space, there is no legal position for the last cell c_5, and thus the method fails. (Note that Figure 2(c) just shows *one possible scenario* of the work [3]; due to its randomness, it could generate different results from that shown in Figure 2(c).)

To achieve high-quality legalization solutions, therefore, it is of particular importance to preserve the original cell ordering from global placement and work with a more global view, as shown in Figure 2(d). With these key observations in mind, we develop a new general mixed-cell-height legalization algorithm that tries to honor the good cell ordering from global placement and makes a more global decision. We observe that Abacus does possess these two good properties (and further the aforementioned optimality for a single row of well-ordered cells), although its operations may incur cell interferences among different rows for the mixed-cell-height problem. Therefore, we shall examine its properties and insufficiencies to derive an effective and efficient mixed-cell-height legalization algorithm.

The major contributions of our work are summarized below:

- We first study the properties of Abacus, theoretically analyze its capability and insufficiencies for tackling the new problem, remedy Abacus's insufficiencies, and extend its advantages to develop an effective and efficient algorithm for the mixed-cell-height legalization problem.
- We observe that dead spaces become a critical issue for mixed-cell-height legalization, which cannot be handled well with an Abacus variant alone. To handle this issue, we derive a dead-space-aware objective function and an effective optimization scheme.
- Inspired from Abacus, in particular, our proposed algorithm can obtain the optimal solution for some specific cases of the mixed-cell-height legalization problem. For other cases, our algorithm provides effective strategies to achieve desired, near optimal solutions.
- Experimental results show that our algorithm can achieve the best wirelength among all published methods in reasonable running time; for example, about 50% smaller wirelength increase than the state-of-the-art work [3], a best paper nominee at DAC'16.

The rest of this paper is organized as follows: Section II formulates the addressed problem. Section III presents our proposed algorithm. Section IV analyzes our proposed algorithm. Section V shows the experimental results and Section VI gives our conclusions and future work.

II. Problem Statement

For mixed-cell-height legalization, we are given a global placement solution, possibly with cell overlaps, and then asked to place cells into rows and remove cell overlaps such that the total cell displacement (and wirelength) from the given global placement is minimized. The problem addressed here is similar to that in [3], so below we will follow part of its formulation.

- **The Mixed-cell-height Legalization Problem:** Given a chip with a global placement of n standard cells $C = \{c_1, ..., c_n\}$, where each cell i has the respective height and width h_i and w_i, and the coordinate (bottom-left corner) (x'_i, y'_i), $\forall i$, $1 \leq i \leq n$, and each even-row-height cell has a boundary power-rail type VDD or VSS, the mixed-cell-height legalization problem places each cell c_i to the coordinate (x_i, y_i) and align its cell boundaries to corresponding power rails such that no two cells overlap with each other, and the total cell displacement is minimized.

The objective function is given by

$$\min \quad \sum_{i=1}^{n} (|x_i - x'_i| + |y_i - y'_i|) \quad , \tag{1}$$

while the constraints are described in more detail as follows:

1) Cells must be non-overlapped and inside the chip region.
2) Cells must be aligned in rows.
3) Cell boundaries must match the VDD/VSS power rails: The VDD/VSS power rails are routed horizontally between the rows, each standard cell has power rails running through its two cell boundaries and also row intersection(s) for a multi-row-height cell; see Figure 1(a) for an illustration. Specifically, odd-row-height standard cells must be placed to match the correct VDD/VSS power rail, possibly by vertical cell flipping. For an even-row-height cell, however, its two boundaries are designed for either VDD or VSS. Accordingly, we must match the correct power rail for an even-row-height cell. See Figure 1(b) for an example.

For Constraints (1) and (2), the mixed-cell-height legalization problem can be formulated as the following mathematical model:

$$\begin{array}{lll} \min & \sum_{i=1}^{n} (|x_i - x'_i| + |y_i - y'_i|) & \\ \text{for each row s.t.} & x_j - x_k \geq w_k, & \text{if } x_j \geq x_k \\ & x_k - x_j \geq w_j, & \text{if } x_k > x_j, \\ \text{where} & j, k \in \{1, 2, ..., n\}. & \end{array} \tag{2}$$

The mixed-cell-height legalization problem is NP-hard if its cell ordering is not given, where this problem becomes a linear placement problem (LPP) [7]. If the cell ordering is given, e.g., aligning cells according to their sorted x-coordinates without altering this order, the mixed-cell-height legalization problem can be solved by quadratic programming. Solving quadratic programming with inequality constraints is time-consuming in general. Placing a new cell to a row, we choose the row with the minimum displacement. This procedure needs $O(r)$ calls to a quadratic program solver, such as OOQP [5], where r is the number of rows. Because there are n cells, it needs totally $O(rn)$ calls to a quadratic program solver. Unlike Abacus which only optimizes cell displacement in a single row, the mixed-cell-height legalization problem needs to optimize cell displacements in the whole chip region. Thus, solving the mixed-cell-height legalization problem with quadratic programming alone would be very time-consuming.

Further, there is another challenge in mixed-cell-height legalization. Unlike traditional single-row-height cell legalization, mixed-cell-height legalization may produce significant dead spaces, due to the non-uniform cell heights. Those dead spaces would make the chip area larger and the solution quality lower.

III. The Proposed Algorithm

In this section, we shall detail our algorithm. Because our algorithm mainly extends Abacus to mixed-cell-height legalization, we shall first examine Abacus's properties. Then, we introduce our algorithm flow. Further, we describe how to find suitable locations for placing cells. Finally, we present a core subroutine of our algorithm, called $Multi\text{-}PlaceRow$.

A. Abacus Review and Extension

Abacus [10] is generally considered the most effective approach for traditional standard cell legalization. It is based mainly on quadratic programming and dynamic programming. Assume that each row has N_r standard cells, and all the cells are sorted by their x-coordinates in global placement, i.e., $x'_i \geq x'_{i-1}$, and each cell has its own weight e_i (e.g., number of pins). Abacus formulates the legalization problem into quadratic programming as follows:

$$\begin{array}{ll} \min & \sum_{i=1}^{N_r} e_i (x_i - x'_i)^2 \\ \text{s.t.} & x_i - x_{i-1} \geq w_{i-1}, \quad \text{for } i = 2, 3, \ldots, N_r, \end{array} \tag{3}$$

However, applying dynamic programming to the mixed-cell-height legalization problem directly may cause the following problems:

- If we extend the dynamic programming method to deal with the mixed-cell-height legalization problem row by row, it may cause another row illegal, as shown in Figure 3. Suppose that the cell ordering is $< c_1, c_2, c_3, c_4 >$, and we are to place the target cell c_4 into row 1. Because c_3 overlaps with c_4, they will be clustered together to find their optimal positions. This procedure will make c_3 and c_2 overlap, generating an illegal solution.

- We could treat the maximum cell height as a row, and extend the dynamic programming method to deal with the mixed-cell-height legalization problem directly. Nevertheless, this method could cause a large number of dead spaces, as shown in Figure 4. Here, suppose that the cell ordering is $< c_1, c_2, c_3, c_4 >$. After doubling the single-row height cells, we apply the dynamic programming method. After placing c_1 into its optimal position, the next target cell c_2 will overlap with c_1. Then, c_1 and c_2 are clustered together to find the optimal position of the cluster. It continues with the same procedure until all cells are placed. Although this method generates a legal solution, it leads to a large number of dead spaces and thus poor solution quality.

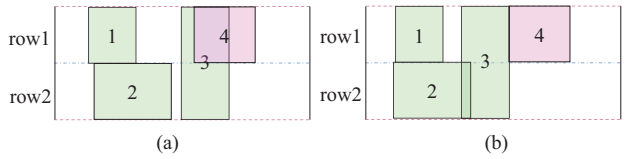


Fig. 3: Suppose that the cell ordering is $< c_1, c_2, c_3, c_4 >$. After legalizing the first three cells, the cell c_4 is to be legalized into row 1. However, legalizing row 1 will make the cells in row 2 overlapping.

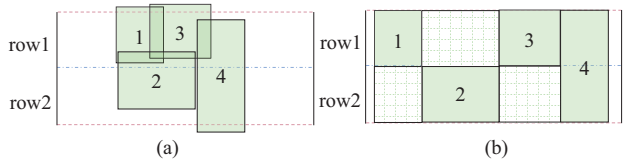


Fig. 4: Suppose that the cell ordering is $< c_1, c_2, c_3, c_4 >$. After doubling the single-row height cells, we apply the dynamic programming method. After placing c_1 into its optimal position, the next target cell c_2 will overlap with c_1. Then, c_1 and c_2 are clustered together to find the optimal position of the cluster. It continues with the same procedure until all cells are placed. Although this method generates a legal solution, it leads to a large number of dead spaces and thus poor solution quality.

We extend Abacus with additional processing to avoid its insufficiencies and propose a new mixed-cell-height legalization algorithm. In the following subsections, we detail our legalization algorithm.

B. Legalization Algorithm

Figure 5 shows the overall flow of our legalization. The input is a global placement result, which roughly spreads the cells on the layout. The objective of our algorithm is to align the standard cells into rows without overlaps.

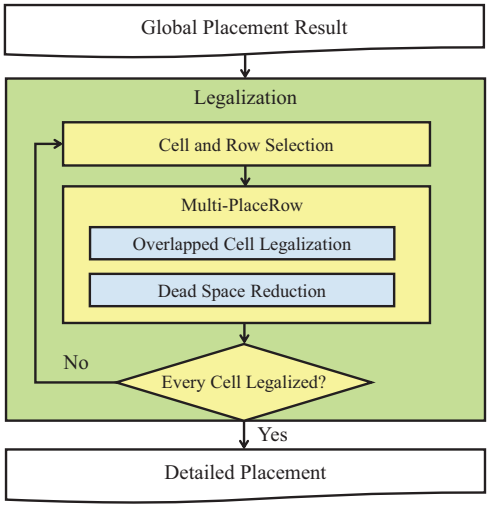


Fig. 5: Overall flow of our legalization.

First, we sort cells by their x-coordinates, then we place cells one by one in non-decreasing order. We try to place cells into a row among neighboring rows with the minimum cost; note that double-row-height cells, which need to avoid violating their power rail constraints, can only be placed in corresponding rows. After determining the row, we aim at finding an available dead space with the minimum cost to place the cell. If there exists such a dead space, the target cell will be tried to be placed in this dead space and compute the cost by the special subroutine called $Multi\text{-}PlaceRow$,

Algorithm 1 Mixed-Cell-Height Legalization

Input: Global placement result
Output: Legalization result
1: Sort cells by their x-coordinates in non-decreasing order
2: **for** each cell c_i
3: $cost_{best} \leftarrow \infty$
4: **for** each row r
5: $cost_1 \leftarrow \infty$
6: Place cell c_i into row r
7: $Multi\text{-}PlaceRow(r, false)$
8: //$false$: not placing cell into a dead space
9: Determine $cost_1$
10: $cost_2 \leftarrow \infty$
11: **if** an available dead space exists
12: $Multi\text{-}PlaceRow(r, true)$
13: //$true$: placing cell into the dead space
14: Determine $cost_2$
15: **if** $cost_1 < cost_2$
16: $cost \leftarrow cost_1$
17: Available dead space $DS \leftarrow false$
18: **else**
19: $cost \leftarrow cost_2$
20: Available dead space $DS \leftarrow true$
21: **if** $cost < c_{best}$
22: $cost_{best} \leftarrow cost$
23: $r_{best} \leftarrow r$
24: $DS_{best} \leftarrow DS$
25: Remove the cell c_i from row r
26: Place cell c_i into row r_{best}
27: $Multi\text{-}PlaceRow(r_{best}, DS_{best})$

which can effectively legalize the cells. After finding the minimum cost row and dead space, we place it by $Multi\text{-}PlaceRow$, to be explained later. We continue with these processes until all cells are placed.

Our algorithm is summarized in Algorithm 1. Line 1 defines a non-decreasing order for cells to be placed. Lines 2–27 legalize all the cells. Lines 4–25 try every row and dead space to find the one with the minimum cost. Lines 5–9 try the row r without placing any cell into a dead space, and compute $cost_1$. The $false$ for the subroutine $Multi\text{-}PlaceRow$ in line 7 means not to place a cell into a dead space. Lines 10–14 try to place a cell into a dead space, and compute $cost_2$. Lines 15–20 compare the costs to decide whether the cell should be placed into a dead space or not. Lines 21–24 save the best row and the decision of placing into a dead space or not. Line 25 removes the placing cell to try another row. Lines 26–27 place the cell in the minimum cost site. In this algorithm, $Multi\text{-}PlaceRow$ is the core subroutine in handling every situation that may occur in mixed-cell-height legalization. In the rest of the section, we shall detail this subroutine.

C. Cell and Row Selection

For better preserving the cell ordering from global placement result, we sort the cells by their x-coordinates obtained during global placement in non-decreasing order. With the cell ordering, we place the cells one by one in this order.

To compute the cost of each row, we introduce a new cost function to avoid excessive dead spaces as follows:

$$\min \sum_{i=1}^{N_r} (|x_i - x_i'| + |y_i - y_i'|) + \alpha \times D, \tag{4}$$

where α is a user-defined parameter, and D is the resulting dead space, included in the area measurement. The larger the α, the more compact the layout. For some high-density layout designs, it is desirable to have a more compact layout to ensure that all cells can be placed in the region.

The reasons to use the above dead space penalty are as follows:

- In order to consider the VSS/VDD alignment, it may generate significant dead spaces. For even-row-height cells, the cells with different power rail types would cause dead spaces.
- Figure 6(a) shows that the target cell c_4 is overlapped with a previous cell. Moving the target cell to a neighboring blank space can reduce the dead space. In this case, we observe that not only the dead space is reduced, but the cell also gets a smaller displacement. So dead spaces are a critical issue to be considered.
- Figure 7(a) shows the result after inserting triple-row-height cells, which contains many more dead spaces. As illustrated in Figure 7(b), if we consider the dead space issue, we can get a more compact result.

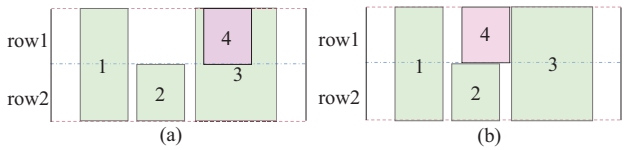


Fig. 6: (a) A target cell overlaps with its previous cell. (b) Changing the cell order can reduce dead space.

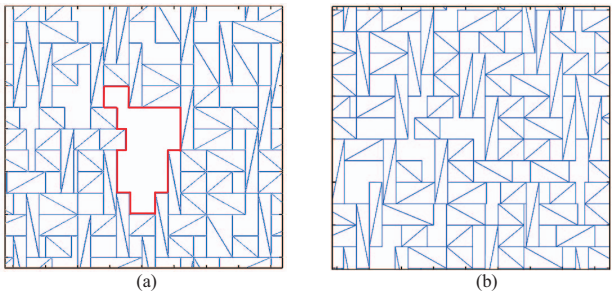


Fig. 7: (a) Adding triple-row height cells, there would be many more dead spaces without considering the dead space issue. (b) After considering the dead space issue, the resulting layout could be more compact.

In our algorithm, finding the minimum cost row is important to get better results. We can simply add a cost of dead spaces to avoid generating too many dead spaces. To ensure that we can find the minimum cost row, we try to place cells in each row and compute their costs. However, it may be very time-consuming to perform this process, so we speed it up by only placing cells into their neighboring rows. When each cell only needs to consider a constant number of rows, the running time of this algorithm can be significantly reduced. As will be shown in our experimental results, this scheme is efficient and can also preserve the same results as those from the original algorithm.

D. Multi-PlaceRow Method

In Abacus, there is a subroutine called $PlaceRow$, which can legalize overlapping cells and help the weighted cells find their optimal positions. We extend this subroutine to handle mixed-cell-height cells in a proposed subroutine, called $Multi\text{-}PlaceRow$. $Multi\text{-}PlaceRow$ is the core part of our algorithm. When a cell overlaps with another cell, the two cells will be legalized based on their original positions and weights, where the positions are given from global placement and the weights are defined by their numbers of pins. After legalization, these cells will be grouped as a cluster, which contains the information of these two cells. A cluster can be treated as a super cell, which can reduce significant runtime when solving overlaps with other cells.

After two cells are legalized, these two cells do not overlap with each other, and we obtain their minimum horizontal average displacement based on their weights. To legalize cells, we modify the legalization method of Abacus, which uses simplified quadratic programming to get the minimum horizontal average displacement of two overlapped cells.

When cell c_i is to be placed, it is treated as a cluster, which will update row r_j, the j^{th} occupied row of the cell. According to the occupied rows, we update the coordinates x_{lj} and x_{rj}, which are the leftmost and the rightmost x-coordinates of j^{th} row respectively.

When a cluster overlaps with another cluster, we legalize it with the minimum horizontal average displacement. We then cluster these two clusters, and update their new positions, total weights, j^{th} occupied row, and the leftmost and rightmost x-coordinate x_{lj} and x_{rj}. If there exist dead spaces after clustering, we update $deadspace_j$ which is the last dead space in the j^{th} row (with two x-coordinates to determine the range of dead space).

In single-row-height legalization, there can be one overlap at most because a cell only needs to be placed in a row. However, in mixed-cell-height legalization, a cell may overlap with two or more cells at the same time. We will choose the cell whose $x_i + w_i$ is the largest (rightmost) to be legalized first, where x_i and w_i are the x coordinate and the width of cell c_i respectively. See Figure 8 for an example, where the target cell c_5 overlaps with two cells c_2 and c_3 at the same time. We will legalize the critical overlapped cell c_3 first. If there still exists an overlap, we will continue with the legalization process until all cells are non-overlapping.

Performing mixed-cell-height legalization may incur dead spaces, which could still accommodate cells. See Figure 9 for an example that a dead

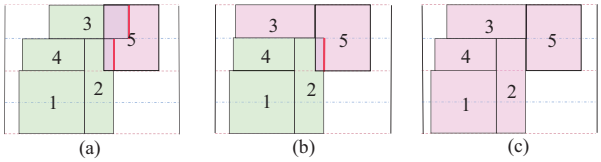


Fig. 8: (a) Two overlaps occur at the same time. (b) Legalize the critical cell first. (c) Legalize all the cells.

space can accommodate a target cell. If a cell can be placed in a dead space, we will try to do so and get the cost of this process. The cost of dead spaces can be negative when a cell is placed in a dead space, implying that the total cost will be reduced because of the better area utilization; on the other hand, when a cell incurs a dead space, its cost will be increased. After performing these processes, the dead space can be reduced effectively.

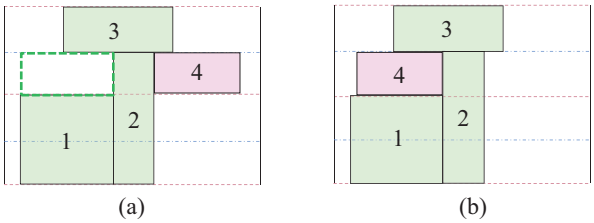


Fig. 9: (a) Legalize a target cell in the original process. (b) Legalize the target cell by placing it in the dead space.

IV. Analysis on the Multi-PlaceRow Method

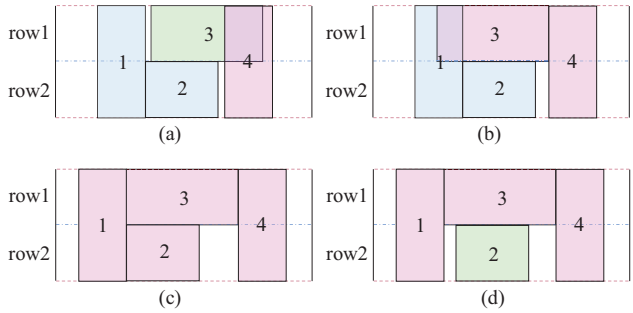


Fig. 10: (a) Placing the cell c_4. (b) After legalizing c_3 and c_4, a new overlap occur between c_3 and the cluster of c_1 and c_2. (c) Cell c_2 loses its best position, because there is enough space to place c_2 in its original position. (d) The optimal solution.

In the mixed-cell-height standard cell legalization problem, cell overlaps among rows are dependent and may interfere with each other. If a cell is shifted in one row, it may cause cell overlap in another row. Placing and legalizing any cell would need to consider the situation in more than one row [3]. Analyzing Abacus [10] for the multi-cell-height standard cell legalization problem, we find that its dynamic programming method cannot guarantee the optimum solution even for the case with only two rows.

As an example shown in Figure 10(a), assume that cells 1, 2, and 3 are placed optimally, the last cluster in row 1 is $\{c_3\}$, and the last cluster in row 2 is $\{c_1, c_2\}$. If a new cell 4 is to be inserted, applying the dynamic programming method on row 1, the cluster $\{c_1, c_2\}$ must be shifted because cells 1 and 3 would overlap as shown in Figure 10(b). Figure 10(c) shows the final legalized result by the dynamic programming method. It is obvious that the result in Figure 10(c) is not optimal. Instead, the optimal solution is shown in Figure 10(d).

In Section III, $Multi\text{-}PlaceRow$ is used to remove the cell overlap and find the best position for each cell. It is desirable to develop a scheme for the dynamic programming method to maintain its optimality as much as possible. In the following, we explore such strategies for $Multi\text{-}PlaceRow$, where the key issue lies in determining the cells positions and updating the clusters.

Given a cell ordering $< c_1, c_2, \ldots, c_n >$, suppose that cells $c_1, c_2, \ldots, c_k$ $(1 \leq k < n)$ have been placed, the position for each cell is x_i, $i = c_1, c_2, \ldots, c_k$, and the clusters are known. If a new cell c_{k+1} is to be inserted, applying the $Multi\text{-}PlaceRow$ method could lead to different legalization cases. We will take different strategies for different cases to achieve desired solutions. We use an instance to better illustrate what cases

may occur, where cells c_1, c_2, c_3 and c_4 are placed after applying the $Multi\text{-}PlaceRow$ method several times. If a new cell c_5 is inserted, one of the following cases may induce. Without loss of generality, we assume that cell c_5 is a double-row-height standard cell and the double-row-height legalization problem is considered. Our analysis can readily extend to more general cases for multi-row-height legalization problem, which would be more complex for explanation.

A. Case 1:

In Figure 11(a), if both of the last cluster $\{c_2, c_3\}$ in row 1 and the last cluster $\{c_4\}$ in row 2 do not overlap with the to-be-inserted-cell c_5, applying the $Multi\text{-}PlaceRow$ method on row 2, a new cluster containing cell c_5 is created. Figure 11(b) shows the final solution. In this case, a new cluster is generated, the position of the inserted cell is determined, and the optimality can obviously be guaranteed because there is not interference among the two clusters and cell c_5.

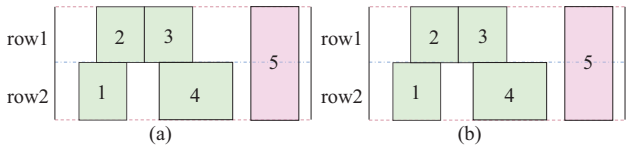


Fig. 11: (a) An initial global placement result. (b) Legalization result.

B. Case 2:

In Figure 12(a), if the to-be-inserted cell c_5 only overlaps with one row, i.e., the last cluster $\{c_4\}$ in row 2. Applying $Multi\text{-}PlaceRow$, the position of cell c_5 is determined, the cell c_5 is added to the last cluster in row 2 as shown in Figure 12(b), and the last cluster in row 2 is recursively legalized with its preceding cluster as long as these clusters overlap. There are three sub-cases when legalizing these clusters.

- Case 2-A: In Figure 12(a), after applying $Multi\text{-}PlaceRow$ on row 2, the position of cell c_5 is determined, and cell c_5 is added to the last cluster in row 2. If the preceding cluster of the last cluster only contains single-row-height standard cells, the basic idea of updating clusters in Abacus is used as long as the clusters overlap. For example, in Figure 12(b), because the penultimate cluster $\{c_1\}$ does not overlap with the last cluster $\{c_4, c_5\}$, and thus the previous clusters do not change.
- Case 2-B: In Figure 10(a), the last cluster in row 1 is $\{c_3\}$, and the last cluster in row 2 is $\{c_1, c_2\}$. We apply $Multi\text{-}PlaceRow$ on row 1, and Figure 10(c) shows the legalization result. In this case, cell c_2 is shifted to left. For each cell c_i in the last cluster (except the inserted cell c_{k+1}), if $d_i > 0$, where d_i is the distance cell c_i can shift, then we shift the cell c_i to its original position as much as possible.
  For cluster update, from the first cell to the last one (left to right) in the last cluster of row 1, if a cell is with a multiple row height, each single-row-height cell behind this cell is de-clustered, and new clusters are generated. For example, in the last cluster, a new cluster containing cell c_2 is generated. In this way, the optimality could be maintained.
- Case 2-C: Recursively legalizing with the preceding cluster as long as the clusters overlap, it may contain both of Case 2-A and case 2-B. The strategies discussed in the two cases can be used to deal with Case 2-C.

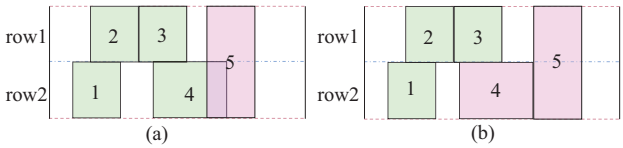


Fig. 12: (a) An initial global placement result. (b) Legalization result.

C. Case 3:

The to-be-inserted cell c_5 overlaps with both the last clusters in row 1 and row 2. $Multi\text{-}PlaceRow$ is called on the row with the larger overlapping area first. There are two subcases for consideration.

- Case 3-A: In Figure 13(a), we apply $Multi\text{-}PlaceRow$ on row 2 first. After applying $Multi\text{-}PlaceRow$ on row 2, if cell c_5 does not overlap with the last cluster in row 1, then Figure 13(b) gives the final solution. This subcase is similar to one of the sub-cases in Case 2, and the optimality could be maintained.
- Case 3-B: In Figure 14(a), after applying $Multi\text{-}PlaceRow$ on row 2, the last cluster in row 1 is $\{c_1, c_3, c_5\}$. If cell c_5 still overlaps with the last cluster on row 1 as shown in Figure 14(b), then we apply $Multi\text{-}PlaceRow$ on row 1 to remove the overlaps. Figure 14(c) shows the legalization solution. Assume that after applying $Multi\text{-}PlaceRow$ on row 2, the moving distance of cell c_1 toward left is d_1. And after applying $Multi\text{-}PlaceRow$ on row 1, the moving distance of cell c_1 toward right is d_2. If $d_2 > d_1$, we do not shift the cell c_1 as shown in Figure 14(d). Hence, in Case 3-B, for each c_i in the last cluster (except the inserted cell c_{k+1}), if $d_2 > d_1$, we do not shift the cell c_i. For this subcase, if it exists same situations as those in Case 2, the corresponding method in Case 2 can be applied.
  In the last cluster of row 1, from the first cell to the last cell (left to right), if a cell has a multiple row height, each single-row-height cells after this cell is de-clustered, and new clusters are generated. For example, in the last cluster, a new cluster contains cell c_1, and a new cluster with three cells are generated. The optimality could also be maintained.

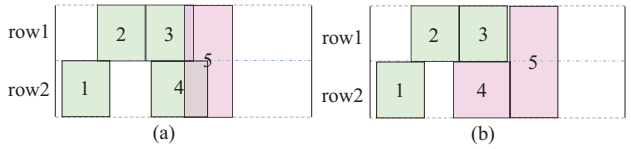


Fig. 13: (a) An initial global placement result. (b) Legalization result without overlapping with row 1.

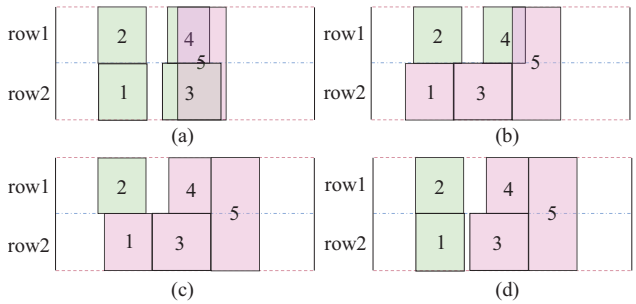


Fig. 14: (a) An initial global placement result. (b) Legalization result still with an overlap in row 1. (c) Because the weight of cell c_3 is large, the cell c_1 loses its best location. (d) Not to move cell c_1 for getting an optimal solution.

If an inserted cell has a triple row height, there could be four cases: non-overlapping, overlapping with one cell, overlapping with two cells, and overlapping with three cells. In each case, we can also determine the position of the cell and update the clusters. This can be extended to handle the general multi-row-height legalization problem, which will be more complex.

In $Multi\text{-}PlaceRow$, the number of clusters and the way of clustering affect the precision of the objective minimization. A larger number of clusters tends to lead to better results, but with a longer runtime. In contrast, the total movement increases as the number of clusters decreases. In $Multi\text{-}PlaceRow$, a cluster can not only be created or legalized, but also can be de-clustered. This is very different from Abacus. With the above methods to update clusters, $Multi\text{-}PlaceRow$ can make a better trade-off between the solution quality and runtime.

From the above analysis, we can claim the following theorem:

Theorem 1. *In the multi-row-height standard cell legalization problem, if every multiple-row-height cell is never overlapped and clustered with any other cells, the solution generated by $Multi\text{-}PlaceRow$ is optimal.*

The reason is that multiple-row-height cells are treated as a fixed macro, and the global position of a multiple-row-height cell is its optimal solution.

V. Experimental Results

The proposed mixed-cell-height legalization algorithm was implemented in the C++ programming language. We used the benchmarks provided by the authors of [3], which were modified by them from the 2015 ISPD Detailed-Routing-Driven Placement Contest [2]. These modified benchmarks do not need to consider fence regions given in the original benchmarks, and 10% of the cells were randomly selected to double their heights and half their widths to form mixed-cell-height benchmarks with single-row-height and double-row-height standard cells. This modification

TABLE I: Experimental Results

Benchmark	#S. Cell	#D. Cell	Density	GP HPWL (m)	Disp. (sites)			ΔHPWL			Runtime (s)		
					ILP	DAC'16	Ours	ILP	DAC'16	Ours	ILP	DAC'16	Ours
des_perf_1	103842	8802	0.91	1.43	2.13	3.32	4.21	2.61%	2.85%	0.99%	4098.7	7.2	18.0
des_perf_a	99775	8513	0.43	2.57	0.66	0.96	0.67	0.11%	0.28%	0.12%	193.8	2.6	6.1
des_perf_b	103842	8802	0.50	2.13	0.62	0.85	0.64	0.12%	0.31%	0.16%	250.8	2.4	6.2
edit_dist_a	121913	5500	0.46	5.25	0.45	0.47	0.48	0.09%	0.10%	0.12%	206.0	1.9	7.8
fft_1	30297	1984	0.84	0.46	1.58	1.81	1.65	2.25%	1.66%	0.89%	776.8	1.1	1.3
fft_2	30297	1984	0.50	0.46	0.66	0.86	0.65	0.55%	0.87%	0.67%	72.7	0.4	0.8
fft_a	28718	1907	0.25	0.75	0.60	0.64	0.59	0.32%	0.33%	0.29%	38.2	0.3	0.8
fft_b	28718	1907	0.28	0.95	0.73	0.80	0.69	0.32%	0.33%	0.30%	61.9	0.4	1.0
matrix_mult_1	152427	2898	0.80	2.39	0.49	0.53	0.47	0.36%	0.28%	0.21%	967.4	3.9	9.1
matrix_mult_2	152427	2898	0.79	2.59	0.45	0.49	0.42	0.30%	0.22%	0.17%	825.0	4.0	8.9
matrix_mult_a	146837	2813	0.42	3.77	0.27	0.33	0.27	0.09%	0.14%	0.09%	150.7	1.6	9.3
matrix_mult_b	143695	2740	0.31	3.43	0.25	0.30	0.25	0.09%	0.13%	0.09%	127.8	1.3	8.9
matrix_mult_c	143695	2740	0.31	3.29	0.27	0.29	0.28	0.11%	0.11%	0.11%	139.0	1.4	9.0
pci_bridge32_a	26268	3249	0.38	0.46	0.88	0.95	0.90	0.52%	0.58%	0.63%	49.4	0.3	0.8
pci_bridge32_b	25734	3180	0.14	0.98	0.95	0.96	0.90	0.12%	0.13%	0.06%	15.3	0.2	0.7
superblue11_a	861314	64302	0.43	42.94	1.85	1.94	2.14	0.15%	0.15%	0.26%	3073.6	23.4	80.2
superblue12	1172586	114362	0.45	39.23	1.45	1.63	1.55	0.18%	0.22%	0.22%	5079.0	106.5	91.1
superblue14	564769	47474	0.56	27.98	2.56	2.62	2.20	0.22%	0.22%	0.18%	3360.6	17.1	70.3
superblue16_a	625419	55031	0.48	31.35	1.61	1.73	1.87	0.10%	0.12%	0.11%	2470.7	21.7	61.0
superblue19	478109	27988	0.52	20.76	1.52	1.60	1.59	0.14%	0.14%	0.13%	1848.8	10.9	44.6
				Average	1.00	1.16	1.12	0.44%	0.46%	0.29%	1190.3	10.4	21.8
				N. Average	0.89	1.03	1.00	1.52	1.59	1.00	54.6	0.48	1.0

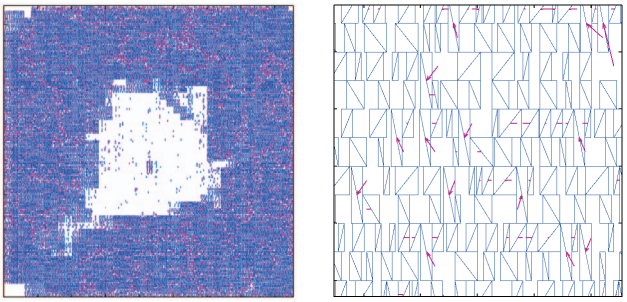
maintains the total cell area and ensures that each chip can accommodate all the cells. However, the binary code of the work [3] is not available to the public. So the results of the work [3] were taken *directly from the paper* (a best paper nominee at DAC16) for comparative study. Our experiments were conducted on a Linux workstation with Intel Xeon 2.93GHz CPU and 48GB memory, while the experiments of [3] were on a 64-bit Linux machine with Intel Xeon 3.4GHz CPU and 32GB memory, which is faster than our platform.

Table I shows the statistics of the benchmarks and the legalization results. In this table, "#S. Cell" gives the total number of single-row-height cells, "#D. Cell" the total number of double-row-height cells, "Density" the density of design, "GP HPWL" the wirelength of global placement in unit of meter, "Disp. (sites)" the displacement measured in the number of the placement site width, "ΔHPWL" the HPWL increase from the corresponding global placement result, and "Runtime" the running time in second. For the last three comparative metrics, "ILP," "DAC'16," and "Ours" give the respective results from an integer linear programming (ILP) based algorithm implemented by the authors of [3], the algorithm proposed in [3], and our algorithm, respectively.

The experimental results show that our algorithm can achieve the best quality among these three methods. Although the ILP method can achieve the lowest displacement, our work achieves 52% smaller HPWL increase rate than the ILP method. Compared with the work [3], our algorithm achieves 59% smaller HPWL increase rate and 11% smaller displacement. Because the work [3] uses greedy random cell and row selections to optimize cell legalization in local regions and does not honor the good cell ordering obtained from global placement, it runs faster, but incurs larger displacement and wirelength. In contrast, our method can preserve the original order from global placement and have a more global view for legalization optimization, thus leading to the highest quality among the three algorithms. Figure 15(a) shows the legalization result of the benchmark fft_2, and Figure 15(b) shows a partial layout which justifies that the order of cells is well preserved by our algorithm.

VI. Conclusions

This paper has presented a new mixed-cell-height standard cells legalization algorithm. We have studied the properties of Abacus, analyzed the capability and insufficiencies of Abacus for tackling the new problem, remedied Abacus's insufficiencies, and extended its advantages to develop an effective and efficient algorithm for the addressed problem. We have also observed that dead spaces imposes a critical challenge in mixed-cell-height legalization, which cannot be handled well with an Abacus variant alone. We have thus derived a dead-space-aware objective function and an optimization scheme to handle this issue. Experimental results have shown that our algorithm can achieve the best wirelength among all published methods in reasonable running time, e.g., about 50% smaller wirelength increase than the state-of-the-art work [3].



(a) (b)

Fig. 15: (a) Legalization result of the benchmark circuit fft_2. The cells are in blue, and the displacement in red. (b) Partial layout of (a).

References

[1] S.-H. Baek, H.-Y. Kim, Y.-K. Lee, D.-Y. Jin, S.-C. Park and J.-D. Cho, "Ultra High Density Standard Cell Library Using Multi-Height Cell Structure," in *Smart Materials, Nano-and Micro-Smart Systems*, pp. 72680C–72680C ,2008.

[2] I. S. Bustany, D. Chinnery, J. R. Shinnerl and V. Tutsi, "ISPD 2015 Benchmarks with Fence Regions and Routing Blockages for Detailed-Routing-Driven Placement," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ACM, pp. 157–164., 2015.

[3] W.-K. Chow, C-.W. Pui, F.Y. Young, "Legalization Algorithm for Multiple-Row Height Standard Cell Design," in *Proc. DAC*, 2016.

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

[5] E. M. Gertz and S. J. Wright, "Object-Oriented Software for Quadratic Programming," in *ACM Transactions on Mathematical Software*, 29.1: pp. 58–81, 2003.

[6] D. Hill, "Method and system for high speed detailed placement of cells within integrated circuit designs," in *U.S. Patent 6370673*, April 2002.

[7] A.B. Kahng, P. Tucker, and A. Zelikovsky, "Optimization of linear placements for wirelength minimization with free sites," in *Proc. ASP-DAC*, pp 1999.

[8] J. Wang, A. K. Wong, and E. Y. Lam, "Standard Cell Layout with Regular Contact Placement," *Semiconductor Manufacturing*, 17.3: pp. 375–383, 2004.

[9] G. Wu, C. Chu, "Detailed Placement Algorithm for VLSI Design with Double-Row Height Standard Cells," in *Proc. TCAD*, 2015.

[10] P. Spindler, U. Schlichtmann, F. M. Johannes, "Abacus: Fast Legalization of Standard Cell Circuits with Minimal Movement," in *Proc. ISPD*, pp. 47–53, 2008.

[11] L.-T. Wang, Y.-W. Chang, and K.-T. Cheng, *Electronic Design Automation: Synthesis, Verification, and Testing*, Elsevier/Morgan Kaufmann, 2009.