Legalization Algorithm for Multiple-Row Height Standard Cell Design [*]

Wing-Kai Chow, Chak-Wa Pui, Evangeline F. Y. Young
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong
{wkchow, cwpui, fyyoung}@cse.cuhk.edu.hk

ABSTRACT

Typical standard cell placement algorithms assume that all cells are of the same height such that cells can be aligned along the placement rows. However, modern standard cell designs are getting more complicated and multiple-row height cell becomes more common. With multiple-row height cells, placement of cells are not independent among different rows. It turns out that most of the commonly used detailed placement and legalization techniques cannot be extended easily to handle the problem. We propose a novel algorithm in handling legalization of placement involving multiple-row height cells. The algorithm can efficiently legalize a local region of cells with various heights, which is especially useful for local cell movement, cell sizing, and buffer insertion. Experiments on the application of the technique in detailed placement show that our approach can effectively and efficiently legalize global placement results and obtain significant improvement in the objective function.

Keywords

Detailed placement, legalization, multiple-row height cell, standard cell design

1. INTRODUCTION

With the increasing complexity in VLSI technology, standard cell design is a common approach for circuit design with millions of logic gates. Standard cell library consists of predesigned circuit units for common logic components. These basic units (cells) are placed on rows on the chip, powered by rails of power circuitry. The cells are connected by chip-level routing to perform the system functions. Standard cell library helps designers by shortening the development time. Time is saved by avoiding design circuit at transistor level, but focusing on chip-level instead. Since standard cells are aligned on the rows, they are designed to have fixed cell height. To increase cell density and thus reduce chip area and cost, cell heights are being reduced as much as possible. This can cause problems in standard cell design since complex standard cells with small cell height is hard to design due to serious routing congestion [1]. Therefore, it is increasingly popular to adopt multi-row height standard cell design, where simple cells are designed as single-row height, while complex cells are designed as double- or even multiple-row height [1, 2]. Comparing to single-row height cell structure, multi-row height cell structure can achieve better layout efficiency and thus better performance.

However, with multi-row height cells, the legalization and detailed placement problem is much more complicated. In original single-row height cell problem, cell overlappings are independent among rows. However, with multi-row height cells, shifting a cell in one row may cause cell overlap in another row. Therefore, placing and legalizing any single cell may need to consider more than one row. So far, there is not much academic research on the topic of legalization and detailed placement with multi-row height cells.

It turns out that the most common legalization and detailed placement techniques cannot be modified easily to handle multi-row height cells. The well-known legalization method, Abacus [3], involves assignment of cells to rows. Cells inside a row may be clustered and shifted to reduce the cost. However, with multi-row height cells, shifting of cells in a row may produce overlapping in another row. With the existing techniques, we can handle multi-row height cells as macros. The work of [4] supports legalization of mixed-size design. However, it is actually a two-step approach of handling legalization of macros first and then the single-row height standard cells. For other mixed-size placer like [5, 6] include an extension of a greedy legalization [7] for supporting mixed-size legalization. However, the placed objects are not allowed to move for accommodating other unplaced objects, which could result in high displacement when the design density is high.

In terms of detailed placement, the well-known technique of cell reordering enumerates all possible ordering of a set of consecutive cells and search for the one with the best objective cost [8]. However, reordering of cells with multi-row height cells in a row may induce cell overlapping in other rows. Another common detailed placement technique proposed by [8] and [9] involves solving a fixed order single row placement problem optimally. However, with multi-row height cells, overlap-free solution in a single row may involves cell overlapping in the row above and below the target row. Recently, Wu and Chu [10] proposed a wirelength-driven double-row height cell placement. However the work limits standard cell height of two and double-row height cells are restricted to be placed on even rows. Recent detailed placement technique proposed by [11] and [12] involves a technique of instant legalization, i.e., for every cell move, the detailed placer performs legalization such that all intermediate placement solutions are legal. Such technique can provide better control in solution quality and constraint satisfaction. This instant legalization technique is also very useful in other scenarios. For example, in gate sizing, we may want to locally legalize the placement after cell size changes. In buffer insertion, we may want to legalize the solution locally to remove overlapping induced by the newly inserted buffer. In this paper, we propose an incremental legalization technique with consideration of multi-row height cells.

The major contributions of our work include:

- Our work explores a new area of detailed placement with multi-row height cells.
- We propose a novel and effective algorithm to locally le-

[*]The work described in this paper was partially supported by grants from the Research Grants Council (Ref. No.: 14209214) and the Innovation and Technology Commission (Ref. No.: IT3/068/14) of the Hong Kong Special Administrative Region, China.

DAC '16, June 05-09, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4236-0/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2897937.2898038

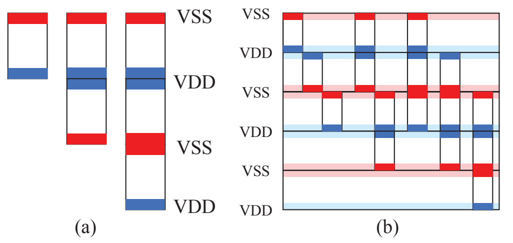

Figure 1: Power line layout on mutli-row height cell

galize a placement with multi-row height cells.

- The experimental results show the effectiveness of our algorithm.

The rest of the paper is organized as follows: Section 2 gives the formal formulation of the multi-row height cell legalization problem. Section 3 provides the overview of our incremental approach of legalization. Section 4 introduces the idea of our proposed efficient algorithm to solve the problem. Section 5 describes the details of our implementation which leads to a high algorithm performance. Section 6 shows the experimental results and analysis. Section 7 concludes the paper.

2. PROBLEM FORMULATION

Given a global placement solution, legalization is a process of moving cells to discrete legal positions and to remove all overlaps. It is assumed that a global placement solution has good distribution of cells. Legalization should preserve the placement quality by doing minimal perturbation to the placement, i.e., by minimizing the total cell displacements.

In practice, standard cells have power rail on the top or bottom side, while ground rail is located on the other side. Power and ground rails are routed horizontally between the rows. Therefore, every cell must be vertically aligned to rows such that it can be properly powered. Furthermore, standard cells are designed such that the pins are aligned to the routing grids. Therefore, standard cells must also be horizontally aligned to predefined uniform placement sites, such that pins are aligned.

For each row, power is either at the top or bottom while the ground is at the other end. Therefore, a single-row standard cell should be placed with a correct vertical flipping such that the power/ground position matches with that of the row. However, for multi-row cells with even number of row height, both sides of the cells are either power or ground, as shown in figure 1(a). Therefore, cells with even row height can only be placed on every other rows with proper power line alignment. On the other hand, multi-row cells with an odd number of row height can be placed on every row provided that the cell orientation is correctly flipped, as shown in figure 1(b).

For each cell c_i, the width and height are w_i^c and h_i^c respectively, and the lower-left corner position is at (x_i^c, y_i^c). For each row r_i, the width and height are w_i^r and h_i^r, and the lower-left corner position is at (x_i^r, y_i^r). We are also given a constant site width $Site_w$ and site height $Site_h$. All cell widths and row widths are multiples of $Site_w$, while all cell heights are multiples of $Site_h$, and all row heights *equal* $Site_h$.

We can now formally define the problem as follow:

In a multi-row height legalization problem, we are given a placement solution $P = \{(x'_1, y'_1), (x'_2, y'_2), \cdots, (x'_n, y'_n)\}$ of a set of n movable cells $C = \{c_1, c_2, \cdots, c_n\}$ on m rows $R = \{r_1, r_2, \cdots, r_m\}$ specified by a floorplan, the netlist and the cell library, the objective is to assign each cell c_i with a position (x_i, y_i), such that the total cell displacement is minimized:

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

while the following constraints are satisfied:

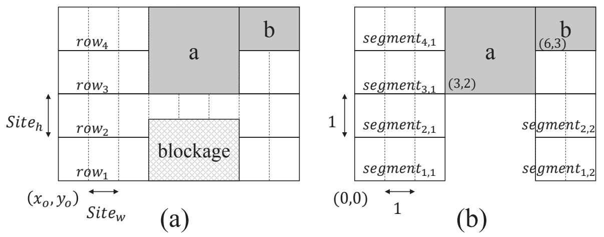

Figure 2: Units of Measurement

1. Cells are overlap-free , i.e.,

$$\forall c_i, c_j \in C,$$
$$(x_i^c + w_i^c \le x_j^c) \vee (y_i^c + h_i^c \le y_j^c) \vee$$
$$(x_j^c + w_j^c \le x_i^c) \vee (y_j^c + h_j^c \le y_i^c)$$

2. Cells are aligned to placement sites on rows, i.e.,

$$\forall c_i \in C, \exists r_j \in R,$$
$$y_i^c = y_j^r \wedge x_i^c = x_j^r + \alpha_i \cdot Site_w \text{ where } \alpha_i \in \{0, 1, 2, \cdots\}$$

3. Cells are completely contained inside one or multiple rows, i.e.,

$$\forall c_i \in C, \forall h \in \{0, 1, \cdots, \frac{h_i^c}{Site_h} - 1\}, \exists r_j \in R,$$
$$y_i^c + h \cdot Site_h = y_j^r \wedge x_i^c \ge x_j^r \wedge x_i^c + w_i^c \le x_j^r + w_j^r$$

4. Cells with height of even multiples of site height must be placed in alternate rows with matching power rail alignment, i.e.,

$$\forall c_i \in C \text{ s.t. } h_i^c = 2m \times Site_h \text{ for some } m = 0, 1, \cdots,$$
$$y_i^c \in \begin{cases} \{y_1^r, y_3^r, y_5^r, \cdots\} & \textbf{if} \text{ the first row } r_1 \text{ matches power rail of } c_i \\ \{y_2^r, y_4^r, y_6^r, \cdots\} & \textbf{otherwise} \end{cases}$$

2.1 Preliminaries and Terminology

2.1.1 *Units of Measurement*

For simplicity, all locations and dimensions used in our algorithm are measured in unit of placement site. For horizontal dimension, the unit is the number of placement site width, $Site_w$, while vertical dimension uses the unit of the number of placement site height, $Site_h$. Cells' and rows' positions refer to the coordinate of their lower-left corners. Figure 2(a) shows the measurement in actual scale, while Figure 2(b) shows the measurement in our implementation. However, when we measure cell displacement or wirelength, we will use the actual unit in micron.

2.1.2 *Segments*

We differentiate between placement *rows* and *segments*. Row is defined by the floorplan and placement sites on a row may be blocked by macros or placement blockages. A *segment* is defined as a continuous sequence of non-blocked placement sites. Figure 2(a) shows the rows defined by the floorplan, while Figure 2(b) shows the corresponding segments defined in our algorithm.

For each segment s, a list of cell references $\{c_1^s, c_2^s, \cdots, c_m^s\}$ is maintained. Cells in the list are ordered by their x-coordinates. An unplaced cell does not appear in any segment cell list, while a placed cell appears in h segment cell lists, where h equal to the height of the cell. For example, in Figure 2(b), *segment*$_{3,1}$ has a cell list of $\{a\}$, and *segment*$_{4,1}$ has a cell list of $\{a, b\}$.

2.1.3 *Local Region*

Given a rectangular window W defined by its lower-left corner's coordinates, width and height as (x_W, y_W, w_W, h_W), we can extract a localized placement problem within W. Cells not *completely* inside W are initialized as *non-local* cells. For example, a floorplan and a window in red dashed-line box are shown

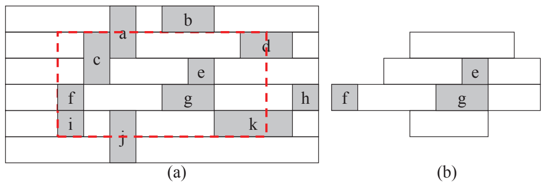

Figure 3: Extraction of local region

in Figure 3(a). Cells a, d, j, k are initialized as non-local cells. The placement rows in W are divided by non-local cells into continuous segments. For each row in W, we will pick only one continuous segment as *local segment*. If there are more than one in a row, the one closest to the window center will be selected as a *local segment*. A cell inside W is a local cell if and only if it is contained completely within the local segments. According to this definition, there may be cells which are completely inside W but are non-local, e.g., cell i and c in Figure 3(a). In this example, only cells e, f and g are *local cells* and *local region* (union of all *local segments*) is shown in Figure 3(b).

In the example shown in Figure 3, although the placement site at the upper-left corner of the local region is not occupied, it is not a local segment because the other one on the same row is closer to the window center. Beside, although cell i is contained completely in the window W, it is not contained in any local segment, it is thus not included as local cell. Note that as long as the window size is reasonable, the local region is large enough to provide solution with high quality. Due to page limit, the algorithm of local region extraction is not described.

3. ALGORITHM OVERVIEW

To solve the legalization problem defined in the previous section, we propose a method as shown in Algorithm 1.

Algorithm 1 Legalization Algorithm

```
1: Unplaced = $C$
2: for each $c_i \in$ Unplaced do
3:     result ← $MLL(c_i, (x'_i, y'_i))$
4:     if result = $Success$ then
5:         Remove $c_i$ from Unplaced
6:     end if
7: end for
8: $k \leftarrow 1$
9: while Unplaced is not empty do
10:    for each $c_i \in$ Unplaced do
11:        result ← $MLL(c_i, (x'_i + Rand_x(k), y'_i + Rand_y(k))$
12:        if result = $Success$ then
13:            Remove $c_i$ from Unplaced
14:        end if
15:    end for
16:    $k \leftarrow k + 1$
17: end while
```

At the beginning, all cells are set as unplaced (line 1). For each cell c_i in an arbitrary order, we will place the cell at the nearest site-aligned and power-rail matching position from the input position (x'_i, y'_i). If the placement does not cause any overlap with other cells, we will place it directly at the position. Otherwise, we will trigger the Multi-row Local Legalization algorithm (MLL) as follow:

The local legalization algorithm takes a target cell c_t and its target position (x^c_t, y^c_t) as parameters. A *Local Region* W is defined with the window's lower-left corner at $(x^c_t - R_x, y^c_t - R_y)$, its width of $(2R_x + w^c_t)$ and its height of $(2R_y + h^c_t)$, where R_x and R_y are variables determining the window size ($R_x = 30, R_y = 5$ in our implementation). The algorithm seeks for a legal placement for all the *local cells* inside W plus the target cell c_t, while the total displacement of every local cell from its current position and the displacement of c_t from its target position (x^c_t, y^c_t) is minimized. When a legal solution is not found with the algorithm, it will abort the target position and return without changing the placement. Otherwise, it will place the cell and legalize the placement.

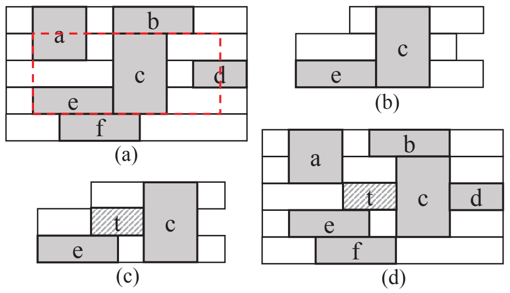

Figure 4: Overview of theMLL algorithm

In the first iteration (line 2–7), every cell c_i is tried to be placed at the input position (x'_i, y'_i). MLL is called when legalization is required (line 3). After the first iteration, when there is still any unplaced cell, we will repeat the placement of each unplaced cell $cell_i$ with another position $(x'_i + Rand_x(k), y'_i + Rand_y(k))$ (line 11), where k is the current iteration number, $Rand_x(k)$ is a random integer in the range of $[-R_x \cdot (k-1), R_x \cdot (k-1)]$ and $Rand_y(k)$ is another random integer in the range of $[-R_y \cdot (k-1), R_y \cdot (k-1)]$. The placement iteration is repeated until all the cells are placed (line 9–17).

As the core of our legalization is the MLL algorithm. In the remaining of this paper, we will focus on the description of the algorithm.

4. MULTI-ROW LOCAL LEGALIZATION

The Multi-row Local Legalization (MLL) algorithm works on a *local region* W with legally placed *local cells* C_W and an unplaced *target cell* c_t. The algorithm finds a legal placement solution while minimizing the total displacement of the local cells and the target cell. The overview of the MLL algorithm is illustrated in Figure 4. Figure 4(a) shows the original placement and the red dashed-line box shows the window from which we can extract the local region. Figure 4(b) shows the extracted local region. Our MLL algorithm will then give a legal solution by inserting the target cell and legalize it within the local region, as shown in Figure 4(c). The resulting placement is then mapped back to the original floorplan and the result is shown in Figure 4(d).

Due to the power line alignment constraint, shifting a placed cell vertically to a compatible row will bring a relatively large displacement or will involve cell flipping, especially for even row-height cells. Therefore, once a cell is placed in a row with matching power line alignment, the y-coordinate of the cell is fixed, while it is still free to shift along the x-direction for placing other cells. Furthermore, to maintain the global placement solution as much as possible, the algorithm does not change the relative cell order in each segment.

With fixed local cells' row and their relative orders in a row, inserting a target cell with height h^c_t implies that we need to place the target cell in some gaps between the cells in h^c_t consecutive segments. For example, given a local region and a set of local cells $C_W = \{a, b, c, d, e\}$ as in Figure 5(a), and assume that we want to insert a 3×2 cell to the local region at a position as shown by the red dash-lined box.

To insert this triple-row cell, we need to find out three gaps between cells from three vertically consecutive segments. We represent gaps as (r, i, j), where r is the segment which the gap is lying on, i is the cell on its left (or L when its left side is the segment boundary), and j is the cell on its right (or R when its right side is the segment boundary). For example, in Figure 5(b), we choose to insert the target cell t at three gaps: $\{(2, c, d), (3, c, R), (4, b, R)\}$. The resulting legal placement for these gaps selection with minimal displacement is shown in Figure 5(b). The resulting total minimal displacement is the sum of the cells' displacement from that in (a) plus the distance of the cell t from the its target position, which is $8 \cdot Site_w$.

Another option is to insert the target cell in another three gaps: $\{(1, e, c), (2, L, c), (3, a, c)\}$ as shown in Figure 5(c), the resulting displacement is $3 \cdot Site_w + Site_h$.

We refer a combination of gaps for inserting the target cell as *insertion point*. The previous two examples show two feasible

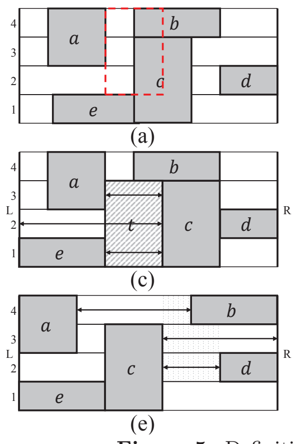
Figure 5: Definition of insertion point

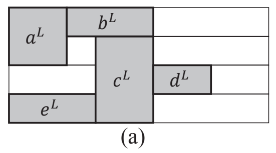
Figure 6: Leftmost/Rightmost placement

insertion points. Figure 5(d) shows the optimal insertion point $\{(2, L, c), (3, a, c), (4, a, b)\}$ of this example, which results in the minimal total dispalcement of $2 \cdot Site_w$. However, not all insertion points can be used, Figure 5(e) and (f) show two examples of infeasible insertion points. There are a finite set of insertion points in a given local region. Our MLL algorithm composes of three main ideas: (1) A fast scanline algorithm of enumerating all *feasible* insertion points (Section 5.1). (2) Given any valid insertion point, a fast evaluation method is proposed to determine the exact target cell position that can minimize the total displacement (Section 5.2). (3) Given any valid insertion point and the exact target cell position, we can produce a legal placement with minimal displacement (Section 5.3).

5. IMPLEMENTATION DETAILS

5.1 Insertion Point Enumeration

5.1.1 *Insertion Interval*

In Section 4, we defined *insertion point* for a target cell with height h as a combination of gaps between cells from h consecutive segments. In this section, we define a gap formally as *insertion interval*. The definition of insertion interval $I^r_{i,j}$ is extended from the gap's definition in the previous section with additional data as $I^r_{i,j} = (r, i, j, x_i, x_j)$, where r is the segment which the gap is lying on, i/j is the cell on its left/right or L/R when its left/right side is the segment boundary, x_i/x_j is the leftmost/rightmost possible x-coordinate for the target cell in the gap on segment r.

To find the values of x_i and x_j, we first find two special placements of the local cells for a local region referred as $leftmost$ and $rightmost$ placement. $Leftmost/Rightmost$ placement is a legal placement solution of the local region with the local cells placed at its smallest/largest possible x-coordinate with the current cell order on each segment. Figure 6 gives an example of the leftmost and rightmost placement of a given placement of a local region. We refer a cell c_i's x-coordinates in its leftmost and rightmost placement as x^L_i and x^R_i respectively. With the leftmost and rightmost placement solutions, we can find each insertion interval as:

$$\begin{cases} (r, i, j, x^L_i + w^c_i, x^R_j - w^c_t) & (a) \text{ or} \\ (r, L, j, x^s_r, x^R_j - w^c_t) & (b) \text{ or} \\ (r, i, R, x^L_i + w^c_i, x^s_r + w^s_r - w^c_t) & (c) \end{cases}$$

where w^c_i is the width of cell i, w^c_t is the width of the target cell, x^s_r is the x-coordinate of the segment r, w^s_r is the width of the segment r, (a) is the case when the gap is between cell i and j; (b) is the case when the gap is between the segment boundary and cell j; (c) is the case when the gap is between cell i and the segment boundary. The three cases are illustrated in Figure 7(a), (b), and (c) respectively.

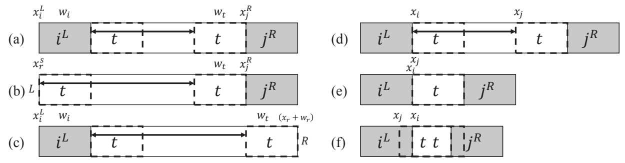
Figure 7: Definition of insertion interval

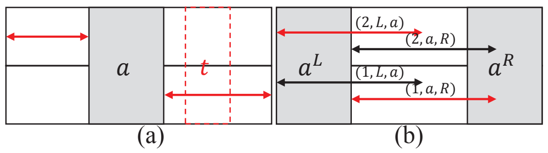
Figure 8: Example of invalid insertion point

The *length* of an interval is calculated as $x_j - x_i$. There are three possible cases of interval length as shown in Figure 7. Figure 7(d) shows an interval with positive length, which means that the target cell has flexibility of choosing its position when it is inserted in the gap. Figure 7(e) shows an interval with length equals 0, which means that the target cell's position is determined by the interval's endpoint. Figure 7(f) shows the case with negative *length* value, which means that there is no legal placement with the target cell in the gap. In such case, it is safe to discard the insertion interval.

5.1.2 *Valid Insertion Point*

After setting up all possible insertion intervals, the objective of *Insertion Point Enumeration* is to find a set of h_t intervals from h_t consecutive segments, which we can find a common x-coordinate x_t for all the intervals. In other words, we find a set of intervals with a *common cutline*. When all local cells are with single row height, every interval set with a common cutline form a valid insertion point. However, when there exists multi-row height local cells, intervals on the left side of a multi-row cell cannot form insertion point with another interval on the right side of the multi-row cell. Figure 8 shows such situation. Figure 8(a) shows two segments with a multi-row cell a and a target cell t to be inserted. In Figure 8(b), the gaps $(1, a, R)$ and $(2, L, a)$ are on different sides of the multi-row cell a and thus they do not form a valid insertion point although they have a common cutline.

5.1.3 *Enumeration of all Valid Insertion Points*

To enumerate all valid insertion points, a naive implementation is to find all permutation of the intervals and screen out the invalid sets. However, it is computationally impractical. We propose the following algorithm.

A set of queues Q^r_s are defined for storing intervals, where r and s are labels of the pair of segments that can possibly form insertion point. Since the target cell height is h_t, the label r and s of a queue Q^r_s should be such that (1) $1 \le r \ne s \le h_W$, and (2) $|r - s| \le h_t - 1$

For the example in Figure 5, we have the following queues defined:

$$Q^1_2, Q^1_3, Q^2_1, Q^2_3, Q^2_4, Q^3_1, Q^3_2, Q^3_4, Q^4_2, Q^4_3 \qquad (1)$$

We will enumerate all the intervals $I^r_{i,j} = (r, i, j, x_i, x_j)$ inside the local region. Note that there will be at most $(|C_W|+1) \times h_W$ of such intervals where $|C_W|$ is the number of local cells in window W and h_W is the height of W. Each interval will be associated with two endpoints x_i and x_j. We are going to sort all these endpoints in a non-decreasing order and process the intervals according to this sorted list, enqueuing them into the queues defined above and forming insertion point simultaneously. Suppose now we are working on a point x_i in this list which is the left endpoint of an interval $I^a_{i,j}$. First, we will

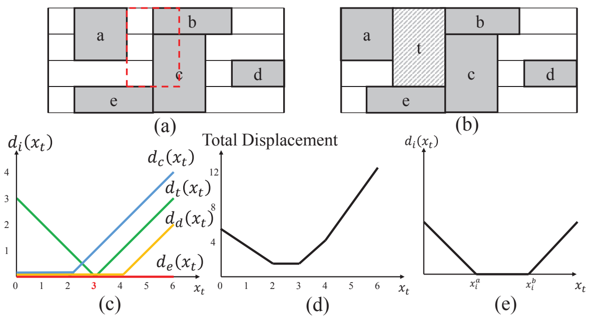

Figure 9: Evaluation of a insertion point

generate the insertion points involving $I_{i,j}^a$ as:

$$\bigcup_{t=\max(1,a-h_t+1)}^{\min(h_W-h_t+1,a)} \{I_{i,j}^a\} \times \prod_{s=t,s\neq a}^{t+h_t-1} Q_s^a \quad (2)$$

For example, assume we have queues as in (1), when the target cell height is 3 and the current interval is $I_{i,j}^2$ from segment 2, the set of insertion points to be generated when we process up to the point x_i in the sorted list is:

$$\{Q_1^2 \times \{I_{i,j}^2\} \times Q_3^2\} \cup \{\{I_{i,j}^2\} \times Q_3^2 \times Q_4^2\}$$

However, due to the constraint that intervals on one side of a multi-row height cell cannot form insertion point with the intervals on the other side, when the left cell i of an interval $I_{i,j}^a$ is a multi-row cell occupying a set of segment $S = \{s_1, s_2, \cdots s_{h_i}\}$, we simply clear the queue Q_s^a with $s \in S$. Furthermore, to consider power line alignment, we can simply discard the insertion point which does not satisfy the constraint. The checking is simple and trivial and we will not discuss further on this due to space limit.

After finishing the enumeration of insertion points involving the current interval $I_{i,j}^a$, the interval is pushed to all queues Q_s^r with $s = a$. On the other hand, when the point being processed is a right endpoint of an interval $I_{i,j}^a$, we pop the interval $I_{i,j}^a$ from all queues Q_s^r with $s = a$. After scanning all the interval endpoints, we have enumerated all valid insertion points. In the algorithm, all insertion points are enumerated without duplication. Therefore the time complexity is $O(((|C_W|+1)\times h_W)^h) = O(|C_W|^h)$, where h is the target cell height, which is usually a very small integer.

5.2 Insertion Point Evaluation

Given a valid insertion point, the relative positions of the cells in the related segments are known. However, the target cell have flexibility for its exact placement position and each position may result in different total cell displacement. For every local cell, the displacement of the cell is either zero or linear to the target cell's position with a slope of 1 if the cell is on the right of the target cell, or -1 if the cell is on the left of the target cell, depending on the position of the target cell. Figure 9(a) shows an example of the original placement and the red dashed-line box showing the target position. Figure 9(b) shows the resulting legal placement with a given insertion point. Figure 9(c) shows the curves of displacement of cell c, d, e against the target cell t's position. Figure 9(d) shows the total displacement by summing up every displacement curve. For each cell c_i on the left of the target cell, we call the smallest x-position of the target cell that does not cause any displacement to cell c_i the *critical position* for cell c_i and denote it as x_i^a. Similarly, for cell c_j on the right of the target cell, we call the largest x-position of the target cell that does not cause any displacement to cell c_j as the *critical position* for cell c_j and denote it as x_j^b. We can generalize all curves as the following function:

$$d_i(x_t) = \begin{cases} x_i^a - x_t^c, & x_t^c < x_i^a \\ 0, & x_i^a \le x_t^c \le x_i^b \\ x_t^c - x_i^b, & x_t^c > x_i^b \end{cases} \quad (3)$$

where for the cells on the left of the target cell, $x_i^b = \infty$; for the cells on the right of the target cell, $x_i^a = -\infty$; and for the target cell, $x_i^a = x_i^b = x_t'$. The displacement curve of each cell has the shape shown in Figure 9(e).

It is not hard to show that the optimal position to place the target cell to minimize the total displacement is the median of the set of critical positions $\{x_i^a, x_i^b | c_i \in C_W \cup \{c_t\}\}$, where C_W is the set of local cells. With the optimal position of the target cell, we can calculate the resulting total displacement as the sum of equation (3) for each local cell and the target cell. It will be used as the cost of the insertion point. The remaining problem is how to find the values of the critical positions. The values of all critical positions can be found in $O(|C_W|)$ time. Due to page limit, the method of finding all critical positions is not discussed in this paper.

For efficiency, we used an approximated calculation of the optimal position for the target cell considering its neighboring cells only. For a multi-row cell with height of h_t, there are at most $2\times h_t$ neighbors. The critical positions for the neighboring cells can be found easily. For the left neighboring cell c_i, its critical position x_i^a due to the target cell is $x_i^c + w_i$. For the right neighboring cell c_j, its critical position is $x_j^c - w_t$. The calculation can be done in $O(h_t)$, where h_t is so small that it can be assumed to be a constant.

5.3 Legal Placement Realization

With the methods proposed in the last section, the insertion point and the target cell's position with the minimal displacement are found. In this section, we propose an efficient algorithm to realize such legal placement with minimal total displacement as follow:

Algorithm 2 Legal Placement Realization

```
1:  cell $c_t$ is placed at $(x_t, y_t)$
2:  $Q_L = \{c_t\}$
3:  while $Q_L$ is not empty do
4:      $c_i = Q_L.pop()$
5:      for each left neighboring cell $c_L$ of $c_i$ do
6:          if $c_L$ overlaps $c_i$ then
7:              $x_L \leftarrow x_i - w_L$
8:              $Q_L.push(x_L)$
9:          end if
10:     end for
11: end while
12: $Q_R = \{c_t\}$
13: while $Q_R$ is not empty do
14:     $c_i = Q_R.pop()$
15:     for each right neighboring cell $c_R$ of $c_i$ do
16:         if $c_R$ overlaps $c_i$ then
17:             $x_R \leftarrow x_i + w_i$
18:             $Q_R.push(x_R)$
19:         end if
20:     end for
21: end while
```

At the beginning, the target cell c_t is placed at the position (x_t, y_t) which we have found in Section 5.2 (line 1). A queue Q_L is initialized with c_t (line 2). We repeat the following procedure until Q_L becomes empty. Cell c_i is popped from Q_L (line 4). It is checked against every c_i's left neighboring cells c_L to see if it overlaps with c_i (line 5–6). If it does, we shift the cell c_L to the left with minimal distance such that it does not overlap with c_i (line 7). After the shifting, we push c_L into Q_R (line 8). The legalization on the left side is finished when Q_L is empty. A similar procedure is applied on the right side of the target cell (line 12–21). In the algorithm, every cell is processed at most once, therefore the time complexity is $O(|C_W|)$, where C_W is the set of local cells in the window W.

6. EXPERIMENTAL RESULTS

To validate our proposed method, the algorithm is implemented in C++. The experiments were performed on a 64-bit Linux machine with Intel Xeon 3.4GHz CPU and 32GB memory, using the benchmarks provided by ISPD2015 Detailed

Table 1: Experimental Resuts

					Power Line Aligned						Power Line Not Aligned					
				GP	Disp. (sites)		ΔHPWL		Runtime (s)		Disp. (sites)		ΔHPWL		Runtime (s)	
Benchmarks	**#S. Cell**	**#D. Cell**	**Density**	**HPWL(m)**	**ILP**	**Ours**	**ILP**	**Ours**	**ILP**	**Ours**	**ILP**	**Ours**	**ILP**	**Ours**	**ILP**	**Ours**
des_perf_1	103842	8802	0.91	1.43	2.13	3.32	2.61%	2.85%	4098.7	7.2	1.79	1.84	2.59%	1.30%	4478.9	6.5
des_perf_a	99775	8513	0.43	2.57	0.66	0.96	0.11%	0.28%	193.8	2.6	0.26	0.31	0.03%	0.04%	151.4	2.4
des_perf_b	103842	8802	0.50	2.13	0.62	0.85	0.12%	0.31%	250.8	2.4	0.24	0.32	0.02%	0.03%	194.7	2.2
edit_dist_a	121913	5500	0.46	5.25	0.45	0.47	0.09%	0.10%	206.0	1.9	0.22	0.24	0.03%	0.03%	173.0	1.8
fft_1	30297	1984	0.84	0.46	1.58	1.81	2.25%	1.66%	776.8	1.1	1.26	1.13	1.77%	0.66%	818.1	0.9
fft_2	30297	1984	0.50	0.46	0.66	0.86	0.55%	0.87%	72.7	0.4	0.32	0.33	0.17%	0.11%	59.3	0.4
fft_a	28718	1907	0.25	0.75	0.60	0.64	0.32%	0.33%	38.2	0.3	0.32	0.35	0.12%	0.11%	30.7	0.2
fft_b	28718	1907	0.28	0.95	0.73	0.80	0.32%	0.33%	61.9	0.4	0.42	0.51	0.13%	0.13%	52.3	0.4
matrix_mult_1	152427	2898	0.80	2.39	0.49	0.53	0.36%	0.28%	967.4	3.9	0.37	0.40	0.23%	0.13%	709.4	3.8
matrix_mult_2	152427	2898	0.79	2.59	0.45	0.49	0.30%	0.22%	825.0	4.0	0.34	0.37	0.18%	0.09%	640.5	4.1
matrix_mult_a	146837	2813	0.42	3.77	0.27	0.33	0.09%	0.14%	150.7	1.6	0.18	0.19	0.05%	0.05%	126.1	1.5
matrix_mult_b	143695	2740	0.31	3.43	0.25	0.30	0.09%	0.13%	127.8	1.3	0.16	0.17	0.05%	0.05%	108.4	1.2
matrix_mult_c	143695	2740	0.31	3.29	0.27	0.29	0.11%	0.11%	139.0	1.4	0.18	0.20	0.06%	0.05%	122.8	1.3
pci_bridge32_a	26268	3249	0.38	0.46	0.88	0.95	0.52%	0.58%	49.4	0.3	0.30	0.32	0.11%	0.11%	35.7	0.3
pci_bridge32_b	25734	3180	0.14	0.98	0.95	0.96	0.12%	0.13%	15.3	0.2	0.24	0.25	0.03%	0.03%	9.5	0.1
superblue11_a	861314	64302	0.43	42.94	1.85	1.94	0.15%	0.15%	3073.6	23.4	1.49	1.54	0.12%	0.12%	2673.5	21.7
superblue12	1172586	114362	0.45	39.23	1.45	1.63	0.18%	0.22%	5079.0	106.5	1.02	1.07	0.12%	0.12%	4462.4	95.9
superblue14	564769	47474	0.56	27.98	2.56	2.62	0.22%	0.22%	3360.6	17.1	2.18	2.20	0.20%	0.19%	3141.1	15.8
superblue16_a	625419	55031	0.48	31.35	1.61	1.73	0.10%	0.12%	2470.7	21.7	1.20	1.26	0.08%	0.08%	2221.0	19.5
superblue19	478109	27988	0.52	20.76	1.52	1.60	0.14%	0.14%	1848.8	10.9	1.24	1.28	0.11%	0.11%	1717.4	10.1
				Avg.	1.00	1.16	0.44%	0.46%	1190.3	10.4	0.69	0.71	0.31%	0.18%	1096.3	9.5
				N. Avg.	0.87	1.00	0.95	1.00	185.0	1.0	0.93	1.00	1.74	1.00	186.5	1.0

Routing-Driven Placement Contest [13]. Since the cell library provided by the contest is single-row height standard cell library, we modified the benchmarks by doubling all sequential cells' height and halving the cells' width. This reflects the fact that sequential cells are usually the cells with complicated layout and being benefited with multi-row design. However, some of the benchmarks provided by the contest do not provide enough information to identify the sequential cells. In this case, we randomly selected 10% of the cells and converted them to double height and half width. Such modification maintains the total cell area and it ensures that the floorplan can contain all the cells. Table 1 shows the information of the benchmarks. In the table, *#S. Cell* denotes total number of single-row height cells, *#D. Cell* denotes total number of double-row height cells, *Density* is the design density, *GP HPWL* is the wirelength in unit of meter after global placement obtained from the global placer provided by one of the top-3 winners of the contest [13].

To evaluate our legalization quality, we also formulated the local legalization problem as an ILP problem according to the objective function and constraints described in Section 2. In the ILP experiment, the MLL algorithm is replaced by a procedure of constructing and solving the ILP problem with an open-source ILP solver, *lpsolve*. We formulated the local legalization to be exactly the same problem solved by the MLL algorithm, which gives the optimal solution to our problem. Due to page limit, the detailed ILP formulation is not included in this paper. It is worth noting that although the ILP approach gives an optimal solution to the local legalization problem, the optimal solutions to the local sub-problems do not necessarily lead to the final global optimal solution. It is the reason why our approach performs better than the ILP approach for the *fft_1* benchmark.

Table 1 shows the result of the legalization in three aspects: (1) average cell displacement measured in number of placement site width, (2) HPWL wirelength change comparing with the input global placement, (3) algorithm runtime in second. Average and normalized average values are shown in the last two rows respectively. Consider the complexity in legalizing a placement with multiple-row height cells, our algorithm can complete the legalization process on an unaligned and overlapping global placement in a very short time. For the small benchmarks, it took a few seconds for most of the benchmarks, while the largest benchmark with over million of cells took less than 2 minutes. On the other hand, the ILP formulation of the problem can be solved with 13% better in displacement on average, but the runtime is 185× higher than our approach. Furthermore, from the resulting total wirelength change, we can see that the impact of the legalization on the wirelength is very small (<0.5% average). The quality loss in terms of displacement in our algorithm comparing to the ILP approach is marginal. It shows that the approximated evaluation of insertion points is accurate enough to choose the near-optimal place for inserting the target cell.

Furthermore, we did another set of experiments on relaxing the constraint of power line alignment, by assuming every cell can be placed on any row. The experiment shows the impact on cell displacement and the total wirelength. Since double-row height cells must be placed in alternate rows, displacement caused by such requirement is significant. By relaxing the power line alignment constraint, the average cell displacement with ILP and our algorithm are 38% and 42% lower, and the wirelength change are 45% and 58% better with ILP and our approach respectively.

7. CONCLUSION

In this paper, we explore a new area of legalization involving multi-row height standard cells. With the ILP formulation, we can obtain high quality solutions, while with our novel idea of interval enumeration and evaluation to find a valid legalization solution, we can solve the problem much more efficiently.

8. 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 *Proc. SPIE*, 2008

[2] T. R. Gheewala, M. J. Colwell, H. H. Yang, D. G. Breid, "Dual-height Cell with Variable Width Power Rail Architecture," in *US Patent US6838713*, 2005

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

[4] J. Cong, X. Min, "A Robust Mixed-Size Legalization and Detailed Placement Algorithm,", in *IEEE TCAD:27(8)*, 2008

[5] A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C.-K. Koh, P. H. Madden, "Recursive Bisection Based Mixed Block Placement,", in *Proc. ISPD*, 2004

[6] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze and M. Xie, "mPL6: Enhanced Multilevel Mixed-Size Placement," in *Proc. ISPD*, 2006

[7] D. Hill, "Method and System for High Speed Detailed Placement of Cells Within Integrated Circuit Designs," in *US Patent US6370673*, 2002

[8] P. Min, N. Viswanathan, C. Chu, "An Efficient and Effective Detailed Placement Algorithm," in *Proc. ICCAD*, 2005

[9] A. B. Kahng, P. Tucker and A. Zelikovsky, "Optimization of Linear Placements for Wirelength Minimization with Free Sites," in *Proc. ASPDAC*, 1999

[10] G. Wu and C. Chu, "Detailed Placement Algorithm for VLSI Design with Double-Row Height Standard Cells," in *IEEE TCAD:35*, 2016

[11] W.-K. Chow, J. Kuang, X. He, W. Cai and E. F. Y. Young, "Cell Density-driven Detailed Placement with Displacement Constraint," in *Proc. ISPD*, 2014

[12] S. Popovych, H.-H. Lai, C.-M. Wang, Y.-L. Li, W.-H. Liu, T.-C. Wang, "Density-aware detailed placement with instant legalization," in *Proc. DAC*, 2014

[13] 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 *Proc. ISPD*, 2015