國立清華大學
NATIONAL TSING HUA UNIVERSITY

Department of Computer Science and Information Engineering

CS6135 VLSI Physical Design Automation
Homework 5: Placement Legalization

黃偉祥 X1136010

Lecturer: Ting-Chi Wang

A report of CS6135 course Homework 5
Institute of Computer Science and Information Engineering

June 5, 2025

Declaration

I, Wei-Xiang Wong, of the Department of Computer Science and Information Engineering, National Tsing Hua University, confirm that this is my own work, figures, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalized accordingly.

I give my consent to having a copy of my report shared with future students as an example.

Wei-Xiang Wong June 5, 2025

Contents

1 About

1.1 Introduction

Implement an existing algorithm, published in the ISPD-08 paper entitled "Abacus: fast legalization of standard cell circuits with minimal movement" Spindler et al. (2008), to legalize a given global placement result with minimal total displacement (measured by Euclidean distance).

1.2 Problem Description

1. **Input:**
   - Max displacement constraint for each cell.
   - A set of standard cells (and blockages), where each standard cell (or blockage) has a rectangular shape specified by its width, height, and coordinates. The design is composed of single-row height movable cells, and multiple-row height fixed blockages.
   - Chip specification, such as the coordinates of each row, the row height, the site width, and the number of sites in a row.
2. **Output:**
   - The coordinates of each cell after legalization are finished and the total displacement t_d as well as the max displacement m_d. The coordinates of each cell are specified by its lower-left corner. Please take the ceiling of the total displacement after summing all the cell displacement and take the ceiling of the max displacement.
3. **Objective:**
   Cells are not allowed to be rotated, but they are allowed to be moved. The total displacement of the legalization result should be as small as possible subject to the following constraints.
   (a) **Aligning constraint:** Each cell is not allowed to cross multiple rows, must be placed within the row, and must align its left boundary with the edge of the site.
   (b) **Non-overlapping constraint:** No cell overlaps with other cells or blockages.
   (c) **Max displacement constraint:** The displacement of each cell should be less than or equal to the max displacement threshold.

1.3 Input File

1. The `.txt` **file:**

   The `.txt` file specifies the information of max displacement constraint, cells, blockages, and rows in the placement region. Here is an example:

2. Only the coordinate of the cell is floating type.

```
MaxDisplacementConstraint 12
// MaxDisplacementConstraint threshold of max displacement

NumCells 3
// NumCells the number of cells
Cell c0 1 12 10.0 10.0
// Cell cell name cell width cell height cell x cell y

⋮

NumBlockages 1
// NumBlockages the number of blockages
Blockage b0 4 24 11 10
// Blockage block name block width block height block x block y

⋮

NumRows 2
// NumRows the number of rows
Row r0 1 12 10 10 10
// Row row name site width row height row x row y site number

⋮
```

1.4 Output File

1. The `.out` **file:**

   The `.out` file specifies the total displacement, the max displacement, and the legalization result containing the coordinates of each cell. Here is an example:

```
TotalDisplacement 12
// TotalDisplacement number of total displacement
MaxDisplacement 7
// MaxDisplacement maximum displacement from all the cells

⋮

NumCells 3
// NumCells the number of cells
c0 10 10
// cell name cell x cell y


⋮
```

1.5 Compile & Execute

1. Go into directory **src/**, enter **make** to compile my program and generate the executable file, called **hw5**, which will be in directory **bin/**.
2. Go into directory **src/**, enter **make grade** to grade the program with **HW5_grading**.
3. Go into directory **src/**, enter **make clean** to delete the ***.o** and **executable** file.
4. Use the following command format to run my program:

   $ ./hw5 *.txt *.out

   E.g.:

   $ ./hw5 ../testcase/public1.txt ../testcase/public1.out

2 Result

Result of **HW5_grading.sh**

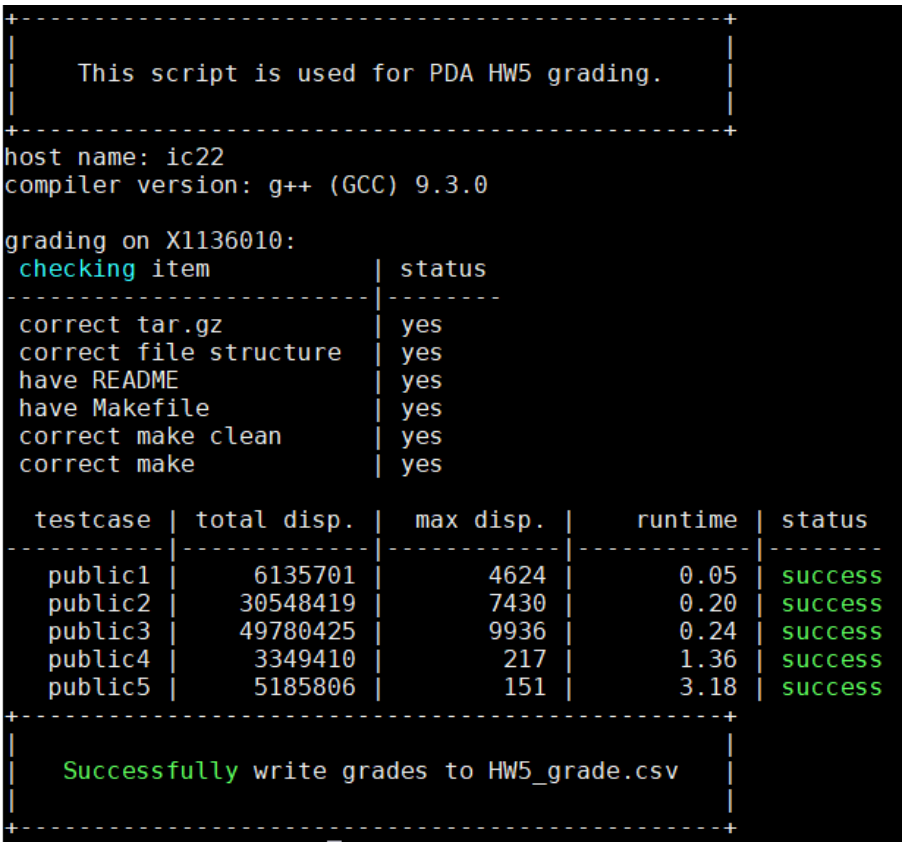

Figure 2.1: Result of HW5_grading

testcase	Total disp.	Max disp.	runtime(s)
public1	6135701	4624	0.05
public2	30548419	7430	0.20
public3	49780425	9936	0.24
public4	3349410	217	1.36
public5	5185806	151	3.18

Table 2.1: Result of HW4_grading.sh

3 Implementation

Algorithm 1 My Abacus Placement Algorithm

```
1: Sort cells by increasing g_x
2: for each cell c in cells do
3:     row_idx ← get_closest_row(c)
4:     best_row ← null
5:     best_subrow ← null
6:     best_cost ← ∞
7:     for round ← 1 to 0 do
8:         down_row_idx ← row_idx
9:         while down_row_idx ≥ 0 and |c.g_y − rows[down_row_idx].y| < best_cost
   do
10:            (sr, cost) ← place_row_trial(rows[down_row_idx], c, max_dis, round)
11:            if cost < best_cost then
12:                best_cost ← cost
13:                best_subrow ← sr
14:                best_row ← rows[down_row_idx]
15:            end if
16:            down_row_idx ← down_row_idx − 1
17:        end while
18:        up_row_idx ← row_idx + 1
19:        while up_row_idx < rows.size() and |c.g_y − rows[up_row_idx].y| <
   best_cost do
20:            (sr, cost) ← place_row_trial(rows[up_row_idx], c, max_dis, round)
21:            if cost < best_cost then
22:                best_cost ← cost
23:                best_subrow ← sr
24:                best_row ← rows[up_row_idx]
25:            end if
26:            up_row_idx ← up_row_idx + 1
27:        end while
28:        if best_subrow ≠ null then
29:            break
30:        end if
31:    end for
32:    place_row_final(best_row, c, best_subrow, max_dis)
33: end for
```

Instead of trying all rows, many of the row will definitely violated the maximum displacement constraints, so we actually can cut off some row. For more advance, instead of checking maximum constraints with y displacement, we can check the current *best* displacement with y displacement between row and cell.

First, we get the closest row with the cell and then searching upward and downward if have a better cost (least cost).

Algorithm 2 place_row_trial(cell c, int max_dis, int $round$)

```
1: subrow ← get_closest_subrow(c)
2: No subrow available return (null, ∞)
3: opt_x ← c.g_x
4: Align opt_x into subrow minimum and maximum
5: last_cluster ← subrow.last_cluster
6: if last_cluster = null or last_cluster.x + last_cluster.width ≤ opt_x then
7:     Place c at (opt_x, y)
8: else
9:     Simulate add c into last_cluster and collapse as below
10:    Initialize stack c_stk
11:    while true do
12:        Push last_cluster to c_stk
13:        Align last_cluster_x into subrow minimum and maximum
14:        pre_cluster ← last_cluster.pre
15:        if pre_cluster ≠ null and overlap then
16:            Simulate merge pre_cluster with last_cluster:
17:            last_cluster ← pre_cluster
18:        else
19:            break
20:        end if
21:    end while
22:    Place c at end of the last_cluster
23:    if round 1 then
24:        while not c_stk.empty() do
25:            cls ← c_stk.top()
26:            for each c_i in cls.cells do
27:                if c_i over maximum displacement constraints then return (null, ∞)
28:                end if
29:            end for
30:            Pop c_stk
31:        end while
32:    end if
33: end if
34: return (sr, c_displacement)
```

We need to check if the others cell will excess the maximum displacement constraints when the new cell place at this subrow.

Algorithm 3 collapse(cluster, sr)

1: $cluster.x \leftarrow cluster.q_c / cluster.weight$
2: **if** $cluster.x < sr.min\_x$ **then**
3: $cluster.x \leftarrow sr.min\_x$
4: **else if** $cluster.x > sr.max\_x - cluster.width$ **then**
5: $cluster.x \leftarrow sr.max\_x - cluster.width$
6: **end if**
7: $pre \leftarrow cluster.pre$
8: **if** $pre \neq$ `null` **and** $pre.x + pre.width > cluster.x$ **then**
9: Append $cluster.cells$ to $pre.cells$
10: $pre.weight \leftarrow pre.weight + cluster.weight$
11: $pre.q_c \leftarrow pre.q_c + cluster.q_c - cluster.weight \cdot pre.width$
12: $pre.width \leftarrow pre.width + cluster.width$
13: **return** collapse(pre, sr)
14: **else**
15: **return** $cluster$
16: **end if**

Algorithm 4 place_row_final(c, sr, max_dis)

1: $sr.free\_sites \leftarrow sr.free\_sites - c.required\_site(s_w)$
2: $opt\_x \leftarrow c.g_x$
3: **if** $opt\_x < sr.min\_x$ **then**
4: $opt\_x \leftarrow sr.min\_x$
5: **else if** $opt\_x > sr.max\_x - c.w$ **then**
6: $opt\_x \leftarrow sr.max\_x - c.w$
7: **end if**
8: $last\_cluster \leftarrow sr.last\_cls$
9: **if** $last\_cluster =$ `null` **or** $last\_cluster.x + last\_cluster.width \leq opt\_x$ **then**
10: $c.y \leftarrow y$
11: $c.x \leftarrow opt\_x$
12: Create new cluster at (opt_x, y)
13: $sr.last\_cls \leftarrow$ new cluster
14: Add c to new cluster
15: **else**
16: Add c to $last\_cluster$ at opt_x
17: $c.x \leftarrow opt\_x$, $c.y \leftarrow y$
18: $sr.last\_cls \leftarrow$ collapse($last\_cluster$, sr)
19: **end if**

4 Tricks

4.1 Handling row to subrow

Figure 4.1 shown the example of original row with the blockage, first I assign all the **site** of the row to 0.

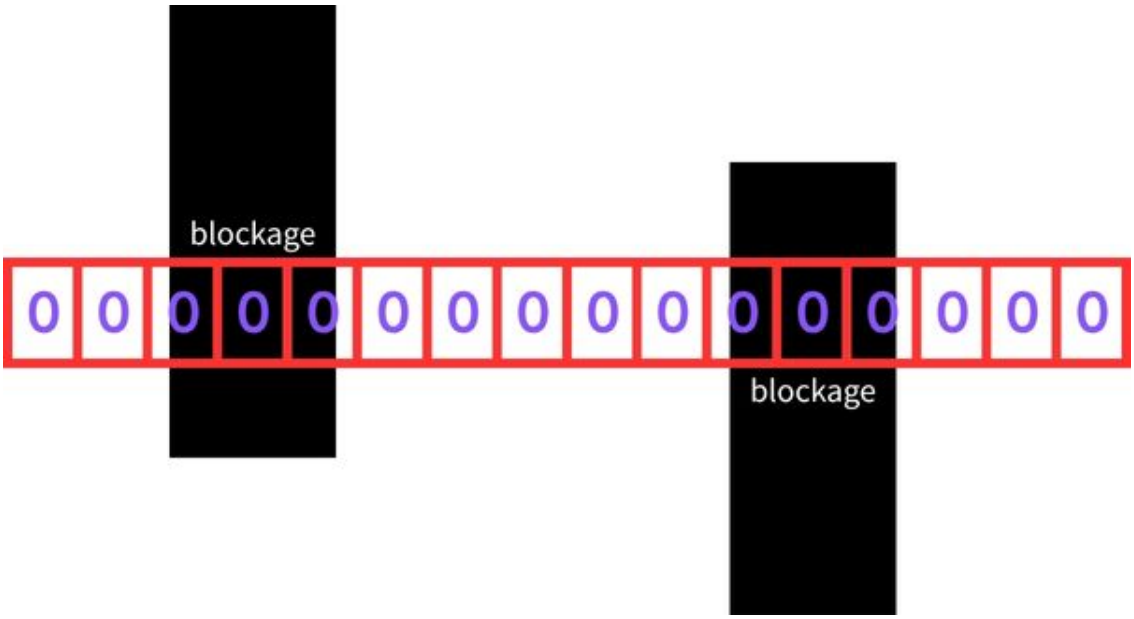

Figure 4.1: Original row and blockage

Figure 4.2 show that assigning the blockage into row, if the blockage overlap with row then the site will assign to 1. **The site will be assign into** 1 **even if the overlap is just a little corner.**

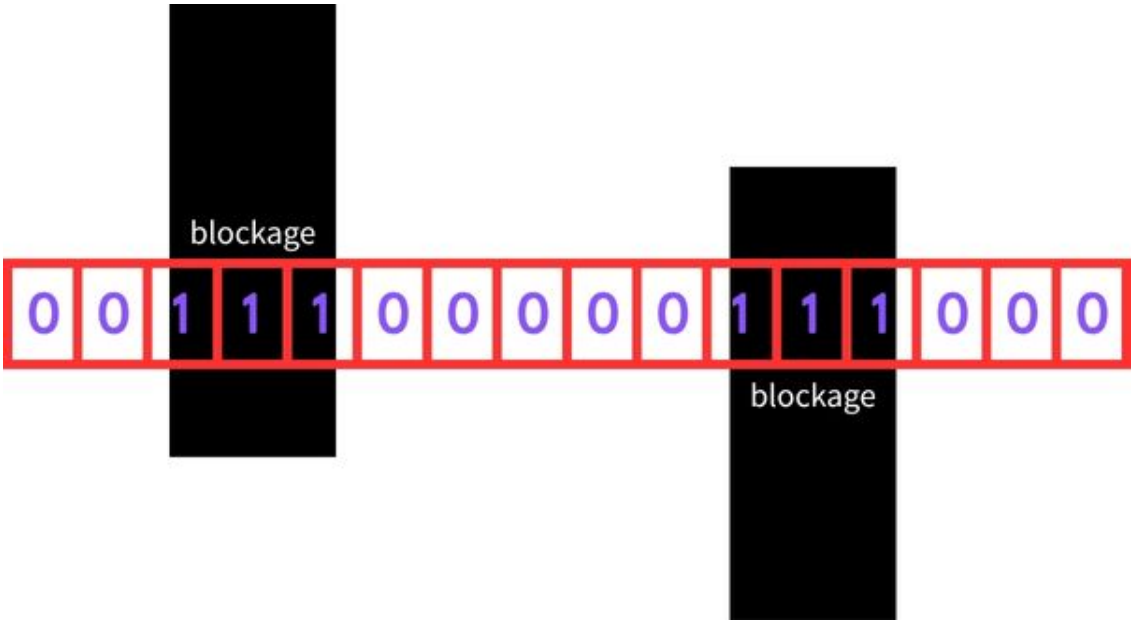

Figure 4.2: Put block into site

After that check each row's site, if have **continuous** with 0 then construct a **subrow**. **Figure 4.3** show the final subrow of example from **Figure 4.1**.

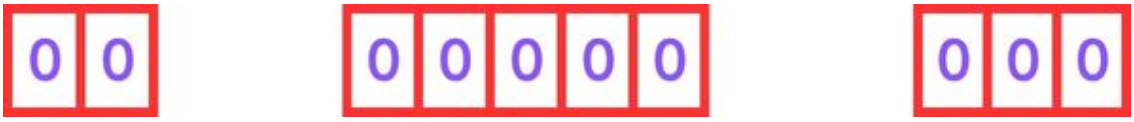

Figure 4.3: Done subrow

4.2 Handling Max displacement constraint

1. Check the influenced cluster's cell, if overlap and after merge these cluster will let any other cell excess maximum displacement constraint then don't place at this subrow. Credited to EricLu1218 (2019)
2. After **Abacus** algorithm, check any cell over the maximum displacement constraint, if it does then try to find some same size cell within the maximum displacement then **swap** this 2 cells. Only swap if **both** cell will **not excess** the maximum displacement constrain.

4.3 Other tricks

1. Start from the **closest** row and subrow, then searching **upward** and **downward** if have a lowest cost.
   - In **original** paper the algorithm check **every** row, but we can actually start from the closest row and check upward and downward. (speed up)
   - In the row, we check only the closest available subrow to speed up.
   - When searching upward or downward, if the y **displacement** between cell and row excess the current **best cost** then we can break the loop. (speed up)

2. Check if the **placed cell** will over maximum displacement constraint.
    - If the placed cell will over maximum displacement constraint after the new cell place into this subrow, then will make the placed cell hard to be legalize within the maximum displacement constraint. (enhance solution and speed up) Credited to EricLu1218 (2019)
3. Swap with other cell after **Abacus**
    - If after the Abacus still have some cells over the maximum displacement constraint, then find the same size cell that within the maximum displacement constraint and swap these 2 cells. Only swap if both will not violate maximum displacement constraint. (enhance solution)
4. **Simulate** place row trial
    - Instead of copy and delete the row to do the trial place row operation, we can use only few variables to simulate the place row operation. (speed up)
    - Instead of add and remove from row, using simulate method with few variables will speed up the program. (speed up)

5 Conclusion

1. **Using shared_ptr:** Instead of create and delete the pointer, we can use smart pointer to have better memory management.
2. **Skip useless operation:** In my first version of program, I do place row trail on every row within the maximum displacement constraint between y displacement of cell and row, but it actually can be speed up with comparing with the current best cost.
3. **Simulate the operation:** In my first version of program, I copy the whole row and delete after place row trial, and this make mine program be really slow and cannot even finish within 1 minutes for public3.txt.
4. **Check at place row trial:** Because if the cell overlap will merge with the previous cluster, so the previous cluster may have larger displacement after merge, so we need to check all the affected cell will over the maximum displacement constraint when placing the new cell in this subrow.

References

EricLu1218 (2019), 'Placement_legalization - physical_design_automation'. Accessed: 2025-06-05.
URL: *https://github.com/EricLu1218/Physical_Design_Automation/tree/main/Placement_Legalization*

Spindler, P., Schlichtmann, U. and Johannes, F. M. (2008), Abacus: fast legalization of standard cell circuits with minimal movement, *in* 'Proceedings of the 2008 International Symposium on Physical Design', ISPD '08, Association for Computing Machinery, New York, NY, USA, p. 47–53.
URL: *https://doi.org/10.1145/1353629.1353640*