Macro Legalization

Team 15 B07901056 張凱鈞
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
b07901056@ntu.edu.tw

***Abstract*—This is the final project report of Physical Design for Nanometer ICs (Spring 2021). Macro legalization is an important step in today's placement flow. In this work, a constraint-graph based macro legalization algorithm flow which combines iterative refinement and simulated annealing is applied to solve the macro legalization problem. Experimental result shows the flow could achieve a balance between cost and runtime.**

***Index Terms*—macro legalization, placement, constraint graph, simulated annealing**

I. Introduction

A. Macro Legalization

As the evolution of IC design industry, the sizes and complexities of circuit designs have grown rapidly. A mixed-size circuit design may contain thousands of macros and millions of standard cells. To achieve good placement result efficiently and robustly, a three-stage placement flow is usually applied [1] [5]. In the first stage (prototyping), analytical placement models are applied to obtain an initial placement, with considerations on wirelength and routability. Note that macro overlaps are permissible in this stage. In the second stage (macro placement), the positions and orientations of macros are determined to remove all the overlaps between macros, while minimizing the macro displacement and maximizing the free space. In the third stage (standard-cell placement), standard cells are placed onto the remaining space. Macro legalization is a step of the second stage, in which all the overlaps are removed.

B. Previous Work

In the previous work regarding macro legalization, a variety of circuit representation methods are applied, including constraint graph, packing trees, etc. Here I focus on constraint-graph based work.

In [2], the transitive closure graph (TCG) [6] is adopted to represent the floorplan. Then, the TCG is optimized through an adaptive simulated annealing method, and the macro positions are determined by linear programming.

In [3], constraint graphs are constructed according to relative positions of each pair of macros. Then, the longest paths in the constraint graphs are refined iteratively until the paths are shorter than chip dimensions. For the determination of macro positions, linear programming is also adopted.

In [4], overlaps of macros are recorded as a special type of edges, which will all be repaired iteratively. Then, the macro locations are determined in a greedy method to minimize the displacement.

C. Overview of the work

In this work, constraint graphs is adopted for circuit design representation. A method combined of iterative refinement and simulated annealing is applied. In the first stage, an acceptable solution is found by iterative refinement efficiently. Then, better solutions are found through a simulated annealing scheme. As a result, the method could achieve a balance between cost and runtime.

The remainder of this report is organized as follows. Section II define the macro legalization problem. Section III describes the proposed algorithm of this work. Section IV shows the experimental result. Section V concludes this work.

II. Problem Formulation

The problem formulation of this work follows the definition in Problem D of ICCAD 2021 CAD Contest [1].

A. Preliminaries

1) **Displacement D** $= \sum_{m_i} |x'_i - x_i| + |y'_i - y_i|$, where (x_i, y_i) and (x'_i, y'_i) are the positions of macro m_i before and after legalization, respectively.
2) **Unavailable areas for standard cells:** Given the "powerplan width constraint", the channels between macros with a width less than the constraint are regarded as unavailable areas for standard cells. The total unavailable area is denoted as A.

B. The Macro Legalization Problem

Given a set of rectangular macros with their initial positions (represented by the coordinates of left-bottom corner) and the boundary of the chip, find positions for the movable macros without violating the following constraints:

1) **All the macros locate inside the chip boundary.**
2) **The locations of fixed macros remain unchanged.**
3) **Minimum channel spacing:** the spaces in x- or y-directions between each pair of neighbor macros should exceed the given distance.
4) **Buffer area reservation:** for each macro, there must exists available spaces for standard cells after extending a given distance from the boundary.

The objective is to minimize the following cost function, where α and β are between 0 and 100000:

$$Cost = \alpha \cdot D + \beta \cdot A^{0.5} \tag{1}$$

III. Algorithm

A. Algorithm Flow

The algorithm flow is shown in Figure 1. After the DEF file (the initial macro placement), the LEF file (the macro library) and the constraint file are parsed, two constraint graphs (for vertical and horizontal directions, respectively) are constructed. In the iterative refinement stage, the longest path length is computed in each iteration. If the length is longer than the chip dimension, the longest path will be refined until the length becomes no longer than chip dimension. The coordinates of macros are then determined. In the last stage, simulated annealing is applied on the constraint graphs to further improve the placement.

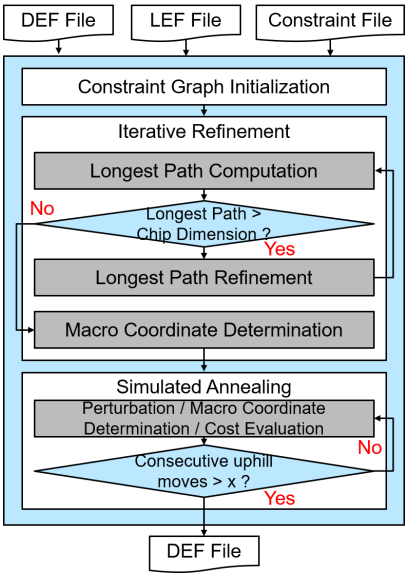

Fig. 1. Algorithm Flow

B. Constraint Graph Initialization

In this stage, two constraint graphs $G_h = (V_h, E_h)$ and $G_v = (V_v, E_v)$ are constructed for horizontal constraints and vertical constraints, respectively.

1) Graph Vertices: Each macro m_i corresponds to a vertex v_i in G_h (e.g., v_1 to v_4 in Figure 2) and a vertex v_i' in G_v. In addition, two extra vertices representing the left boundary and the right boundary are added to G_h (e.g., s and t in Figure 2), while two vertices for the bottom boundary and the top boundary are added to G_v. For rectilinear chips, pseudo macros and corresponding vertices are added to fill the chip into a rectangular one (e.g., p_1 and p_2 in Figure 2). The pseudo macros are set as fixed to make sure these macros won't be moved in the following stages.

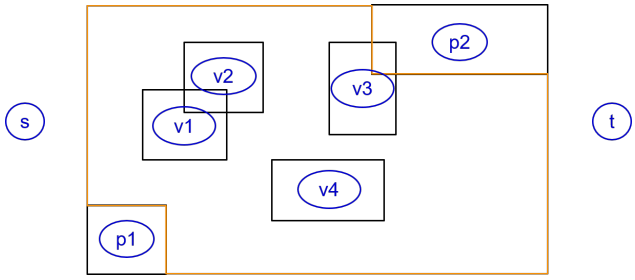

Fig. 2. Chip (the chip boundary is highlighted in orange)

2) Graph Edges: Between each pair of macros (including pseudo macros), exact one constraint edge is added to either G_h or G_v. In addition, an edge is added between each source vertex (left boundary in G_h and bottom boundary in G_v) and each macro, and an edge is added between each macro and each sink vertex (right boundary in G_h and top boundary in G_v). To determine to which graph the edge is added, consider the relative positions of macros as described in, which is inspired by [3]. If the two macros overlap in neither directions, the direction of larger distance is chosen (case 1 in Figure 3). If the two macros overlap in one direction, the other direction is chosen (case 2 in Figure 3). If the two macros overlap on both directions (i.e. need to be legalized), the direction with smaller overlap is chosen (case 3 in Figure 3). In summary, the direction which is less possible to become overlap will be chosen.

The edge weight is determined by formula (2) to (7). For

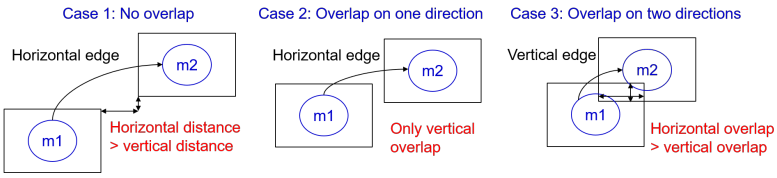

Fig. 3. Determination of edge direction

edges between source vertices (s in G_h and s' in G_v) and macros, the edge weights are set as:

$$w(s, v_i) = \begin{cases} 0 & \text{if } m_i \text{ is movable} \\ m_i.x & \text{if } m_i \text{ is fixed} \end{cases} \tag{2}$$

$$w(s', v_i') = \begin{cases} 0 & \text{if } m_i \text{ is movable} \\ m_i.y & \text{if } m_i \text{ is fixed} \end{cases} \tag{3}$$

For edges between macros and sink vertices (t in G_h and t' in G_v), the edge weights are set as:

$$w(v_i, t) = \begin{cases} m_i.width & \text{if } m_i \text{ is movable} \\ chip.right - m_i.x & \text{if } m_i \text{ is fixed} \end{cases} \tag{4}$$

$$w(v_i', t') = \begin{cases} m_i.height & \text{if } m_i \text{ is movable} \\ chip.top - m_i.y & \text{if } m_i \text{ is fixed} \end{cases} \tag{5}$$

For edges between two macros or pseudo macros, the edge weights are set as:

$$w(v_i, v_j) = m_i.width + powerplan\ width \quad (6)$$

$$w(v'_i, v'_j) = m_i.height + minimum\ channel\ spacing \quad (7)$$

An example initial placement and its corresponding horizontal constraint graph is shown in Figure 4.

C. Iterative Refinement

Inspired by [3], in the iterative refinement stage (Algorithm 1), the longest path of G_H and G_v are computed in each iteration. The longest paths will be refined if they are longer than the corresponding chip dimensions. If both longest path exceeds the corresponding chip dimension, the direction exceeding more will be refined first. To refine the longest path (Algorithm 2), each edge in the path is tested if it's "available", that is, moving the edge to the opposite direction doesn't further violate the chip dimension. The available edge whose two corresponding macros having the largest distance on the opposite direction is chosen as the edge being moved. The iterative refinement stages continues until the longest paths are shorter than chip dimensions in both directions. Note that in each direction, the algorithm only focuses on the longest path. Moreover, the process often terminates after a few iterations empirically.

Algorithm 1: Iterative Refinement

```
//length_h denotes the longest horizontal path length
//length_v denotes the longest vertical path length
while length_h > (chip.right - chip.left) or length_v >
 (chip.top - chip.bottom) do
    if length_h > (chip.right - chip.left) and length_v >
     (chip.top - chip.bottom) then
        if length_h - (chip.right - chip.left) > length_v -
         (chip.top - chip.bottom) then
            refineLongestPath(H, length_v);
        else
            refineLongestPath(V, length_h);
        end
    else if length_h > (chip.right - chip.left) then
        refineLongestPath(H, chip.top − chip.bottom);
    else if length_v > (chip.top - chip.bottom) then
        refineLongestPath(V, chip.right − chip.left);
end
```

D. Simulated Annealing

After feasible constraint graphs are found, simulated annealing is performed to further improve the cost. According to the operations to perturb a TCG [6], three operations are adopted for the constraint graph perturbation:

1) **Move:** Move an edge to the opposite direction
2) **Swap:** Swap two macros in both horizontal and vertical graphs
3) **Reverse:** Reverse an edge in horizontal or vertical graph

Algorithm 2: refineLongestPath(H/V, l_d)

```
input: H/V, chip dimension of the other direction l_d
//path_h denotes the longest horizontal path
//path_v denotes the longest vertical path
if H then
    for each edge_i ∈ path_h do
        Set edge_i available if move edge_i to vertical
         doesn't violate l_d;
    end
    Move the available edge with the largest vertical
     distance to vertical;
else if V then
    for each edge_i ∈ path_v do
        Set edge_i available if move edge_i to
         horizontal doesn't violate l_d;
    end
    Move the available edge with the largest horizontal
     distance to horizontal;
```

The cost function is defined as formula (1). The simulated annealing terminates when the number of consecutive uphill moves exceeds a specific number x.

E. Macro Coordinate Determination

Given the constraint graphs, the locations of the macros can be determined through Algorithm 3, which is inspired by [4]. Macros are moved to a location as close to the initial location as possible, in order to minimize the macro displacement.

Algorithm 3: Macro Location Determination

```
// For x-coordinates determination
Propagate from source vertices to obtain most left
 possible coordinates for macros;
Back-propagate from sink vertices to obtain most right
 possible coordinates for macros;
Compute m_i.slack = m_i.rightmost − m_i.leftmost
 for each m_i;
for each m_i in increasing order of slack do
    if m_i.left <= m_i.x_initial <= m_i.right then
        m_i.x = m_i.x_initial;
    else if m_i.x_initial > m_i.right then
        m_i.x = m_i.right;
    else if m_i.x_initial < m_i.left then
        m_i.x = m_i.left;
end
// Similar for y-coordinates determination (omit)
```

IV. Experimental Result

Table I shows the statistics of the two cases provided by ICCAD 2021 CAD Contest [1], including the weights of cost functions ,the number of macros and the unit distance. Note that the displacements and areas in below tables follow these units.

Table II shows the result with iterative refinement only. With a quite short runtime and few iterations, the method could

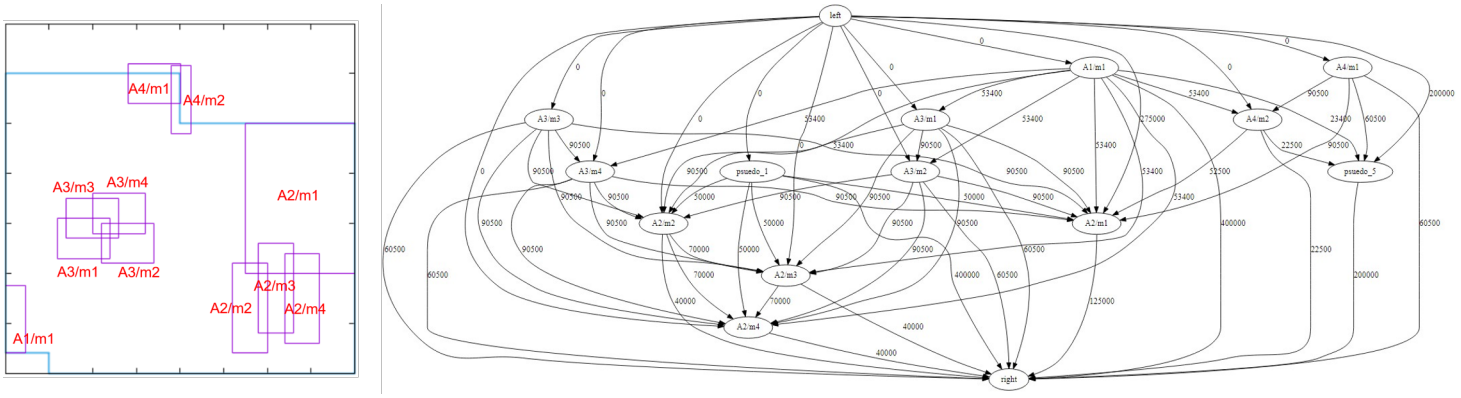

Fig. 4. Example Initial Placement and Corresponding Horizontal Constraint Graph

TABLE I
CASE STATISTICS

Case	α	β	♯ Macros	Unit Distance
caseSample	1	8	11	$\mu m/1000$
case1	1	4	84	$\mu m/2000$

TABLE II
RESULT OF ITERATIVE REFINEMENT

Case	D	A	Cost	CPU time(sec)	♯ Iterations
caseSample	409	2643.8	**820.3**	0.001	1
case1	6998.3	123960.8	**8406.6**	0.005	8

find acceptable solutions.

After iterative refinement, simulated annealing is applied to further improve the solution. The result is shown in Table III. The costs of both cases are further reduced through simulated annealing, especially for the unavailable areas A. However, it takes a longer runtime.

Based on the experimental results, it may be referred that iterative refinement could find an acceptable solution in a short time, while simulated annealing could obtain a better solution, paying a price of longer runtime. As a result, the combination of iterative refinement and simulated annealing may achieve a balance between performance and efficiency.

The layouts of initial placement and the placement after iterative refinement plus simulated annealing of case1 is shown in Figure 5.

V. CONCLUSION

In this work, a constraint-graph based macro legalization algorithm flow that combines iterative refinement and simulated annealing is proposed. Experimental result shows the flow could achieve a balance between cost and runtime.

Future work includes:

1) **Design more aggressive strategies for constraints:** To meet the buffer area reservation requirement, power-plan width constraint is currently relaxed into the edge weight, which might be too conservative.
2) **Apply linear programming on macro location determination:** The current method on the determination of locations is quite fast; however, through linear programming, the locations causing lower cost may be found.

TABLE III
RESULT OF ITERATIVE REFINEMENT + SIMULATED ANNEALING

Case	D	A	Cost	CPU time(sec)
caseSample	421.5	1232.5	**702.4**	0.006
case1	7002.3	90139.7	**8203.2**	0.091

REFERENCES

[1] Maxeda Technology Inc., "ICCAD 2021 CAD Contest Problem D: Macro Legalization," http://iccad-contest.org/2021/tw/

[2] Hsin-Chen Chen, Yi-Lin Chuang, Yao-Wen Chang and Yung-Chung Chang, "Constraint graph-based macro placement for modern mixed-size circuit designs," 2008 IEEE/ACM International Conference on Computer-Aided Design, 2008, pp. 218-223

[3] J. Cong and M. Xie, "A Robust Mixed-Size Legalization and Detailed Placement Algorithm," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 8, pp. 1349-1362, Aug. 2008

[4] Michael D. Moffitt, Jarrod A. Roy, Igor L. Markov, Martha E. Pollack, "Constraint-driven floorplan repair," in ACM Trans. Design Autom. Electr. Syst. 13(4): 67:1-67:13, 2008

[5] Yao-Wen Chang, "Unit 5: Placement (Lecture Notes in Physical Design for Nanometer ICs 2021 Spring)," 2021

[6] Jai-Ming Lin and Yao-Wen Chang, "TCG: A transitive closure graph-based representation for general floorplans," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 13, no. 2, pp. 288-292, Feb. 2005

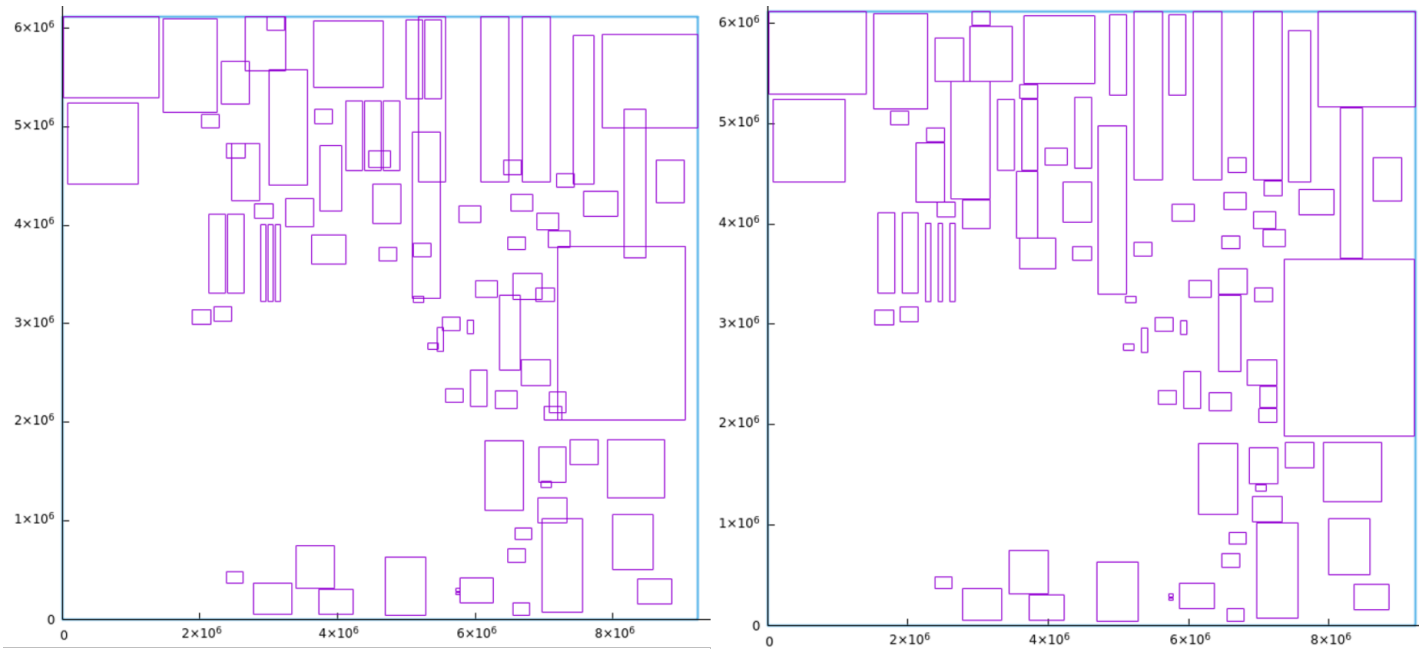

Fig. 5. Layout (left: initial; right: output)