Modern Fixed-Outline Floorplanning with Rectilinear Soft Modules

Yu-Yang Chen, Yi-Chen Lin, Tzu-Han Hsu, Iris Hui-Ru Jiang*†
Tung-Chieh Chen, Tai-Chen Chen, Hua-Yu Chang
Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan
Design Technology Group, Synopsys, Inc., Taipei, Taiwan

ABSTRACT

To better utilize space and reduce wirelength within a fixed-outline with preplaced modules, modern floorplanning is desired to be able to handle rectilinear soft modules. Nevertheless, the induced special shape constraints have not been fully explored in the literature. In this paper, we propose a novel analytical-based approach to address the challenges of fixed-outline, preplaced modules, and rectilinear soft modules. Unlike most previous work, which abstracts modules as circles during global floorplanning, we treat modules as shape-adjustable rectangles and propose a differentiable shape mechanism to capture the impact of shaping on the floorplan quality. For legalization, we first construct an overlap graph to extract the neighborhood of overlaps, modules, and whitespaces. Then, a shortest-path based algorithm effectively migrates area from overlaps through a chain of modules to whitespaces while carefully considering the shape constraints. Finally, we iteratively expand and shrink the bounding boxes of modules to further refine the wirelength by improving area utilization. Our legalization and refinement allows rectilinear shapes to form naturally. Based on the experiments conducted on the GSRC, MCNC, and CAD contest benchmark suites, our results show that our approach achieves superior wirelength and runtime to the state-of-the-art works and the contest winning team, demonstrating its effectiveness and efficiency.

ACM Reference Format:
Yu-Yang Chen, Yi-Chen Lin, Tzu-Han Hsu, Iris Hui-Ru Jiang and Tung-Chieh Chen, Tai-Chen Chen, Hua-Yu Chang. 2024. Modern Fixed-Outline Floorplanning with Rectilinear Soft Modules. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), October 27–31, 2024, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3676536.3676818

1 INTRODUCTION

As the first major stage in physical design, floorplanning defines the top-level spatial structure of a design and provides early feedback for architecture, timing, power, and congestion evaluation. The generated floorplan profoundly affects the design quality of the subsequent placement and routing stages. Therefore, floorplanning is essential and becomes more critical as the hierarchical design style and Intellectual Property (IP) modules become more widespread to handle the ever-growing design complexity [36].

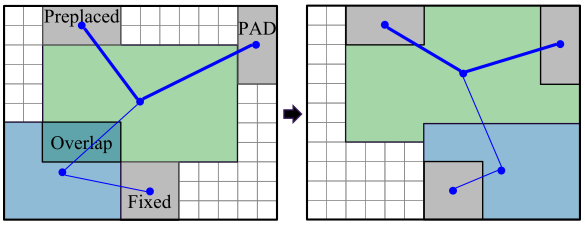


Figure 1: Rectilinear shapes improve the feasibility and wirelength by better space utilization. Assume the module aspect ratio ranges from 0.5 to 2.0. The rectangles in the upper floorplan are of minimum area. Thicker blue lines indicate nets with higher criticality.

In modern design flows, floorplanning is typically done after the die size, I/O pads, and preplaced modules have been determined. This makes area minimization relatively less important during floorplanning [4, 17, 18]. In contrast, the ability to handle the fixed-outline and preplaced modules is of particular importance. In such a scenario, industrial designs usually adopt rectilinear modules (not limited to rectangular, L- or T-shape) to achieve better space utilization and wirelength reduction [17, 24], e.g., modern mobile phones [11, 20]. Consequently, in addition to the fixed-outline constraint and preplaced modules, it is desirable for modern floorplanning to be able to handle *rectilinear soft modules*; as shown in Figure 1, the shapes of rectilinear soft modules are to be decided during floorplanning, with flexible aspect ratios and variant polygonal shapes. The shape flexibility not only enlarges the solution space for achieving more wirelength reduction but also introduces new shape constraints for physical implementation consideration.

Existing floorplanning approaches mainly fall into three categories: 1) *packing-based* approach, 2) *learning-based* approach, and 3) *analytical-based* approach. The *packing-based* approach encodes a non-overlapping floorplan by a topological representation (e.g., sequence-pair [32], B*-tree [7], TCG [27]) and relies on local search (especially simulated annealing [22]) to incrementally modify and optimize the floorplan [5, 35]. The packing-based approach tends to converge to a high-quality solution with a long runtime. Moreover, most methods assume that the shapes of modules are rectangular or limited type of rectilinear (like L- or T-shape). Several packing-based floorplanners can partially handle rectilinear shapes by partitioning a rectilinear module into a set of abutting rectangular submodules, e.g., [35]. However, this handling is valid only when the rectilinear shape is prespecified. Recently, the *learning-based* approach applies reinforcement learning (RL) techniques to enhance the local search heuristics used in the packing-based approach and/or learn the policy to place modules adequately [6, 13, 23, 31, 37]. The RL training process is usually time-consuming. As reported in studies, e.g., [37], test runs sometimes may fail the fixed-outline constraint, thus requiring multiple test runs for unseen designs. The *analytical-based* approach starts with global floorplanning, followed by legalization [16, 25, 26, 28–30]. Global floorplanning allows partial overlaps and seeks the proper relative positions of modules to optimize the objective function based on attractor-repeller (AR) model [30], push-pull (PP) model [28, 29], semi-definite programming (SDP) [25], electrostatic potential energy [16, 26], etc. Because

*Corresponding author.
†This work was supported in part by Google, Synopsys, Siemens EDA, TSMC, and National Science and Technology Council of Taiwan.

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10
https://doi.org/10.1145/3676536.3676818

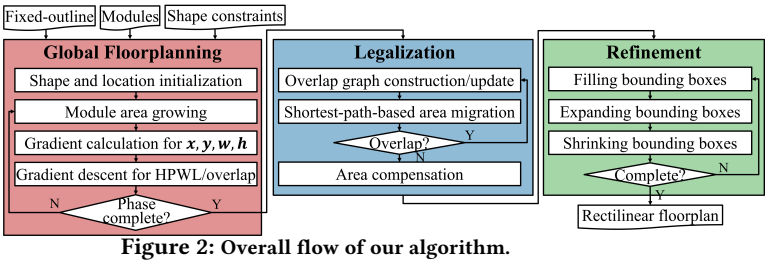


Figure 2: Overall flow of our algorithm.

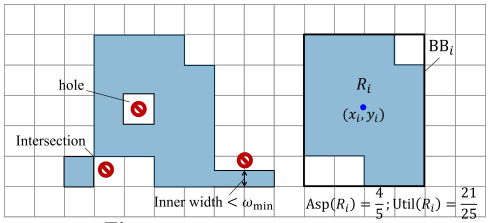


Figure 3: Shape Constraints.

the actual shapes of modules are unknown during global floorplanning, most previous work represents modules as circles [25, 28, 30]. However, we observe that the gap between this abstraction and the actual shapes induces the difficulty and bias of wirelength and overlap optimization. After global floorplanning, legalization removes overlaps, determines module shapes and produces a feasible floorplan. Several legalization algorithms have been proposed in the literature to convert rectangular modules in a near-legal floorplan to rectilinear modules, e.g., [10, 19, 29]. The network-flow-based method in [10] sometimes may convert a module into disconnected polygons, which is invalid. UFO [29] redistributes the space around preplaced modules and then generates rectilinear shapes. Very recently, TOFU [19] adopts a simple greedy strategy that reallocates whitespace to change nearby modules to rectilinear shapes.

Overall, the analytical-based approach has shown its great potential in fixed-outline floorplanning and shape handling. Nevertheless, the special shape constraints induced by rectilinear soft modules have not been fully explored in the literature. Therefore, in this paper, we propose a novel analytical-based approach to address the challenges of fixed-outline, preplaced modules, and rectilinear soft modules. Unlike previous work [25, 28, 30], which abstracts modules as circles during global floorplanning, we treat modules as shape-adjustable rectangles to close the shape gap and propose a differentiable shape mechanism to optimize wirelength and control overlaps. At legalization, we first construct an overlap graph to capture the neighborhood of overlaps, modules, and whitespaces. Then, a shortest-path based algorithm effectively migrates area from overlaps through a chain of modules to whitespaces while carefully considering the shape constraints. Finally, we iteratively expand and shrink the bounding boxes of modules to further refine the wirelength by better area utilization. Our legalization and refinement allows rectilinear shapes to form naturally. The experiments are conducted on the GSRC [1], MCNC [2], and CAD contest [3] benchmark suites. Experimental results show that our approach outperforms the state of the art in wirelength at short runtime, which demonstrates the effectiveness and efficiency of our approach.

The remainder of this paper is organized as follows. Section 2 states the problem formulation. Section 3 outlines our algorithm. Section 4 presents our global floorplanning stage. Sections 5 and 6 detail our legalization and refinement schemes. Section 7 shows experimental results. Finally, Section 8 concludes this work.

2 PROBLEM FORMULATION

As mentioned earlier, the shape flexibility of rectilinear soft modules enlarges the solution space for achieving more wirelength reduction, but we should consider the following shape constraints during floorplanning to ensure success of downstream physical implementation (see Figure 3).

Simple Rectilinear Polygon. The shape R_i of a soft module M_i should be a simple rectilinear polygon. The sides of the polygon should be parallel to the axes of Cartesian coordinates, and thus the interior angle at each corner is either 90° or 270°. It should be hole-free (only a single continuous boundary) and intersection-free (no two sides cross or intersect each other).

Minimum Inner Width. The inner width of each simple rectilinear polygon R_i should be no less than $\omega_{\min}$.

Minimum Area. The polygon area of each module should be no less than its given minimum area, i.e., $\text{Area}(R_i) \geq \text{Area}_{\min}(M_i)$.

Aspect Ratio. The minimum bounding box (rectangle) BB_i drawn by a rectilinear shape R_i should follow the required aspect ratio range, i.e., $\text{Asp}(R_i) = \text{Width}(\text{BB}_i)/\text{Height}(\text{BB}_i) \in [1/AR, AR]$.

Area Utilization. The ratio of the polygon area to the area of its minimum bounding box should be no less than a threshold $\mu_{\min}$, i.e., $\text{Util}(R_i) = \text{Area}(R_i)/\text{Area}(\text{BB}_i) \geq \mu_{\min}$.

Assume the fixed-outline is a rectangle of width W^* and height H^*. If it is not rectangular, blockages can be added accordingly. Let M be the set of m soft modules; each soft module M_i is associated with a adjustable shape R_i and its center coordinate (x_i, y_i), defined by the center coordinate of the minimum bounding box that encloses R_i. Let F be the set of f fixed modules (e.g., preplaced modules, I/O pads, or blockages) of fixed shapes and locations. Let N be the set of n nets; each net N_k is associated with criticality c_k and half-perimeter wirelength (HPWL) $HPWL_k$.

With the above information and knowledge of shape constraints, our goal is to determine the location (x_i, y_i) and rectilinear shape R_i of each soft module M_i to generate a non-overlapping floorplan by solving the **fixed-outline floorplanning with rectilinear soft modules problem** described below.

$$\begin{aligned} &\min \sum_{N_k \in N} c_k \cdot HPWL_k \\ &\text{subject to} \\ &\qquad \text{no module overlap, } O_{ij} = \emptyset \ \forall M_i, M_j \in M \cup F \\ &\qquad \text{shape constraints} \\ &\qquad \text{all modules are within the fixed-outline.} \end{aligned} \tag{1}$$

3 ALGORITHM OVERVIEW

As shown in Figure 2, we propose a novel analytical-based floorplanning approach to address the challenges of fixed-outline, preplaced modules, and rectilinear soft modules. During global floorplanning (detailed in Section 4), we treat modules as shape-adjustable rectangles to close the shape gap. Via a differentiable shape mechanism, we capture the impact of module shaping on wirelength and overlap. We gradually grow the rectangle area of each module to optimize wirelength while controling overlaps. At legalization (detailed in Section 5), we first construct an overlap graph to capture the neighborhood of overlaps, modules, and whitespaces. Then, a shortest-path based algorithm migrates area from overlaps through a chain of modules to whitespaces. Finally, at refinement (detailed in Section 6), we iteratively expand and shrink the bounding boxes of modules to further improve the wirelength by better area utilization. Through legalization and refinement, rectilinear shapes can form naturally.

4 GLOBAL FLOORPLANNING

The goal of global floorplanning is to decide the optimal locations for soft modules that minimize HPWL, while allowing for a tolerance of overlaps. We model this goal into an optimization problem and apply gradient descent iteratively to find the minimum value of the objective function. Our objective function is formulated by relaxing the overlap area constraint with a penalty coefficient α:[1]

$$\begin{aligned} &\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{w},\boldsymbol{h}} \sum_{k\in[1,n]} c_k \cdot HPWL_k + \alpha \cdot \sum_{i,j\in[1,m+f]} \text{Area}(O_{ij}) \\ &\text{subject to} \\ &\quad \frac{1}{AR} \le \frac{w_i}{h_i} \le AR, \forall i \in [1,m] \\ &\quad \text{all modules are within the fixed-outline.} \end{aligned} \tag{2}$$

Here, $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{w}$, and $\boldsymbol{h}$ are the vectors formed by the x-coordinates, y-coordinates, widths, and heights of all soft modules in M, respectively. There are two constraints that soft modules should meet during optimization. First, the aspect ratio of each soft module must not exceed the given limit. Second, the placement of soft modules must be within the fixed-outline.

Note that in our objective function, the dimensions of the soft modules are also included as parameters. Different from prior studies [25, 28–30], which model soft modules as circles during the global floorplanning stage, our approach represents these modules as rectangles. This rectangular modeling offers three distinct advantages. Firstly, it reduces the gap between the abstracted and actual shapes. Secondly, it simplifies the calculation of overlap areas and their gradients, enhancing the computational efficiency. Thirdly, the ability to adjust the dimensions of soft modules can, in certain scenarios, contribute to a reduction in the HPWL.

4.1 Gradient Calculation

When calculating the gradient of the objective function, it is inevitable that some functions will be non-differentiable. In our work, we directly define the derivatives of these non-differentiable functions, which will be introduced in this section.

4.1.1 max, min *functions.* max and min are the two non-differentiable functions which appear in our objective function. Since $\max(\boldsymbol{x}) = -\min(-\boldsymbol{x})$, we propose our method for calculating the gradient of max, which can also be applied to calculate the gradient of min. Consider the partial derivative $\partial \max(x,y)/\partial x$:

$$\begin{aligned} &\text{(i) } x > y, \max(x,y) = x \Rightarrow \frac{\partial \max(x,y)}{\partial x} = \frac{\partial x}{\partial x} = 1 \\ &\text{(ii) } x < y, \max(x,y) = y \Rightarrow \frac{\partial \max(x,y)}{\partial x} = \frac{\partial y}{\partial x} = 0 \\ &\text{(iii) } x = y, \frac{\partial \max(x,y)}{\partial x} = \text{undefined} \end{aligned} \tag{3}$$

In condition (iii), $\partial \max(x,y)/\partial x$ is undefined since $\max(x,y)$ is not smooth at $x = y$. To address the non-differentiability, we define $\partial \max(x,y)/\partial x = 0$ when $x = y$, leading to

$$\frac{\partial \max(x,y)}{\partial x} = \begin{cases} 1, & x > y \\ 0, & x \le y \end{cases} \tag{4}$$

This approach to handling non-differentiable functions can also be found in the field of machine learning. A widely used activation function, ReLU (Rectified Linear Unit) is defined as

$$\text{ReLU}(x) = \max(0,x) \tag{5}$$

During the backpropagation process, the derivative of ReLU is specified as

$$\frac{d\ \text{ReLU}(x)}{dx} = \begin{cases} 1, & x > 0 \\ 0, & x \le 0 \end{cases}, \tag{6}$$

which applies the same strategy to handle non-differentiable functions as ours.

When it comes to the gradient calculation of max with multiple arguments, previous work uses approaches such as Weighted-Average (WA) [15] or Log-Sum-Exp (LSE) [33] to find a smooth approximation. In our work, we define the derivative of max with multiple arguments by expanding Equation (4), leading to

$$\begin{aligned} \nabla \max(\boldsymbol{x}) &= \begin{cases} \boldsymbol{0}, & x_1 = x_2 = \ldots = x_n \\ \boldsymbol{g}, & \text{otherwise} \end{cases} \\ \text{where } g_i &= \begin{cases} 1, & x_i = \max(\boldsymbol{x}) \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{7}$$

With this definition, we adjust the objective function to be everywhere differentiable, and the differential calculation is immensely simplified. Moreover, we are able to directly calculate the gradient of each parameter based on their geometric relationships, which will be shown in the following sections.

4.1.2 HPWL. The HPWL of a net N_k is calculated as

$$HPWL_k = \max_{i\in N_k} x_i - \min_{i\in N_k} x_i + \max_{i\in N_k} y_i - \min_{i\in N_k} y_i \tag{8}$$

According to the derivative definitions of max and min described in Equation (7), only modules on the bounding box of N_k have gradients. Thus, gradient descent minimizes the size of the bounding box, as depicted in Figure 4, which matches the objective in global floorplanning.

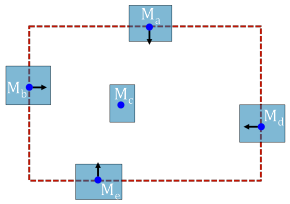


Figure 4: The direction of gradient descent of a net connecting $\{M_a, M_b, M_c, M_d, M_e\}$.

4.1.3 Overlap Area w.r.t. Coordinates. The overlap area between two modules M_i and M_j can be calculated as

$$\begin{aligned} \text{Area}(O_{ij}) &= W_o \cdot H_o \\ \text{where } W_o &= \min(x_{ir}, x_{jr}) - \max(x_{il}, x_{jl}) \\ &= \min(x_i + \frac{1}{2}w_i, x_j + \frac{1}{2}w_j) - \max(x_i - \frac{1}{2}w_i, x_j - \frac{1}{2}w_j) \\ H_o &= \min(y_{iu}, y_{ju}) - \max(y_{ib}, y_{jb}) \\ &= \min(y_i + \frac{1}{2}h_i, y_j + \frac{1}{2}h_j) - \max(y_i - \frac{1}{2}h_i, y_j - \frac{1}{2}h_j) \end{aligned} \tag{9}$$

The partial derivative of the overlap area with respect to x_i is calculated as

[1] Different from the global placement stage of analytical placement, which often imposes a density constraint to distribute modules uniformly, global floorplanning only needs to minimize module overlap. As a result, it is sufficient to directly relax the overlap area constraint into the objective function.

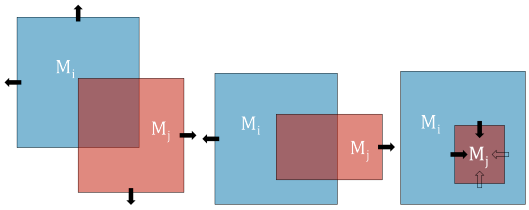


Figure 5: Illustration of the gradient descent direction gradient descent direction of two overlapping modules.

$$\frac{\partial(W_o \cdot H_o)}{\partial x_i} = H_o \cdot \frac{\partial W_o}{\partial x_i}$$

where

$$\frac{\partial W_o}{\partial x_i} = \frac{\partial \min(x_i + \frac{1}{2}w_i, x_j + \frac{1}{2}w_j)}{\partial x_i} - \frac{\partial \max(x_i - \frac{1}{2}w_i, x_j - \frac{1}{2}w_j)}{\partial x_i} \tag{10}$$

Based on the derivative definitions of max and min described in Equation (4), the computation of $\partial W_o/\partial x_i$ can be determined by examining the geometric relationships between the right and left sides of the two modules. Similarly, the partial derivative $\partial W_o/\partial y_i$ is derived from the relationships between the upper and lower sides of these modules. Figure 5 shows the direction of gradient descent on three overlapping scenarios of the modules. Note that in Figure 5(c), the gradient descent direction suggests a reduction in the size of M_j. To maintain compliance with the area constraint of each module, the gradients on the sides where the arrows appear hollow in the figure are nullified. This approach ensures that adjustments to module positions do not result in violations of minimum area constraint, thus optimizing the configuration within the designated boundaries.

4.1.4 Overlap Area w.r.t. Soft Modules' Dimensions. The partial derivative of the overlap area with respect to the width of a soft module, w_i, can be calculated as

$$\frac{\partial(W_o \cdot H_o)}{\partial w_i} = H_o \cdot \frac{\partial W_o}{\partial w_i}$$

where

$$\frac{\partial W_o}{\partial w_i} = \frac{\partial \min(x_i + \frac{1}{2}w_i, x_j + \frac{1}{2}w_j)}{\partial w_i} - \frac{\partial \max(x_i - \frac{1}{2}w_i, x_j - \frac{1}{2}w_j)}{\partial w_i} \tag{11}$$

Similar to the calculation described in Section 4.1.3, the derivatives $\partial W_o/\partial w_i$ and $\partial W_o/\partial h_i$ can be calculated based on the geometric relationships between M_i and M_j. The main difference is that w_i and h_i are dependent on each other. Consequently, during the gradient descent optimization process, the updates to w_i and h_i are conducted alternately.

4.2 Special Treatment in Global Floorplanning

4.2.1 Gradient Descent Techniques. Several gradient-descent-based techniques are employed in global floorplanning. Firstly, Adam [21] serves as our optimizer. This method leverages an adaptive learning rate and momentum, which prevents gradient descent from getting stuck at local minima or saddle points, and thus shortens the runtime and enhances result quality. Secondly, to ensure compatibility with various chip outlines, particularly those that are extreme, the outline of the chip is normalized to an aspect ratio of 1. Thirdly, to avoid gradient explosion, gradient clipping is implemented.

4.2.2 Growth of Soft Modules. It is obvious that the overlap area is the primary factor preventing the final floorplan from reaching a global optimum. To address this issue, we have organized the global floorplanning process into multiple phases. In this phased approach, we initially set the soft modules to be small as dots. Afterward, the sizes of these modules are incrementally increased in each phase. This strategy allows us to prioritize optimal module placement in the initial phases, while gradually adding the consideration for module sizes in later stages.

Furthermore, the expansion of these soft modules can be strategically controlled to minimize the overlap area as they grow. To do this, we measure the overlap area on each side of every soft module, denoted as A_t, A_b, A_l, and A_r. Then we use softmax function to decide how much each side should grow. (Softmax is widely used in deep learning to normalize the scale of each value in a vector between 0 and 1.) For instance, the fraction of growth for the top side is given by the formula:

$$\frac{\exp(-A_t)}{\exp(-A_t) + \exp(-A_b) + \exp(-A_l) + \exp(-A_r)} \tag{12}$$

We calculate similar fractions for the growth on the other sides, ensuring that the sides with more overlap grow less.

4.3 Overall Procedure

The overall procedure of global floorplanning is listed in Algorithm 1. At the beginning, all soft modules are placed at the center of the chip as the initial position. Note that each soft module's size increases quadratically, which corresponds to linear growth of the dimensions (width, height) of each module.

Algorithm 1 Global Floorplanning Procedure

```
P ← Number of phases
I ← Number of iterations
Place all soft modules at the center of the fixed-outline
for p ← 1 to P do
    Grow each soft module M_i to size (p/P)^2 · Area_min(M_i)
    for i ← 1 to I do
        Calculate gradient for each parameter
        Apply gradient descent
    end for
end for
```

5 LEGALIZATION

The goal of the legalization stage is to eliminate all overlaps by reshaping modules to a rectilinear shape. The resulting legalized floorplan must be as close as possible to the result of global floorplanning. Furthermore, extra care should be taken to make sure the final rectilinear shapes satisfy the shape constraints.

Inspired by [17], the basic idea is to migrate area from overlaps to whitespace. Given the topological information of a global floorplan, a graph G that represents the neighboring relation of each module can be constructed. We name this graph G an "overlap graph". Area is then migrated by traversing some path from overlap to whitespace, repeating until all overlaps have been resolved.

5.1 Overlap Graph

The overlap graph $G = (V, E)$ is a weighted directed graph, as shown in Figure 6. First, define the region that module M_i covers as R_i. Note that although R_i is normally perfectly rectangular in a global floorplan, the following definitions still hold even if R_i were rectilinear in shape.

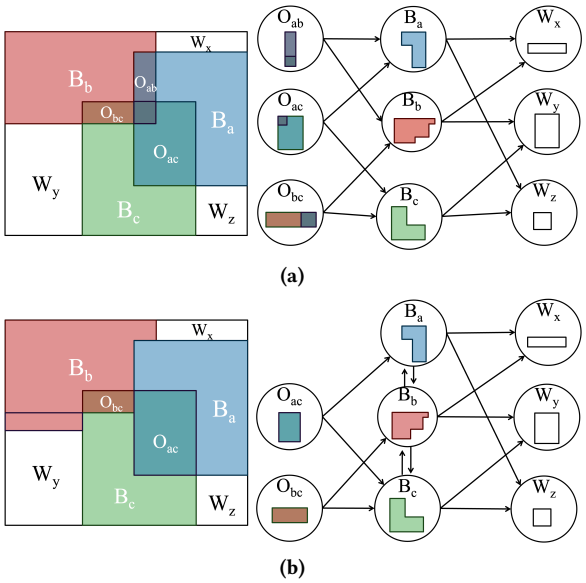


Figure 6: Example of an overlapping floorplan and its corresponding overlap graph. (a) The initial floorplan and overlap graph. No block-block edges exist here because no common boundary is shared between these blocks. (b) A possible configuration after resolving the overlap O_{ab}. Since the geometry has changed, B_b now shares a common boundary with B_a and B_c, so new edges are drawn.

5.1.1 Vertices. We define three types of vertices: overlap vertices, block vertices, and whitespace vertices. If any two modules M_i and M_j overlap, then define the overlapping region between the two modules as O_{ij}. Then, for each soft module we define non-overlapping regions as B_i. Formal definitions are stated in Equations (13) and (14). Finally, each continuous whitespace region is defined as W_A. For all the regions defined above, a corresponding vertex with the same name is drawn.

$$O_{ij} = R_i \cap R_j \ \forall \text{ overlapping module pairs } M_i \text{ and } M_j \tag{13}$$

$$B_i = R_i - \bigcup_{j \in [1, m+f], i \neq j} O_{ij} \ \forall \text{ soft modules } M_i \tag{14}$$

5.1.2 Edges. For each overlap vertex O_{ij}, draw two directed edges (O_{ij}, B_i) and (O_{ij}, B_j). If O_{ij} is an overlap between a soft module and preplaced module, then only one edge connecting O_{ij} and the soft module is drawn. Then, consider all pairs of block regions B_i and B_j which share a common boundary. If the length of this common boundary exceeds a certain threshold, e.g., $\omega_{\min}$, then we draw two directed edges (B_j, B_i) and (B_i, B_j). Likewise, consider all block-whitespace pairs B_i and W_A that share a common boundary. If the length of this common boundary exceeds a certain threshold, e.g., $\omega_{\min}$, then we draw an edge (B_i, W_A).

5.1.3 Edge Weights. Recall that one of the secondary goals during legalization is to ensure the final rectilinear shape of each module is as regular as possible. To do this, we introduce edge weights for every edge on the overlap graph to reflect this objective. The main idea is that the smaller an edge weight, the less likely the module would become irregularly shaped if area were migrated via this edge.

As stated above, there are three types of edges in the overlap graph G. The edge weight of each type is calculated differently.

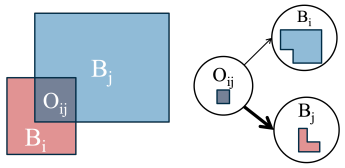


Figure 7: An overlap O_{ij} and its corresponding overlap-block edges. Thicker edges indicate larger edge weights.

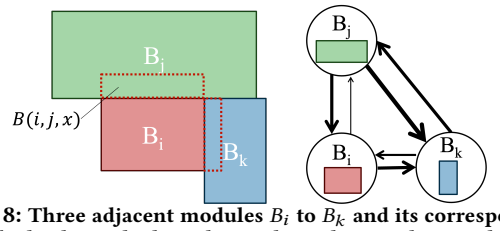


Figure 8: Three adjacent modules B_i to B_k and its corresponding block-block edges. Thicker edges indicate larger edge weights.

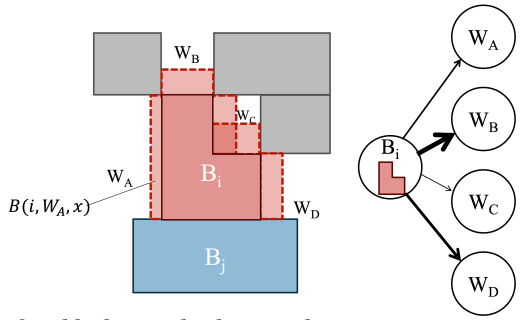


Figure 9: A block B_i with adjacent whitespaces W_A to W_D and its corresponding block-whitespace edges. Thicker edges indicate larger edge weights.

For an overlap-block edge (O_{ij}, B_i), such as in Figure 7, define the edge weight $w(O_{ij}, B_i)$ as:

$$w(O_{ij}, B_i) = \alpha_{OB} \mathrm{A}^{\mathrm{OB}} + \beta_{OB} \mathrm{B}^{\mathrm{OB}} + \gamma_{OB} \mathrm{C}^{\mathrm{OB}}, \tag{15}$$

where $\mathrm{A}^{\mathrm{OB}} = \mathrm{Area}(O_{ij})/\mathrm{Area}(B_i)$, $\mathrm{B}^{\mathrm{OB}} = 1 - \mathrm{Util}(B_i)$, $\mathrm{C}^{\mathrm{OB}} = (1 - \mathrm{Asp}(B_i))^4$. $\alpha_{OB}, \beta_{OB}, \gamma_{OB}$ are hyperparameters.

Next, consider a block-block edge (B_i, B_j), like in Figure 8. If we want x amount of area to be migrated via this edge, then we say that region B_i "borrows" x area from B_j. The borrowed region can be found by extending the common boundary in the normal direction until a rectangle with $\geq x$ area is formed. Denote the borrowed region $B(i, j, x)$. Then, define the edge weight $w(B_i, B_j, x)$ as:

$$w(B_i, B_j, x) = \alpha_{BB} \mathrm{A}^{\mathrm{BB}} + \beta_{BB,i} \mathrm{B}^{\mathrm{BB,i}} + \beta_{BB,j} \mathrm{B}^{\mathrm{BB,j}} + \gamma_{BB} \mathrm{C}^{\mathrm{BB}} + \lambda_{BB}, \tag{16}$$

where $\mathrm{A}^{\mathrm{BB}} = \mathrm{Area}(B_i)/\mathrm{Area}(B_j)$, $\mathrm{B}^{\mathrm{BB,i}} = 1 - \mathrm{Util}(B_i \cup B(i, j, x))$, $\mathrm{B}^{\mathrm{BB,j}} = 1 - \mathrm{Util}(B_j - B(i, j, x))$, $\mathrm{C}^{\mathrm{BB}} = (1 - \mathrm{Asp}(B_i \cup B(i, j, x)))^4$. $\alpha_{BB}, \beta_{BB,i}, \beta_{BB,j}, \gamma_{BB}, \lambda_{BB}$ are hyperparameters.

Finally, consider a block-whitespace edge (B_i, W_A), like in Figure 9. Assume that we want x amount of area to be migrated via this edge. A newly constructed region can be found in a similar way to the above case, which we denote $B(i, W_A, x)$. Then, define the edge weight $w(B_i, W_A, x)$ as:

$$w(B_i, W_a, x) = \beta_{BW} \mathrm{B}^{\mathrm{BW}} + \gamma_{BW} \mathrm{C}^{\mathrm{BW}}, \tag{17}$$

where $\mathrm{B}^{\mathrm{BW}} = 1 - \mathrm{Util}(B_i \cup B(i, W_A, x))$, $\mathrm{C}^{\mathrm{BW}} = (1 - \mathrm{Asp}(B_i \cup B(i, W_A, x)))^4$. β_{BW}, γ_{BW} are hyperparameters.

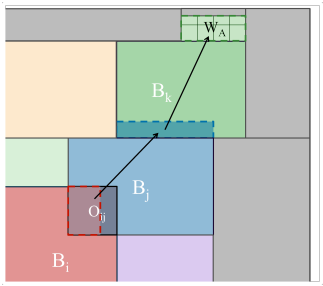


Figure 10: A close up of an overlapping floorplan example and a possible shortest path $(O_{ij}, B_j), (B_j, B_k), (B_k, W_A)$ from the overlap to some whitespace. Regions with dotted outlines indicate the regions where area will be migrated.

Note that for an edge (B_i, B_j) or (B_i, W_A), there may be multiple choices of $B(i, j, x)$ or $B(i, W_A, x)$ due to multiple adjacent boundaries. In this case, we choose the one that minimizes the edge weight.

5.2 Graph-Based Area Migration Algorithm

Based on the definitions given in Section 5.1.3, the smaller the total edge weight of a path, the less likely the resulting floorplan will violate shape constraints. Thus, the overlap removal procedure becomes an iterative shortest path problem. Algorithm 2 details the full legalization procedure.

Algorithm 2 Legalization Procedure

```
1:  Construct overlap graph G
2:  while overlap exists do
3:      Choose some overlap O_ij
4:      x ← Area(O_ij)
5:      Calculate all edge weights:
        w(O_ij, B_i), w(B_i, B_j, x), w(B_i, W_A, x)
6:      O_ij ⇝ W_A ← shortest path to nearest whitespace W_A
7:      x ← min(Area(O_ij), Area(W_A))
8:      for all edge e ∈ O_ij ⇝ W_A do
9:          Migrate x area along e
10:     end for
11:     Update graph G
12: end while
13: for all modules M_i with inner width or area violation do
14:     Remove spikes from M_i
15:     Compensate M_i for insufficient area
16: end for
```

In Line 6, a simple path $O_{ij} \rightsquigarrow W_A$ from an overlap vertex to a whitespace vertex is found at each iteration. For each edge along this path, a region with x area must be migrated between the two regions connected by that edge, as shown in Figure 10. If this edge is a block-block or block-whitespace edge, then the region is created using the same method as above. If this edge is an overlap-block edge, for instance (O_{ij}, B_i), then starting from the segment in the opposite of the common boundary, a rectangle of size x (defined as B') is removed from O_{ij}, and given to the other overlapping module (a.k.a. $O_{ij} \leftarrow O_{ij} - B', B_j \leftarrow B_j \cup B'$).

As a result of area migration, the adjacency relations and areas of each region may change after each iteration. For example, a whitespace W_A may no longer exist after migrating area to it, so W_A is removed from G. Thus in Line 11, the graph G is updated accordingly. By this process, we can ensure that each module owns the same amount of area after each iteration besides the neighboring relations being maintained.

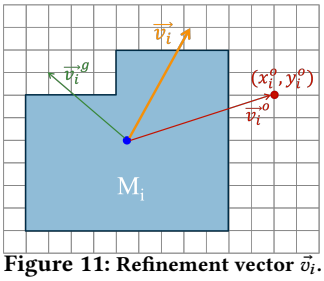
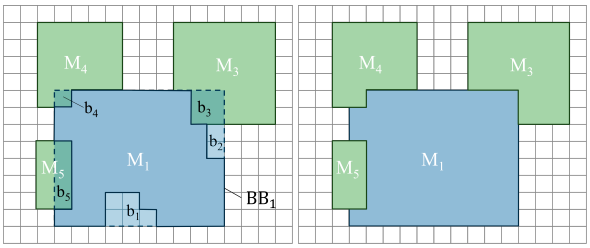


Figure 11: Refinement vector $\vec{v}_i$.



(a) Before filling. **(b) After filling.**

Figure 12: Filling BB_1 of soft module M_1 ($\mu_{\min} = 0.8, \omega_{\min} = 2$).

However, certain violations may still happen at the end of legalization. We directly fix modules that fail these constraints in Lines 13–16. Firstly, the rectilinear shape of some modules may violate the inner width constraint. To fix this, the thin regions that violate this constraint (we call these regions "spikes") are directly removed. Then, either as a result of the previous step or a rounding error in integer division, some modules may end up with less area than required. These modules are compensated by attempting to fill in the whitespace encompassed by its bounding box. If the module still has insufficient area, then its contour is expanded outwards until the module encompasses enough area.

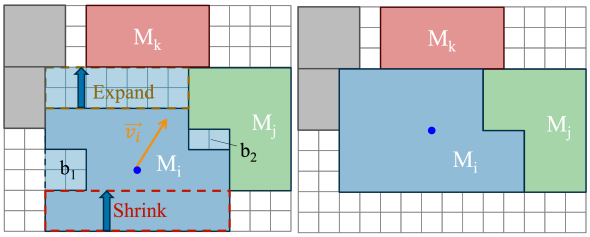
Our legalization framework leverages the high flexibility of rectilinear modules and resolves overlaps iteratively. Good runtime can be achieved by utilizing efficient shortest-path algorithms such as Djikstra's [9] or A* search [12] algorithm.

6 REFINEMENT

The refinement stage aims to propel soft modules towards the wirelength advantageous location, while ensuring the legality of all modules. Despite the rectilinear shape constraint allowing for diverse contour formations of soft modules, it is ultimately the location of the bounding box that impacts HPWL performance. A refinement vector indicates the direction of movement for the module, integrating both global and local influences. Subsequently, the bounding box of the soft module is filled to enhance utilization and reserve area for subsequent processes. Finally, the bounding box is shifted towards the desired direction through iterative adjustments of its sides, leading to improvements in HPWL.

6.1 Refinement Vector

The refinement vector $\vec{v}_i$ guides how the bounding box of M_i should move throughout the refining process. It integrates two pivotal vectors: the optimal center vector ($\vec{v}_i^{\,o}$) and the the gradient descent vector ($\vec{v}_i^{\,g}$); see Figure 11. The former offers a global perspective on the moving direction, while the latter provides insight into local movements. The optimal center (x_i^o, y_i^o) for module M_i can be computed by taking the weighted average of all connected module center coordinates. $\vec{v}_i^{\,o}$ then points towards (x_i^o, y_i^o). While $\vec{v}_i^{\,o}$ offers valuable guidance for the long-term movement direction, exclusive reliance on this factor poses risks. Given the densely populated layout characteristic of floorplans with numerous modules,



(a) Before refinement. (b) After refinement.

Figure 13: Refinement operation on M_i. b_1 and b_2 are merged into M_i by filling BB_i. BB_i then expands and shrinks according to $\vec{v}_i$.

it is probable that M_i will encounter adjacent modules shortly after minor adjustments, impeding its movement towards (x_i^o, y_i^o). Conversely, the gradient descent vector $\vec{v}_i^g$ signifies the direction of movement that offers the most immediate benefit, which is calculated by the same method introduced in Section 4.1. The refinement vector $\vec{v}_i$ integrates both aspects and is defined as follows: $\vec{v}_i = \lambda \vec{v}_i^o + (1-\lambda)\vec{v}_i^g$, where λ is a hyperparameter that adjusts between moves and iterations to strike a balance between the two characteristics.

6.2 Filling Bounding Box

Filling the bounding box benefits the subsequent refining process by augmenting the module's utilization and residual area while also aiding in shaping its contour into a more rectangular form. The process commences by detecting all candidate blocks within the bounding box, and any block that represents a whitespace or belongs to another soft module is of our interest. Each candidate block b_i is accepted and merged into M_i if the move maintains legality on both the donor module and M_i. This iterative process continues until no new blocks are merged.

As illustrated in Figure 12, M_1 fills its bounding box by initially identifying candidate blocks b_1 through b_5. b_1 and b_2, representing whitespaces, are merged after confirming the legality of M_1 following the acceptance of the blocks. Similarly, b_3 is accepted with an additional check to ensure that M_3 remains compliant with constraints following the removal of b_3. Blocks like b_4 and b_5 are not merged due to violations of shape constraints post-movement. Specifically, M_4 violates the minimum inner width constraint, while M_5 violates the aspect ratio constraint if their corresponding blocks are acquired by M_1.

6.3 Moving Bounding Box

Moving the bounding box towards a desired direction shifts the module's center position, thereby reducing HPWL. To ensure simplicity and efficiency, only one side of the bounding box is moved during each operation. If the movement shrinks BB_i, a legality check on M_i alone suffices since no other modules are involved. If the movement expands the bounding box, the filling operations (in Section 6.2) are conducted along with shape continuity checks on M_i since disjoint modules are considered illegal. If the aspect ratio or utilization constraint is violated after moving the selected side, remedies like moving the other sides of the bounding box is conducted. Ultimately, the expansion of the bounding box movement is accepted only if it preserves legality constraints on all moved modules and improves overall HPWL. Figure 13 demonstrates the refinement move of M_i. Whitespaces b_1 and b_2 are merged into M_i by filling BB_i. The upper side is determined by $\vec{v}_i$, causing BB_i to expand upwards as its upper side moves. Similarly, BB_i shrinks by moving the lower side of BB_i upwards. Both movements shift BB_i toward $\vec{v}_i$, resulting in a favorable reduction of HPWL.

6.4 Overall Procedure

Algorithm 3 outlines the process by which soft modules move towards their refinement vectors through the filling and adjustment of their respective bounding boxes. We start with the module with the maximum magnitude of refinement vector. Based on the filling and moving operations, we reshape (expand or shrink) its module contour towards the wirelength advantageous location. This process is repeated until no more movements can be done.

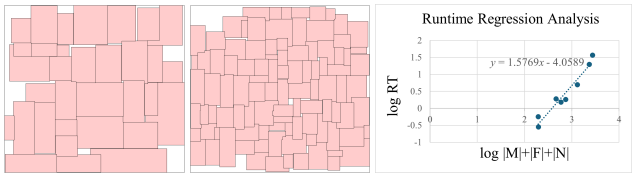
Algorithm 3 Refinement Procedure

```
repeat
    Unlock all modules
    while unlocked module exist do
        M_i ← unlocked module with max_i|v_i|
        Fill BB_i of M_i
        repeat
            S_x ← side to expand BB_i according to v_i
            Expand BB_i by moving S_x side
        until BB_i can no longer move
        repeat
            S_s ← side to shrink BB_i according to v_i
            Shrink BB_i by moving S_s side
        until BB_i can no longer move
        Lock M_i
    end while
until no soft module can be further refined
```

7 EXPERIMENTAL RESULTS

Our algorithm was implemented in C++ programming language with Boost library and evaluated on an Apple M2 MacBook Pro notebook with 3.49 GHz CPU and 16 GB memory.

The first experiment was conducted on commonly used GSRC [1] and MCNC [2] benchmark suites. As listed in Table 1, we compared



(a) ami33 (b) n100 (c) Runtime analysis

Figure 14: Floorplans and runtime analysis of our algorithm.

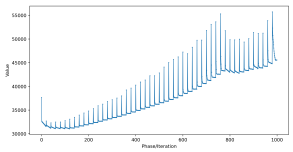
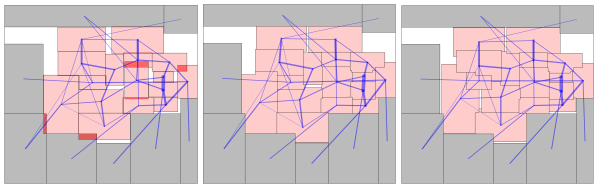


Figure 15: Objective function plot on ami33 during global floorplanning.



(a) (b) (c)

Figure 16: Floorplan of case02 generated by our algorithm. (a) Global floorplanning. (b) Legalization. (c) Refinement. Red boxes indicate overlaps, thicker lines indicate nets with higher criticality.

Table 1: Comparison on GSRC and MCNC benchmark suites ($W^* : H^* = 1 : 1$; $AR = 3$)

Benchmark	Statistics			'10 UFO [28]		'11 UFO [29]		GoodFloorplan [37]		TOFU [19]		SDP [25]		Ours		
				Rectangular		Rectangular		Rectangular		Rectilinear		Rectilinear		Rectilinear		
	$\|M\|$	$\|F\|$	$\|N\|$	HPWL	RT	HPWL	RT	HPWL	RT	HPWL	RT	HPWL	RT	HPWL_G	HPWL	RT
n10	10	69	118	45193	3	36398	3	N/A	N/A	35123	1.86	36277	N/A	33830	32810	0.28
n30	30	212	349	120110	57	102100	87	N/A	N/A	109189	3.54	106013	N/A	99710	97534	1.52
n50	50	209	485	143170	107	124300	204	N/A	N/A	147650	7.43	136547	N/A	123550	121342	1.81
n100	100	334	885	240430	391	195200	677	309320	N/A	250942	24.32	228040	N/A	192342	189571	4.98
n200	200	564	1585	385440	1587	346660	3306	558330	N/A	484507	87.07	407091	N/A	351313	345852	19.65
n300	300	569	1893	526330	3441	476560	4718	690760	N/A	712547	197.25	N/A	N/A	479704	470681	36.69
ami33	33	40	122	50699	N/A	50699	N/A	87540	N/A	71069	2.87	N/A	N/A	45545	44669	0.57
ami49	49	22	396	671920	N/A	671920	N/A	1067590	N/A	1144620	5.99	N/A	N/A	628085	586760	1.89
avg. Δ / $O(\cdot)$				19.62%	$O(\text{n}^{2.586})$	6.52%	$O(\text{n}^{2.757})$	69.86%		39.84%	$O(\text{n}^{1.579})$	13.96%		2.64%	0.00%	$O(\text{n}^{1.577})$

[1]The runtimes (RT) (units in second) of previous works are quoted from their papers and listed here for reference. N/A means no runtimes reported in their papers. In [25], the reported runtime per iteration ranges from seconds to hundreds of seconds. [2]For fair comparison, I/O pads are fixed on the boundary of the chips in the same way as [25], serving as the set of fixed modules F. [3][25] adopts a legalization algorithm similar to [19, 29] to generate rectilinear floorplans. [4]avg. Δ lists the average HPWL difference. [5]$O(\cdot)$ lists the empirical time complexity by regression analysis, where $\text{n} = |M| + |F| + |N|$ represents the number of soft modules, fixed modules, and nets.[6]We provide two versions in our results: HPWL_G represents the wirelength after legalization, and HPWL represents the wirelength after refinement. The results show that both of our versions outperform the other methods.

Table 2: Comparison on CAD Contest benchmark suite ($\omega_{\min} = 30$; $\mu_{\min} = 0.8$; $AR = 2$)

Benchmark	Statistics				1st Place		Ours					
					w/ spikes	w/o spikes	Global Floorplanning		Legalization		Refinement	
	$\|M\|$	$\|F\|$	$\|N\|$	W^*/H^*	HPWL	HPWL	HPWL	RT	HPWL	RT	HPWL	RT
case01	15	5	45	11267/10450	142648351	144074741	148842230	0.204	151676370	0.004	141097909	0.426
case02	16	8	39	2300/2300	17647636	18366208	18172921	0.130	18385512	0.094	17672388	0.254
case03	28	14	108	2500/3000	1790497	1813731	1799815	0.301	1818443	0.095	1779146	0.366
case04	20	8	47	4995/4407	57488025	60376550	60645088	0.370	61442538	0.004	56885700	0.305
case05	16	8	33	4620/3740	14308200	14706300	15012200	0.111	15343850	0.009	14388500	0.287
case06	21	13	56	3000/2700	35576350	36642600	34581550	0.150	34976650	0.001	34694650	0.027
case07	16	11	39	12200/12400	358878150	379504500	378991700	0.117	384267400	0.014	355815550	0.482
case08	37	4	72	21500/14480	95074900	97965450	95819450	0.370	96024250	0.007	93211100	0.370
case09	14	7	19	22570/18200	84403250	90645250	83753750	0.102	87682000	0.004	75539750	0.219
case10	7	5	11	13510/13400	16564550	17145500	16767700	0.027	17059150	0.001	15308000	0.108
avg. Δ					2.55%	5.95%	4.98%		6.77%		0.00%	

*The runtime of 1st place is 25 minutes for each case set by the executable binary. Our runtimes (RT) are reported in seconds.

our algorithm with the state of the art. Overall, our algorithm achieves the best HPWL for each case. Compared with '10 UFO [28], '11 UFO [29], GoodFloorplan [37], TOFU [19], and SDP [25], ours on average reduces HPWL by 19.62%, 6.52%, 69.86%, 39.84%, and 13.96%, respectively. Due to the limited space, we reported only the results on the chip aspect ratio of 1 : 1 here; our results show that other ratios share the same trend. In addition, although our algorithm was evaluated on a different platform than previous work, we performed runtime regression analysis, and the empirical time complexity showed superior efficiency. Figure 14 shows the results of our algorithm on ami33 and n100 and provides the runtime analysis. Moreover, we sample 1000 points of objective function during global floorplanning, as shown in Figure 15. It can be seen, after the peak caused by the soft modules' enlargement at the beginning of each phase, the objective function converges to a stable value, which indicates our global floorplanning engine is able to reach to convergence.[2]

The second experiment was conducted on one CAD contest benchmark suite [3] designated for fixed-outline floorplanning with rectilinear soft modules. The benchmark statistics is listed in Table 2, where the fixed modules F include I/O pads, preplaced modules, and blockages, making the chip boundaries of some cases very irregular. We compared our algorithm with the contest winning team; the binary was provided by the team and executed on our platform for fair comparison. We observed that their floorplanning results contain "spikes", violating the minimum inner width constraint. Therefore, we also compared with their HPWL results after trimming these spikes. It is evident that our algorithm outperforms the two versions of the 1st place team in terms of HPWL by 2.55% and 5.95% on average. Particularly noteworthy is the performance on case09, where our algorithm significantly reduces HPWL by 10.50% and 16.66%. The 1st place team took 25 minutes to converge to a high-quality floorplan for each case; in contrast, our algorithm can generate comparable or even better floorplans within 0.75 seconds for all cases. Figure 16 shows the results of our algorithm on case02. The results reveal that our global floorplanning stage indeed finds decent relative positions for all modules, and our legalization and refinement schemes can form rectilinear shapes naturally for further wirelength reduction.

[2]Smooth wirelength models can be incorporated into our global floorplanning stage. We experimented on two smooth wirelength models: stable weighted average model [14] and BiG [34] with CHKS [8] as the basis model, using the MCNC benchmark. Our findings show that the average wirelength increased by 1.9% and 0.8% respectively compared to our wirelength model.

8 CONCLUSION

In this paper, we present a novel analytical-based floorplanning algorithm to gain the benefits of rectilinear soft modules, which generates floorplans with better space utilization and wirelength reduction. Our differentiable shape scheme enables fast gradient calculation and reflects the impact of dimension change on wirelength. The incremental module area growth allows us to focus on wirelength optimization at the beginning and then gradually add on the consideration of legality. Via shortest-path-based area migration, our legalization can remove overlaps while maintaining the relative positions derived by global floorplanning. For refinement, we devise simple and efficient operations to iteratively alter the module contour towards wirelength advantageous locations. Experimental results demonstrate the superior effectiveness and efficiency of our algorithm, outperforming the state-of-the-art works and contest winning team in terms of wirelength and runtime. Future work includes the extension to multiply initiated blocks handling, floorplanning in heterogeneous integration systems, and GPU acceleration.

REFERENCES

[1] 2007. *GSRC Benchmark*. Retrieved May 26, 2007 from http://vlsicad.eecs.umich.edu/BK/GSRCbench/

[2] 2007. *MCNC Benchmark*. Retrieved May 26, 2007 from http://vlsicad.eecs.umich.edu/BK/MCNCbench/

[3] 2023. *CAD Contest Benchmark Suite on Fixed-Outline Floorplanning with Rectilinear Soft Blocks*. Retrieved December 04, 2023 from https://drive.google.com/file/d/1oFAoYKMvhQwvJ0MH4R1yxnDWcxxarsNv/view?usp=sharing

[4] Saurabh N. Adya and Igor L. Markov. 2003. Fixed-outline floorplanning: enabling hierarchical design. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)* 11, 6 (2003), 1120–1135. https://doi.org/10.1109/TVLSI.2003.817546

[5] Hayward H. Chan, Saurabh N. Adya, and Igor L. Markov. 2005. Are floorplan representations important in digital design?. In *Proceedings of International Symposium on Physical Design (ISPD)*. 129–136. https://doi.org/10.1145/1055137.1055164

[6] Fu-Chieh Chang, Yu-Wei Tseng, Ya-Wen Yu, Ssu-Rui Lee, Alexandru Cioba, I-Lun Tseng, Da-shan Shiu, Jhih-Wei Hsu, Cheng-Yuan Wang, Chien-Yi Yang, Ren-Chu Wang, Yao-Wen Chang, Tai-Chen Chen, and Tung-Chieh Chen. 2022. Flexible chip placement via reinforcement learning: late breaking results. In *Proceedings of Design Automation Conference (DAC)*. 1392–1393. https://doi.org/10.1145/3489517.3530617

[7] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. 2000. B*-trees: a new representation for non-slicing floorplans. In *Proceedings of Design Automation Conference (DAC)*. 458–463. https://doi.org/10.1145/337292.337541

[8] Bintong Chen and Patrick T Harker. 1993. A non-interior-point continuation method for linear complementarity problems. *SIAM J. Matrix Anal. Appl.* 14, 4 (1993), 1168–1190. https://doi.org/10.1137/0614081

[9] E. W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1 (December 1959), 269–271. https://doi.org/10.1007/BF01386390

[10] Yan Feng, Dinesh P. Mehta, and Hannah Yang. 2004. Constrained floorplanning using network flows. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 23, 4 (2004), 572–580. https://doi.org/10.1109/TCAD.2004.825877

[11] Linley Gwennap. 2022. MediaTek Delivers Efficient Cortex-X2. Retrieved August 1, 2022 from https://www.eetasia.com/express/mediatek-delivers-efficient-cortex-x2/

[12] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics (TSSC)* 4, 2 (1968), 100–107. https://doi.org/10.1109/TSSC.1968.300136

[13] Zhuolun He, Yuzhe Ma, Lu Zhang, Peiyu Liao, Ngai Wong, Bei Yu, and Martin D.F. Wong. 2020. Learn to Floorplan through Acquisition of Effective Local Search Heuristics. In *Proceedings of International Conference on Computer Design (ICCD)*. 324–331. https://doi.org/10.1109/ICCD50377.2020.00061

[14] Meng-Kai Hsu, Valeriy Balabanov, and Yao-Wen Chang. 2013. TSV-aware analytical placement for 3-D IC designs based on a novel weighted-average wirelength model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 32, 4 (2013), 497–509. https://doi.org/10.1109/TCAD.2012.2226584

[15] Meng-Kai Hsu, Yao-Wen Chang, and Valeriy Balabanov. 2011. TSV-aware analytical placement for 3D IC designs. In *Proceedings of Design Automation Conference (DAC)*. 664–669. https://doi.org/10.1145/2024724.2024875

[16] Fuxing Huang, Duanxiang Liu, Xingquan Li, Bei Yu, and Wenxing Zhu. 2023. Handling Orientation and Aspect Ratio of Modules in Electrostatics-Based Large Scale Fixed-Outline Floorplanning. In *International Conference on Computer Aided Design (ICCAD)*. 1–9. https://doi.org/10.1109/ICCAD57390.2023.10323841

[17] Andrew B. Kahng. 2000. Classical floorplanning harmful?. In *Proceedings of International Symposium on Physical Design (ISPD)*. 207–213. https://doi.org/10.1145/332357.332401

[18] Andrew B. Kahng, Ravi Varadarajan, and Zhiang Wang. 2022. RTL-MP: Toward Practical, Human-Quality Chip Planning and Macro Placement. In *Proceedings of International Symposium on Physical Design (ISPD)*. 3–11. https://doi.org/10.1145/3505170.3506731

[19] Shixiong Kai, Chak-Wa Pui, Fangzhou Wang, Shougao Jiang, Bin Wang, Yu Huang, and Jianye Hao. 2023. TOFU: A Two-Step Floorplan Refinement Framework for Whitespace Reduction. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1–5. https://doi.org/10.23919/DATE56975.2023.10137175

[20] Inyup Kang. 2022. The Art of Scaling: Distributed and Connected to Sustain the Golden Age of Computation. In *Proceedings of International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 25–31. https://doi.org/10.1109/ISSCC42614.2022.9731536

[21] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

[22] S. Kirkpatrick, C.D. Jr. Gelatt, and M.P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (May 1983), 671–680. https://doi.org/10.1126/science.220.4598.671

[23] Jen-Wei Lee, Yi-Ying Liao, Te-Wei Chen, Yu-Hsiu Lin, Chia-Wei Chen, Chun-Ku Ting, Sheng-Tai Tseng, Ronald Kuo-Hua Ho, Hsin-Chuan Kuo, Chun-Chieh Wang, Ming-Fang Tsai, Chun-Chih Yang, Tai-Lai Tung, and Da-Shan Shiu. 2022. A Learning-Based Algorithm for Early Floorplan With Flexible Blocks. In *Proceedings of Asian Solid-State Circuits Conference (A-SSCC)*. 1–2. https://doi.org/10.1109/A-SSCC56115.2022.9980637

[24] Tsu-chang Lee. 1993. A bounded 2D contour searching algorithm for floorplan design with arbitrarily shaped rectilinear and soft modules. In *Proceedings of Design Automation Conference (DAC)*. 525–530. https://doi.org/10.1145/157485.165014

[25] Wei Li, Fangzhou Wang, José M. F. Moura, and R. D. Blanton. 2023. Global Floorplanning via Semidefinite Programming. In *Proceedings of Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC56929.2023.10247967

[26] Ximeng Li, Keyu Peng, Fuxing Huang, and Wenxing Zhu. 2023. PeF: Poisson's Equation-Based Large-Scale Fixed-Outline Floorplanning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 42, 6 (2023), 2002–2015. https://doi.org/10.1109/TCAD.2022.3213609

[27] Jai-Ming Lin and Yao-Wen Chang. 2001. TCG: a transitive closure graph-based representation for non-slicing floorplans. In *Proceedings of Design Automation Conference (DAC)*. 764–769. https://doi.org/10.1145/378239.379062

[28] Jai-Ming Lin and Hsi Hung. 2010. UFO: Unified convex optimization algorithms for fixed-outline floorplanning. In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*. 555–560. https://doi.org/10.1109/ASPDAC.2010.5419821

[29] Jai-Ming Lin and Zhi-Xiong Hung. 2011. UFO: Unified Convex Optimization Algorithms for Fixed-Outline Floorplanning Considering Pre-Placed Modules. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 30, 7 (2011), 1034–1044. https://doi.org/10.1109/TCAD.2011.2114531

[30] Chaomin Luo, Miguel F. Anjos, and Anthony Vannelli. 2008. Large-scale fixed-outline floorplanning design using convex optimization techniques. In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*. 198–203. https://doi.org/10.1109/ASPDAC.2008.4483939

[31] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. 2021. A graph placement methodology for fast chip design. *Nature* 594 (June 2021), 207–212. https://doi.org/10.1038/s41586-021-03544-w

[32] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake, and Yoji Kajitani. 1995. Rectangle-packing-based module placement. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*. 472–479. https://doi.org/10.1109/ICCAD.1995.480159

[33] William C. Naylor, Ross Donelly, and Lu Sha. 2001. Non-linear optimization system and method for wire length and dealy optimization for an automatic electric circuit placer. US Patent No. 6,301,693, Filed Dec. 16., 1998, Issued Oct. 9., 2001.

[34] Fan-Keng Sun and Yao-Wen Chang. 2019. BiG: A bivariate gradient-based wirelength model for analytical circuit placement. In *Proceedings of Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/3316781.3317782

[35] Xiaoping Tang and Martin D.F. Wong. 2004. On handling arbitrary rectilinear shape constraint. In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*. 38–41. https://doi.org/10.1109/ASPDAC.2004.1337536

[36] Laung-Terng Wang, Kwang-Ting (Tim) Cheng, and Yao-Wen Chang. 2009. *Electronic Design Automation: Synthesis, Verification, and Test*. Morgan Kaufmann, Burlington, MA, USA.

[37] Qi Xu, Hao Geng, Song Chen, Bo Yuan, Cheng Zhuo, Yi Kang, and Xiaoqing Wen. 2022. GoodFloorplan: Graph Convolutional Network and Reinforcement Learning-Based Floorplanning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 41, 10 (2022), 3492–3502. https://doi.org/10.1109/TCAD.2021.3131550