Handling Orientation and Aspect Ratio of Modules in Electrostatics-based Large Scale Fixed-Outline Floorplanning

Fuxing Huang[1], Duanxiang Liu[1] Xingquan Li[2], Bei Yu[3], Wenxing Zhu[1*]
[1]Fuzhou University, [2]Peng Cheng Laboratory, [3]The Chinese University of Hong Kong

***Abstract*—In this paper, we present an improved electrostatics-based analytical method for fixed-outline floorplanning, which incorporates module rotation and sizing driven by wirelength. To accurately compute the density function after module rotation, we propose a novel density calculation algorithm based on line drawing and polygon clipping algorithms commonly used in computer graphics. By using this algorithm, we are able to accurately compute the density function after module rotation without adding any complexity. Moreover, we propose a module legalization algorithm by adding module sizing and module rotation after the existing constraint graph adjustment step. Furthermore, we adopt a linear programming to minimize wirelength to improve the quality of the results. Experimental results demonstrate that our floorplanning algorithm achieves at least 5.9% and 11% reduction in half-perimeter wirelength on the HB+ and ami49_x benchmarks, respectively, compared to state-of-the-art floorplanners.**

***Index Terms*—Fixed-outline floorplanning, Global floorplanning, Module rotation, Module sizing, Legalization, Constraint graph.**

I. Introduction

Floorplanning is an essential component of the VLSI physical design process, requiring the organization of a number of circuit modules within a fixed-outline. However, this task is particularly challenging given the current utilization of IP cores and the large number of cells in modern chips. It requires careful consideration of several complex considerations, including thermal and power issues, timing, and voltage-island concerns. Despite its importance, floorplanning is a known NP-hard problem, making designing an efficient algorithm a significant challenge. A state-of-the-art floorplanner called PeF [1], which utilizes Poisson's equation, has shown positive results. However, it has a limitation since it neglects module orientation. In this paper, we aim to address this issue by employing analytical methods driven by wirelength optimization to handle orientation and aspect ratio of modules.

A. Previous Work

Fixed-outline floorplanning techniques can be divided into two categories: heuristic algorithms and analytical methods.

Heuristic algorithms are commonly utilized to address the floorplanning problem, which generally begins by selecting a floorplan representation, such as slicing tree [2]. The algorithms use heuristics, e.g., simulated annealing, to search for a solution within the chosen representation with the ultimate objective of finding an optimal solution. Then the solution is subsequently decoded into an actual floorplan. To overcome efficiency challenges when dealing with large-scale benchmarks, heuristic algorithms are often combined with hierarchical frameworks. These frameworks adopt partitioning or clustering to tackle the problem size, e.g., Capo [3] and DeFer [4].

Capo [3] uses a min-cut partitioner to divide the original circuit into multiple partitions. At the same time, the chip region is partitioned into smaller bins. Parquet [5] is then used to perform fixed-outline floorplanning for each partition along with its corresponding bin. DeFer [4] begins by using partitioner to generate a slicing tree [2]. The tree is designed to have a maximum of ten modules per leaf node. For each such node, a dynamic programming approach is used to derive its shape curve via enumerated packing process, which is then merged from the bottom to up. Finally, using fixed-outline as a constraint, the optimal shape curve of the root is obtained and the top-down program is run to determine the position and shape of each module.

Floorplanning algorithms based on heuristics efficiently handle orientations and aspect ratios of modules. DeFer [4] is an example that the orientations and aspect ratios of modules are used to optimize the shapes of nodes in the slicing tree [2], via the shape curve. In addition, the simulated annealing algorithm used in Capo [3] expands the solution space by incorporating modules' orientations and aspect ratios, which allows for better solution exploration.

However, analytical-based floorplanning methods have not consider both the orientations and aspect ratios of modules, due to their difficulty in modeling. In general, analytical-based floorplanning consists of two distinct stages. The first stage is the global floorplanning stage that permits partial overlapping of modules and calculates their optimal positions by optimizing an objective function. The second stage, called the legalization stage, aims to eliminate overlap and achieve a legitimate floorplanning result.

Recently, analytical-based algorithms have been introduced for fixed-outline floorplanning, such as F-FM [6], [7], and PeF [1]. F-FM [6] addresses voltage drop constraints, and [7] focuses on temperature constraints. In their global floorplanning stage, both algorithms consider their constraints in the NTUplace [8] framework. After global floorplanning, F-FM divides modules into a slicing tree, balances the number and area of the partitions, and merges shape curves with DeFer-like [4] method to acquire a feasible solution. Conversely, the legalization stage of [7] uses SAINT [9], which models the legalization problem as an ILP. PeF [1] proposes an electrostatic model for density control in the global floorplanning stage, which was originally introduced in ePlace [10]. During the legalization stage, PeF develops a constraint graph-based legalization algorithm and utilizes constraint graph adjustment to obtain a legal floorplan.

Neither F-FM [6] nor [7] optimizes the orientation and aspect ratio of modules during the global floorplanning stage. Although PeF [1] optimizes the module aspect ratio by width adjustment in global floorplanning, it uses a density-driven approach instead of a wirelength-driven approach. Besides, PeF [1] does not consider the optimization of module orientation.

B. Our Contributions

Analytical-based algorithms are known to be the most effective for floorplanning, but have limitation in handling module orientation and module's aspect ratio. Even the most advanced analytical floorplanners, including PeF [1], tend to neglect module orientation despite its significant potential in reducing wirelength. Therefore, the aim of this paper is to address the challenge of handling module orientation and aspect ratio in analytical-based floorplanning. The main contributions of this paper are summarized as follows:

- To handle the orientations and aspect ratios of modules in global floorplanning, we introduce two new variables: module rotation degree and soft module width. We use an analytical method to optimize these two variables, which are driven by wirelength. This allows the modules to be placed in desired orientations and aspect ratios by rotation and sizing.

This work was supported by the National Natural Science Foundation of China under Grant 62174033, and the Major Key Project of PCL (No. PCL2023AS2-3).
[*]Corresponding author: wxzhu@fzu.edu.cn

- After module rotation, we propose a novel density calculation algorithm that accurately calculates the density function of the rotated modules without increasing the complexity.
- We propose a legalization algorithm based on transitive reduction graph. To improve the robustness of our legalization algorithm, we add module sizing and module rotation after the existing constraint graph adjustment step. Finally, we further enhance the solution quality by minimizing the wirelength using a linear programming.
- Our algorithm achieves at least 5.9% and 11% reduction in half-perimeter wirelength on the HB+ and ami49_x benchmarks, respectively, compared to the state-of-the-art floorplanners, while maintaining an acceptable runtime.

The remainder of this paper are organized into the following sections. In Section II, we introduce the problem of fixed-outline floorplanning and the necessary background knowledge. In Section III, we propose our global floorplanning algorithm and legalization algorithm. In Section IV, we present experimental results and comparisons. Finally, we conclude and summarize our work in Section V.

II. Preliminaries

This section begins with an introduction of the fixed-outline floorplanning problem, followed by presenting the model for global floorplanning and the corresponding floorplan representation for legalization.

A. Fixed-outline Floorplanning Problem

Let V be the set of rectangular modules that consists of two subsets, V_h and V_s, representing the set of hard and soft modules, respectively. Each module v_i has its center coordinate (x_i, y_i), dimension (w_i, h_i), and area $A_i = w_i \times h_i$. Let θ_i denote the degree of counter-clockwise rotation of the module v_i around its center coordinate, while each module has four legal orientations with their corresponding θ_i of $0°(360°)$, $90°$, $180°$, and $270°$, respectively. The hard modules in V_h have fixed dimensions, while the soft modules in V_s can be adjusted with fixed area, subject to aspect ratio constraints, for which $AR_i = \frac{h_i}{w_i}$, and AR_i^l and AR_i^u represent the lower and upper bounds on a soft module's aspect ratio. Additionally, a set of nets, $Net = \{net_1, net_2, \ldots, net_m\}$, in which each net net_j interconnects multiple modules. Finally, $WL(V)$ represents the total wirelength of the netlist. The fixed-outline floorplanning problem aims to place all modules in a rectangular fixed-outline, R, with width W and height H. The modules must not overlap each other, and the goal is to minimize the total wirelength, $WL(V)$, among the modules.

B. Global Floorplanning Model

The rectangular fixed-outline R is divided into $K \times K$ equal-sized bins, denoted by B, and denote the bin in the j-th column and k-th row by $b_{j,k}$. The width and height of each bin are $w_b = \frac{W}{K}$ and $h_b = \frac{H}{K}$ respectively. Let ρ be the density function and $\rho_{j,k}$ be the density of bin $b_{j,k}$. The formulation of the global floorplanning problem can be described as a constrained minimization problem, which is presented as:

$$\begin{aligned} \min \quad & WL(V) \\ s.t. \quad & overflow \leqslant t_{of}, \end{aligned}$$

where $overflow$ is used to approximate the overlap among modules, which can be computed by $overflow = \sum_{j,k=0}^{K-1} \max(\rho_{j,k} - 1, 0)/K^2$ and t_{of} is a user-specified target overflow value.

The total wirelength, $WL(V)$, is commonly calculated using the half-perimeter wirelength (HPWL) in floorplanning. Floorplanning based on analytical methods uses gradient descent to optimize the floorplan. However, the HPWL function is non-differentiable. Therefore, a differentiable wirelength function, such as the log-sum-exp (LSE) wirelength [11], is required to approximate the HPWL function.

Electrostatic-based density control model is currently the mainstream approach applied in floorplanning, such as PeF [1], and in placement such as ePlace [10] and Pplace [12]. Pplace defines a new density function and obtains the analytical solution of the Poisson's equation. In [12], the density function ρ is defined as:

$$\rho_{j,k} = \sum_{v_i \in V} \frac{Area(b_{j,k} \cap v_i)}{w_b \times h_b}, \forall \rho_{j,k} \in \rho,$$

where $Area(b_{j,k} \cap v_i)$ is the intersection area between bin $b_{j,k}$ and module v_i. Subsequently, the discrete solution of the Poisson's equation is obtained, and based on the result, the electrostatic potential $\psi(j,k)$ of the bin $b_{j,k}$ is calculated according to the following formula:

$$\psi(j,k) = \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} a_{u,v} \cos\left(\frac{u(j+\frac{1}{2})\pi}{K}\right) \cos\left(\frac{v(k+\frac{1}{2})\pi}{K}\right).$$

Here, the discrete cosine transform coefficient $a_{u,v}$ is described in Pplace [12]. The electrostatic potential energy of the individual module v_i and the total electrostatic potential energy of all modules V are defined by:

$$N(v_i) \approx \sum_{b_{j,k} \in B} Area(b_{j,k} \cap v_i)\psi(j,k),$$

$$N(V) \approx \sum_{v_i \in V} N(v_i).$$

The reduction of module overlap is thus translated into the minimization of the electrostatic potential energy. Therefore, the problem of global floorplanning expressed as:

$$\begin{aligned} \min \quad & LSE(V) + \lambda N(V) \\ s.t. \quad & AR_i^l \leqslant AR_i \leqslant AR_i^u, \quad \forall v_i \in V_s \\ & \frac{w_i}{2} \leqslant x_i \leqslant W - \frac{w_i}{2}, \quad \forall v_i \in V \\ & \frac{h_i}{2} \leqslant y_i \leqslant H - \frac{h_i}{2}, \quad \forall v_i \in V, \end{aligned} \tag{1}$$

where λ is a penalty factor, which is updated in the same way as in ePlace [10].

C. Floorplan Representation for Legalization

In this paper, we use a horizontal constraint graph $G_h = (V, E_h)$ and a vertical constraint graph $G_v = (V, E_v)$ to represent a floorplan. Both graphs are directed acyclic graphs (DAGs). The construction rules for the graphs are similar to those in XDP [13]. We define the following symbols: Let module v_i be represented as vertex v_{h_i} in G_h. Each edge $e_{i,j}^h \in E_h$ represents a connection from v_{h_i} to v_{h_j}, i.e., v_i is on the left of v_j. We use $w(e_{i,j}^h) = \frac{w_i + w_j}{2}$ to denote the weight of edge $e_{i,j}^h$. We set dummy nodes v_{h_s} and v_{h_t} to represent the source and sink of G_h, respectively, i.e., the leftmost and rightmost points. Similar to XDP [13], we use the following attributes:

$$\begin{aligned} & L(v_{h_s}) = 0, \\ & L(v_{h_j}) = \max_i(L(v_{h_i}) + w(e_{i,j}^h)), \\ & R(v_{h_t}) = \max(L(v_{h_t}), W), \\ & R(v_{h_i}) = \max_j(R(v_{h_j}) - w(e_{i,j}^h)), \\ & slack(e_{i,j}^h) = R(v_{h_j}) - L(v_{h_i}) - w(e_{i,j}^h). \end{aligned} \tag{2}$$

For each node v_{h_i} in G_h, we set two attributes $L(v_{h_i})$ and $R(v_{h_i})$ to represent the leftmost and rightmost positions that v_i can reach, respectively. For each edge $e_{i,j}^h$, we set the slack $slack(e_{i,j}^h)$ to represent the maximum acceptable distance between v_{h_i} and v_{h_j}. The symbols and attributes in G_v are defined in the same way and are not repeated here.

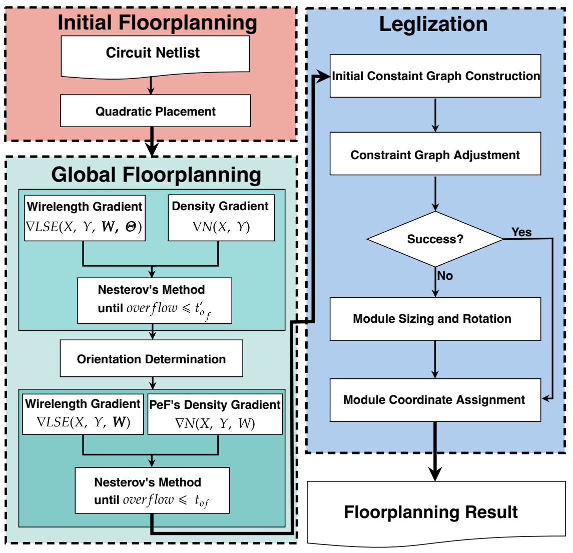
Initial Floorplanning
Circuit Netlist
Quadratic Placement
Global Floorplanning
Wirelength Gradient $\nabla LSE(X, Y, \boldsymbol{W}, \boldsymbol{\Theta})$
Density Gradient $\nabla N(X, Y)$
Nesterov's Method until $overflow \leqslant t'_{of}$
Orientation Determination
Wirelength Gradient $\nabla LSE(X, Y, \boldsymbol{W})$
PeF's Density Gradient $\nabla N(X, Y, W)$
Nesterov's Method until $overflow \leqslant t_{of}$
Leglization
Initial Constaint Graph Construction
Constraint Graph Adjustment
Success?
Yes
No
Module Sizing and Rotation
Module Coordinate Assignment
Floorplanning Result

Fig. 1. Flowchart of our fixed-outline floorplanning algorithm.

III. Our Floorplanning Algorithm

This section presents our floorplanning algorithm that comprises three parts: initial floorplanning, global floorplanning, and legalization. The overall flowchart of the algorithm is depicted in Fig. 1. In the initial floorplanning stage, we input the circuit netlist and employ the Quadratic Placement method [14] to obtain an initial floorplan. We divide the global floorplanning stage into two steps, detailed in Subsection III-A. In the first step, the coordinates of modules are optimized by the wirelength gradient and density gradient, while the rotation degrees and widths of modules are optimized by wirelength gradient. After determining the orientations of modules, in the second step, we optimize the coordinates and widths of modules using the wirelength gradient and density gradient proposed by PeF [1], to further eliminate overlaps between modules. Our legalization algorithm is based on the XDP [13], detailed in Subsection III-B, using three steps of initial constraint graph construction, constraint graph adjustment, module sizing and rotation to obtain two legal constraint graphs. Finally, we solve a linear programming problem to determine the final position of each module.

Next, we provide a detailed description of the global floorplanning and legalization stages.

A. Global Floorplanning

During the global floorplanning stage, we consider both the wirelength and the density of the modules, and strive to minimize the increase in wirelength while eliminating overlap. Besides positions, the rotation degrees θ of all modules and widths w of soft modules are considered in the optimization process, resulting in a better solution than those obtained by considering only the modules coordinates. As shown in Fig. 2, the wirelength can be shorter while keeping the center positions of the modules unchanged.

We divide the global floorplanning into two steps. In the first step, due to the complexity of calculation, the gradient with respect to (w, θ) is calculated only for the wirelength, and (w, θ) is regarded as a constant in the calculation of density gradient. We also set the target overflow t'_{of} to 0.15 in the first step, because changing (w, θ) does not affect the increase in overlap too much in the early stage of optimization. After the first step, we determine and fix the orientations of modules through the Orientation Determination step. In the second step, the module coordinates and widths are optimized simultaneously by the density gradient proposed by PeF [1] and the wirelength gradient including our sizing force in Equation (5). During the gradient descent, when the coordinate or aspect ratio of a module violate the constraints in Equation (1), we use the method proposed by PeF [1] to project the coordinate or aspect ratio to the feasible region.

Below, we introduce our formula for calculating the rotation force and sizing force.

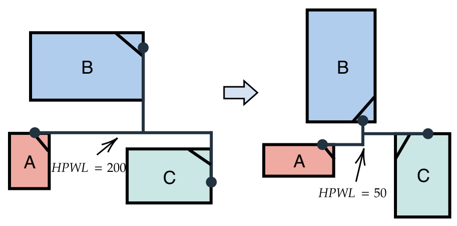

Fig. 2. Example of before and after optimizing module orientation and aspect ratio. Module A is a soft module, which is sized to the desired aspect ratio. Modules B and C are hard modules, which are rotated to the desired orientations.

1) Rotation Force and Sizing Force: Our rotation force and sizing force are defined as the gradient of the objective for optimization, i.e., wirelength, with respect to the rotation degree θ of modules and the width w of soft modules. To take partial derivatives of the wirelength with respect to (θ, w), we use the method proposed in [15]. In the field of placement, [15] is an excellent work for placing standard cells and macros simultaneously. Due to the differences between macros and standard cells, placing both simultaneously can lead to many complex problems. To solve this issue, [15] introduces a rotation force that allows a macro to rotate around its center point to obtain its desired orientation. The rotation force is the partial derivative with respect to θ obtained by the chain rule differentiation.

After rotating the module v_i by θ_i degrees, its pin_k position (x_k, y_k) formula is proposed by [15] as follows:

$$\begin{cases} x_k = x_i + x_{or} w_i \cos\theta_i - y_{or} h_i \sin\theta_i \\ y_k = y_i + x_{or} w_i \sin\theta_i + y_{or} h_i \cos\theta_i, \end{cases} \tag{3}$$

where x_{or} and y_{or} are the offset rates of pin_k from the center of v_i in the x and y directions.

The partial derivative of the wirelength model with respect to the direction θ_i of the module v_i, also known as the rotation force as described in [15], can be expressed by:

$$\begin{aligned} \frac{\partial LSE(V)}{\partial \theta_i} &= \frac{\partial LSE(V)}{\partial x_k} \cdot \frac{\partial x_k}{\partial \theta_i} + \frac{\partial LSE(V)}{\partial y_k} \cdot \frac{\partial y_k}{\partial \theta_i} \\ &= \frac{\partial LSE(V)}{\partial x_i} \cdot (-x_{or} w_i \sin\theta_i - y_{or} h_i \cos\theta_i) \\ &+ \frac{\partial LSE(V)}{\partial y_i} \cdot (x_{or} w_i \cos\theta_i - y_{or} h_i \sin\theta_i). \end{aligned} \tag{4}$$

Similarly, using the chain rule of differentiation, we obtain the sizing force, obtained by the partial derivative of the wirelength model with respect to the width w_i of soft module $v_i \in V_s$, as follows:

$$\begin{aligned} \frac{\partial LSE(V)}{\partial w_i} &= \frac{\partial LSE(V)}{\partial x_k} \cdot \frac{\partial x_k}{\partial w_i} + \frac{\partial LSE(V)}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_i} \\ &= \frac{\partial LSE(V)}{\partial x_i} \cdot (x_{or} \cos\theta_i + \frac{y_{or} A_i \sin\theta_i}{w_i^2}) \\ &+ \frac{\partial LSE(V)}{\partial y_i} \cdot (x_{or} \sin\theta_i - \frac{y_{or} A_i \cos\theta_i}{w_i^2}). \end{aligned} \tag{5}$$

As seen from Fig. 3, the modules are subject to rotation force and sizing force and tend to rotate and size to their desired orientations and aspect ratios.

With the computation of the rotation force and sizing force, we proceed to discuss the calculation of density force.

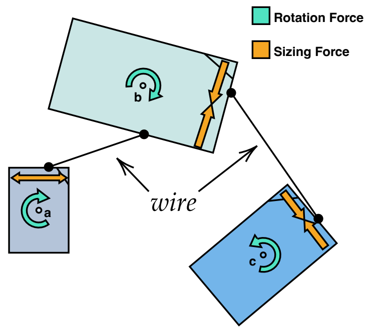

Fig. 3. Diagram of Rotation Force and Sizing Force.

2) Density Force: The electrostatics-based density model considers the density force on each module v_i in the x and y directions as its electric field force, represented as $(q_i\xi_{i_x}, q_i\xi_{i_y})$. The electric field force can be calculated using the following formula:

$$q_i\xi_{i_x} \approx \sum_{b_{j,k}\in B} Area(b_{j,k}\cap v_i)\xi_x(j,k),$$

$$q_i\xi_{i_y} \approx \sum_{b_{j,k}\in B} Area(b_{j,k}\cap v_i)\xi_y(j,k),$$

where $\xi_x(j,k)$ and $\xi_y(j,k)$ represent the electric field associated with each $b_{j,k}$ in the x and y directions, as described in detail in [10], [12].

Unlike the Cross Potential Model proposed in [15] to approximate the density force after module rotation, our density force formulas remain unchanged from those of other electrostatics-based density models. This is because our density calculation algorithm maintains the accuracy of the model. Obtaining the electric fields, $\xi_x(j,k)$ and $\xi_y(j,k)$, requires a triple Fast Fourier Transform (FFT) dependent on the density function ρ. Therefore, the first step is to calculate the density function ρ. Our proposed density calculation algorithm is outlined below.

When a module is at a legal orientation, i.e., $0^\circ(360^\circ)$, 90°, 180°, or 270°, i.e., the module is either horizontal or vertical, we can easily calculate the intersection bins $X_i = \{b_{j,k}|b_{j,k} \in (B\cap v_i)\}$ between all bins B and module v_i, and then calculate the intersection area $Area(b_{j,k}\cap v_i)$. However, when a module is free to rotate around its center, the form of intersection between the module and bins changes, as shown in Fig. 4.

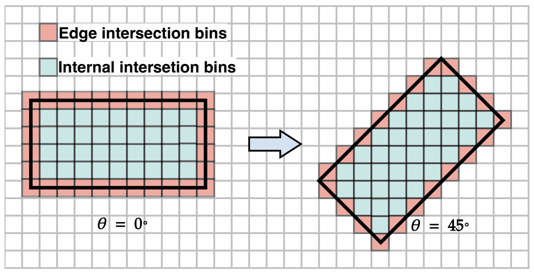

Fig. 4. Two forms of intersection between a module and bins.

To expedite the computation of the density function ρ after rotating each module, we initially determine the intersection bins X_i among module v_i and all bins B. For ease of downstream processing, we partition X_i into two kinds of bins: the set of edge intersection bins $E_i = \{(col, row), \dots\}$ and the set of internal intersection bins $I_i = \{(col, (row_{min}, row_{max})), \dots\}$, as illustrated in Fig. 4. The set of edge intersection bins E_i comprises bins partially overlapped by module v_i, namely, these bins intersecting with an edge of v_i. Each bin in the set of internal intersection bins I_i is wholly covered by module v_i. Specifically, E_i is a regular set, in which each element represents the position of a bin, i.e., the bin at row row-th and column col-th. On the other hand, I_i is a hash table, with col serving as the key and (row_{min}, row_{max}) as the values. Within the hash table I_i, col captures the bins located in the col-th column, while (row_{min}, row_{max}) records the upper and lower bounds of the bins in the column col-th intersecting module v_i, namely, the bins spanning rows row_{min}-th to row_{max}-th in the column col-th.

To obtain E_i, we use Bresenham's line drawing algorithm [16], which is an algorithm for finding the pixels that intersect a line on a bitmap and drawing the line accordingly. The algorithm uses only integer addition and subtraction in its computation, making it highly efficient, and has therefore been widely applied in computer graphics. By treating bins as pixels, we can use the Bresenham's line drawing algorithm to obtain E_i. After obtaining E_i of the module v_i using Bresenham's line drawing algorithm, I_i can be obtained from E_i as a subsequent step.

Algorithm 1 Module Drawing Algorithm

Input: module v_i; width and height w_b and h_b of bins.
Output: edge intersection bins E_i; internal intersection bins I_i.

```
1:  function ModuleDraw($v_i, w_b, h_b$)
2:      $Points \leftarrow$ EndPoints($v_i$);
3:      for $j = 0 \rightarrow 3$ do
4:          $k = (j+1) \mod 4$;
5:          $(x_j, y_j) \leftarrow Points[j]$;
6:          $(x_k, y_k) \leftarrow Points[k]$;
7:          $E_i \cup$ LineDraw($x_j, y_j, x_k, y_k, w_b, h_b$);
8:      for each $(col, row) \in E_i$ do
9:          if $I_i[col] = NULL$ then
10:             $I_i[col] = (row, row)$;
11:         else
12:             $(row_{min}, row_{max}) \leftarrow I_i[col]$;
13:             $row_{min} \leftarrow \min(row, row_{min})$;
14:             $row_{max} \leftarrow \max(row, row_{max})$;
15:             $I_i[col] = (row_{min}, row_{max})$;
16:     return $E_i, I_i$.
```

Algorithm 1 delineates this process, whereby we input module v_i along with the width and height of the bins to compute E_i and I_i. At line 2, in the counterclockwise order the four vertices of module v_i are computed, akin to how we determine the position of point pin through Equation (3). Next, from lines 3 to 7, the algorithm loops across the four edges and applies Bresenham's line drawing algorithm to each to yield E_i. Subsequently, from line 8, each element in E_i is traversed, and I_i is computed from E_i utilizing the property that I_i is enclosed by E_i. Within lines 9 to 15, the presence of the bin in the col-th column is checked in I_i. If absent, the bin at row row-th and column col-th is appended to I_i. Otherwise, the upper and lower bounds of the column col-th bin in I_i are updated to (row_{min}, row_{max}). Finally, the sets of edge intersection bins E_i and internal intersection bins I_i are returned.

Once we have the bins that intersect the module v_i, we calculate the intersection area between the module v_i and each bin, and accumulate it in the density function ρ. For the two forms of intersection bins, we use two different area calculation formulas. For the bin in E_i as shown in Fig. 5, which is equivalent to the intersection of two polygons, we use the Sutherland-Hodgman polygon clipping algorithm [17] to clip the clipping polygon. The area of the polygon is calculated using the following approach:

Given a polygon $ploy_i$ consisting of vertices ordered either clockwise or counterclockwise, i.e., $ploy_i = \{(x_1, y_1), \dots, (x_n, y_n)\}$, its area can

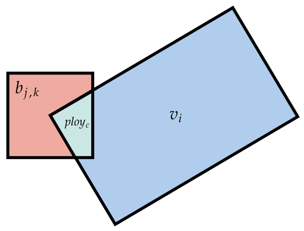

Fig. 5. Polygon clipping between module v_i and bin $b_{j,k}$, and clipping polygon $ploy_c$.

be expressed by Green's formula as:

$$Area(ploy_i) = \frac{1}{2}\sum_{i=1}^{n} |x_i y_{i+1} - x_{i+1} y_i| ,$$

where $x_{n+1} = x_1$, $y_{n+1} = y_1$ means that the first vertex and the last vertex are adjacent.

Each bin in I_i has an intersection area that equals its own area, since the module entirely surrounds it v_i. Based on this, we can derive the density computation algorithm shown in Algorithm 2.

Algorithm 2 Density Calculation

Input: uncalculated density function ρ; all modules V; width and height of bins w_b and h_b.

Output: calculated density function ρ.

```
 1: Initialize $\rho$ to 0;
 2: for each $v_i \in V$ do
 3:     $(E_i, I_i) \leftarrow ModuleDraw(v_i, w_b, h_b)$;
 4:     $Poly_v \leftarrow Polygon(v_i)$;
 5:     for each $(j,k) \in E_i$ do
 6:         $Poly_b \leftarrow Polygon(b_{j,k})$;
 7:         $Poly_c \leftarrow PolygonClip(Poly_v, Poly_b)$;
 8:         $\rho_{j,k} = \rho_{j,k} + PolygonArea(Poly_c)/(w_b \times h_b)$;
 9:         $Visit(j,k)$;
10:     for each $(col, (row_{min}, row_{max})) \in I_i$ do
11:         $j = col$;
12:         for $k = row_{min} + 1 \rightarrow row_{max} - 1$ do
13:             if $isVisit(j,k)$ then $continue$;
14:             $\rho_{j,k} = \rho_{j,k} + 1$;
15: return $\rho$.
```

Algorithm 2 takes as input the uncalculated density ρ, the set of all modules V, the width and height w_b and h_b of the bins. The output is the calculated density ρ. In line 1, the density is initialized by setting the density of each bin to 0. In line 2, we loop through all modules in V to compute the intersection area between each module and its intersection bins. In line 3 we call Algorithm 1 to get E_i and I_i. In line 4 we find the polygonal geometry of the module v_i and denote it as $Poly_v$. In lines 5-9, we loop through each bin in E_i, apply the polygon intersection algorithm to obtain the intersecting polygon $ploy_c$, calculate its area using Green's formula, and accumulate it into ρ. In lines 10-14, we calculate the overlapping area between each bin in I_i and module v_i, and accumulate it into ρ.

Now, let us analyze the complexity of our density calculation algorithm. For each module v_i, the time complexity of Bresenham's line drawing algorithm is determined by the number of elements in the set E_i, which is $O(|E_i|)$. Therefore, the time complexity of Algorithm 1 is also $O(|E_i|)$. Note that, the time complexity of Sutherland-Hodgman algorithm [17] is $O(nm)$, where n is the number of edges of the polygon and m is the number of edges of the clipping window. Since we are using a 4-edge polygon in our case, Sutherland-Hodgman algorithm is of constant complexity. Thus, we can conclude that the time complexity for calculating the density of each module v_i is $O(|v_i \cap B|)$. Therefore, the time complexity of our density calculation algorithm is $O(\sum_{v_i \in V} |v_i \cap B|) = O(K^2)$. As we need to maintain E_i and I_i, the space complexity is $O(avg(|E_i| + |I_i|)) \ll O(K^2)$. The time consumption in practical applications will be verified in the experiment section (Section IV). The density calculation algorithm can be used to calculate the density function ρ, while a similar method can be used to calculate the density force for each module, which will not be repeated here.

3) Orientation Determination: At the first step of the global floorplanning stage, we use a density calculation algorithm to allow modules to rotate at arbitrary rotation degrees driven by wirelength. However, legal module orientations are limited to four rotation degrees: $0°(360°)$, $90°$, $180°$, and $270°$. Therefore, at the end of the first step of global floorplanning, it is necessary to determine the legal placement orientations of modules based on their rotation degrees. To accomplish this, we adopt the macro orientation determination method from [15]. This method uses the proximity principle to rotate the macro to the nearest legal orientation relative to the current rotation degree. For certain special rotation degrees, such as $45°$, the proximity principle is not applicable. As such, [15] coped with this problem by solving an ILP to determine the orientations of the modules with the minimum overlap.

After the first step of global floorplanning, we fix the rotation degrees of all modules to their legal degrees, and proceed to the second step of global floorplanning. The second step is similar to the first step, except that the rotation degrees of modules are not optimized and the density gradient formula of PeF [1] is used, which is not repeated here.

B. Legalization

Our legalization algorithm is modified from the classic macro legalization and detailed placement algorithm XDP [13]. The speed of legalization is improved by using the more efficient transitive reduction algorithm GK [18] during the initial constraint graph construction step, and the more efficient min-cut-max-flow algorithm Dinic [19] during the constraint graph adjustment step. For enhancing robustness of the legalization process, we modify the edge capacity formula in the constraint graph adjustment step, and introduce the module sizing and rotation step. The quality of the solution after legalization is improved by a linear programming modified from [13] in the module coordinate assignment step.

1) Initial Constraint Graph Construction: We construct the G_h and G_v based on the results of the global floorplanning. Following the construction rules in [13], the total number of edges in the combined graph $E = E_h \cup E_v$ will be up to $O(|V|^2)$, i.e., $|E| = O(|V|^2)$, where many transitive edges are not needed but are shown as in Fig. 6(a).

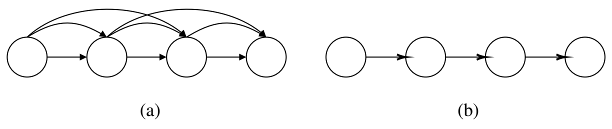

Fig. 6. (a) Transitive Closure Graph; (b) Transitive Reduction Graph.

When the constraint graph is adjusted, computing the constraint graph attributes, as specified in Equation (2), requires a topological sort with time complexity $O(|V| + |E|)$. In the subsequent module coordinate assignment step, we need to solve linear programming problems, which takes less time if the total number of edges can be decreased. Therefore, to reduce the time and space consumption, we need to remove transitive edges, as shown in Fig. 6(b).

The transitive reduction algorithm used in [13] is the depth first search. The time complexity for constructing the complete constraint graph using this algorithm is $O(|V|(|V|+|E|))$, where $O(|E|)=O(|V|^2)$, so using the depth first search for transitive reduction requires a time complexity of $O(|V|^3)$, which can be time-consuming for large $|V|$. Therefore, in this paper, we adopt the transitive reduction algorithm proposed in GK [18], whose time complexity is $O(|V||E^{tr}|)$, where $|E^{tr}|$ is the total number of edges after transitive reduction. Through our experiments, we found that $O(|E^{tr}|)=O(|V|\log(|V|))$, so using this algorithm, the time complexity for constructing the constraint graphs is reduced to $O(|V|^2\log(|V|))$.

2) Constraint Graph Adjustment: The purpose of this step is to adjust the constraint graphs to satisfy the fixed-outline constraint. Taking the horizontal constraint graph G_h as an example, after constructing the constraint graph G_h, we compute the attributes of each module (as shown in Equation (2)) to obtain its range of feasible positions after eliminating overlaps, i.e., $L(v_{h_i})\leqslant x_i\leqslant R(v_{h_i})$. At the same time, we get the longest path consisting of all edges with $slack=0$. If the length of the longest path exceeds the fixed-outline, i.e., $R(v_{h_t})>W$ for the sink node v_{h_t} of the constraint graph, it indicates that the constraint graph exceeds the fixed-outline and needs to be adjusted.

The adjustment process has three parts. First, we need to find the longest path subgraph G_c, i.e., the graph consisting of all edges $e^h_{i,j}$ with $slack(e^h_{i,j})=0$ and their nodes v_{h_i} and v_{h_j} in G_h. Then we compute the capacity of each edge in G_c and use the Dinic algorithm [19] to compute the min-cut-max-flow to get the min-cut set. Finally, we remove all edges $e^h_{i,j}$ in the min-cut set and construct the edges $e^v_{i,j}$ in G_v. If adjusting the edge $e^h_{i,j}$ causes the longest path of G_v exceeding the fixed-outline height, then we set the capacity of the edge as $+\infty$. For all edges connecting v_{h_s} and v_{h_t}, their capacities are also set to $+\infty$. The capacities of the remaining edges are computed as follows:

$$capacity_e(e^h_{i,j})=\max(L(v_{v_i})+\frac{h_i+h_j}{2}-L(v_{v_j}),0).$$

Note that our capacity calculation formula is somewhat different from that in [13]. The goal of [13] is to minimize the total displacement of all modules, while our goal is to select the edges that have the least impact on another constraint graph.

The step of constraint graph adjustment is iterated until the length of the longest path in both constraint graphs is less than the fixed-outline, indicating that we have obtained two legal constraint graphs. For further information on the constraint graph adjustment step, refer to the comprehensive introduction provided by [13].

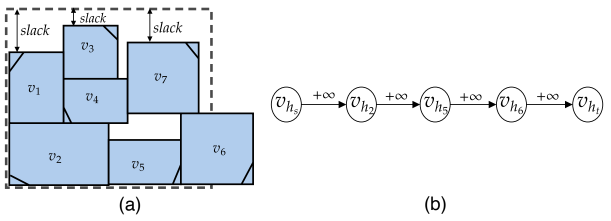

Fig. 7. (a) Floorplan that violates constraints; (b) The longest path subgraph G_c of G_h corresponding to (a).

When the global floorplanning result is particularly complex, both *slack* of the two constraint graphs are tight. As shown in Fig. 7, Fig. 7(a) represents the graph in which all the modules v_i are centered at the coordinate $(L(v_{h_i}),L(v_{v_i}))$ after eliminating the overlaps. It can be seen that the horizontal direction has exceeded the fixed-outline, which requires a constraint graph adjustment. However, the capacities of all edges in the longest path subgraph G_c of the horizontal constraint graph in Fig. 7(a) are set to $+\infty$, as shown in Fig. 7(b). This means that adjusting these edges will increase the longest path in the vertical constraint graph G_v, making it impossible for the min-cut-max-flow algorithm to select the min-cut set, resulting in a legalization failure. However, we found that many nodes in G_c still have some *slack* in the vertical direction. To fully utilize the remaining *slack* and obtain a legal result, we propose a new step called module sizing and rotation.

3) Module Sizing and Rotation: In floorplanning, any soft module can change its aspect ratio within a certain range with a fixed area. In addition, all modules can be rotated in any of the four orientations. Based on this premise, we propose the step of module sizing and rotation to make the constraint graph legal. To facilitate subsequent discussion, we introduce several attributes using horizontal constraint graph as an example.

- The slack of a module is defined as the range within which the module can move left or right without overlapping with other modules, and is calculated as follows:
$$slack(v_{h_i})=R(v_{h_i})-L(v_{h_i}).$$
- The sizing capacity of a soft module is defined as the maximum reduction in width that can be achieved while maintaining its area and without increasing the length of the longest path of another constraint graph. The sizing capacity of a soft module is calculated by:
$$capacity_s(v_{h_i})=w_i-\frac{A_i}{\min(slack(v_{v_i})+h_i,\sqrt{AR^u_i\cdot A_i})}.$$
For a soft module with slack 0 or aspect ratio that reaches its upper bound, its sizing capacity is set to 0.
- Since the aspect ratio of a hard module cannot be adjusted, we can achieve width-height swapping by rotating the module. Thus, we define the rotation capacity of a hard module as the maximum reduction in width that can be achieved after rotating the module, without increasing the length of the longest path of another constraint graph. For hard modules with slack 0 or widths which are smaller than their heights respectively, their rotation capacities are set to 0. The rotation capacities of the remaining hard modules are calculated as:
$$capacity_r(v_{h_i})=w_i-h_i.$$

The following Algorithm 3 describes the step of module sizing and rotation in the horizontal direction.

In Algorithm 3, we take as input two constraint graphs and the width and height of fixed-outline, and output a horizontal constraint graph that has been completed with module sizing and rotation. First, we compute the attributes of both constraint graphs using Equation (2). We then enter a loop that continues until the length of the longest path of the horizontal constraint graph is less than or equal to the width of the fixed-outline. At each iteration of the loop, we extract the longest path subgraph G_c from the horizontal constraint graph and compute the sizing capacity of all soft modules and the total sizing capacity. We then compute the amount of compression required and perform sizing on each soft module, distributing the compression amount using weighted averaging, with the total compression limited to the adjustable capacity $capacity_S$ (see lines 6-9). Next, we compute the remaining compression amount, and if it is less than or equal to zero, this indicates that G_c satisfies the constraints and no further hard module rotations are required (see lines 10-11). However, if there is still compression remaining, then hard module rotation is required (see lines 11-17). First, we calculate the rotation capacity for hard modules, and then we construct a min-heap based on the rotation capacity. We loop through the heap, taking the top element for rotation, which can be either clockwise or counterclockwise, until the compression amount is less than zero. Finally, after completing module sizing and hard module rotation, we update the attributes of the two constraint graphs (see line 18). Our strategy is performing module sizing first, followed by module rotation. This is mainly because

Algorithm 3 Module Sizing and Rotation

```
Input: constraint Graphs $(G_h, G_v)$, fixed-outline$(W, H)$.
Output: $G_h$ after module sizing and rotation.
 1: Calculate attributes according to Eq. (2);
 2: while $R(v_{h_t}) \le W$ do
 3:     Extract the longest path subgraph $G_c$ from $G_h$;
 4:     Calculate $capacity_s$ of all soft modules in $G_c$;
 5:     Calculate the sum of sizing capacities $capacity_S$;
 6:     $Press = R(v_{h_t}) - W$;
 7:     for each $v_i \in (V_s \cap G_c)$ do
 8:         $p(v_{h_i}) = \frac{capacity_s(v_{h_i})}{capacity_S} \min(Press, capacity_S)$;
 9:         Set the width and height of $v_i$ to $w_i - p(v_{h_i})$ and $\frac{A_i}{w_i - p(v_{h_i})}$;
10:     $Press = Press - capacity_S$;
11:     if $Press \le 0$ then $Continue$;
12:     Calculate $capacity_r$ of all hard modules in $G_c$;
13:     Build the min heap $heap$ according to the $capacity_r$;
14:     while $Press > 0$ do
15:         $v_i \leftarrow heap$;
16:         Rotate the module $v_i$ by $90° or -90°$;
17:         $Press = Press - capacity_r(v_{c_i})$;
18:     Calculate attributes according to Eq. (2);
19: return $G_h$.
```

module sizing is more controllable than module rotation, allowing us to precisely distribute the amount of adjustment needed to each module. Moreover, module rotation is more rigid, so we give priority to rotating modules with smaller rotation capacity.

4) Module Coordinate Assignment: Through the aforementioned steps, we obtain a legal horizontal constraint graph G_h and a legal vertical constraint graph G_v. In this step, similar to the approach in [13], we use a linear programming to assign the final position for each module. However, unlike [13], which minimizes the displacement of each module, we directly use linear programming to minimize wirelength presented in [20] to determine the final positions of the modules. The linear programming model is as follows:

$$\begin{aligned}
\min \quad & \sum_{net_k \in Net} (R_k - L_k + T_k - B_k) \\
s.t. \quad & L_k \le x'_i \le R_k \quad \forall v_i \in net_k, \forall net_k \in Net \\
& B_k \le y'_i \le T_k \quad \forall v_i \in net_k, \forall net_k \in Net \\
& x'_j - x'_i \ge \frac{w_i + w_j}{2} \quad \forall e^h_{i,j} \in E_h \\
& y'_j - y'_i \ge \frac{h_i + h_j}{2} \quad \forall e^v_{i,j} \in E_v \\
& \frac{w_i}{2} \le x'_i \le W - \frac{w_i}{2} \\
& \frac{h_i}{2} \le y'_i \le H - \frac{h_i}{2}.
\end{aligned}$$

In the above model, the variables R_k, L_k, T_k, and B_k represent the right, left, top, and bottom boundaries of the bounding box of the net net_k, respectively. Additionally, the variables x'_i and y'_i denote the center coordinates of the final position of module v_i. Moreover, the first two constraints ensure that the center position of each module is inside the bounding box of the net. For easy illustration and understanding, we assume that all pins are located at the center of the module. In practice, we use the real pin location. The third and fourth constraints ensure that the modules do not overlap. Finally, the fifth and sixth constraints ensure that all modules are placed within the fixed-outline. For the solution of this LP problem, we used CPLEX [21] optimization solver.

IV. Experimental Results

In this section, we present experimental results of our proposed floorplanning algorithm. To demonstrate the effectiveness, we compare it to state-of-the-art floorplanners on IBM-HB+ [22] and ami49_x [23] benchmarks. Our algorithm is implemented in C++ and operates in single-threaded mode on a Linux operating system. The machine used for the experiments has a 5.10GHz 12th Gen Intel Core CPU and 64GB memory. The evaluation metrics used in the experiments are the HPWL (half-perimeter wirelength) and the CPU time. In the experiments, we set the target overflow t_{of} to 0.05.

A. IBM-HB+ Benchmarks

The HB+ benchmarks were created by enlarging the largest hard modules in the HB benchmarks by 100% and decreasing the area of the remaining soft modules, while retaining the total area of modules. This complex design makes processing the HB+ benchmarks more challenging. The floorplanners compared on this benchmarks include DeFer [4], and the state-of-the-art PeF [1] which also uses the electrostatics-based density model. Unfortunately, we found that all pins of the modules of the HB+ benchmarks are located at the centers of the modules, i.e., $x_{or} = 0$ and $y_{or} = 0$. This implies that the partial derivatives of the wirelength with respect to θ (Equation (4)) and w (Equation (5)) are 0. Therefore, our wirelength-driven module rotation and sizing are ineffective for these benchmarks. However, it must be remarked that in practical chip design, the pins are usually not placed at the centers of macros, hence this situation does not really occur.

TABLE I
Results on the HB+ benchmarks

Basic Information		HPWL ($\times 10^6$)			time (s)		
Name	white-space	DeFer	PeF	ours	DeFer	PeF	ours
HB+01	26%	3.09	3.00	2.72	1.8	4.3	4.1
HB+02	25%	6.17	6.02	5.37	15.3	4.3	7.8
HB+03	30%	9.19	8.17	7.67	4.0	8.1	11.0
HB+04	25%	10.26	9.72	8.98	14.2	7.5	10.7
HB+06	25%	8.78	7.91	7.44	5.0	7.1	6.2
HB+07	25%	15.48	14.07	13.58	4.6	7.2	8.0
HB+08	26%	18.73	17.19	16.49	19.3	9.7	12.7
HB+09	25%	16.66	15.81	14.56	4.2	9.0	9.1
HB+10	20%	45.12	40.61	37.62	6.3	32.9	16.6
HB+11	25%	26.99	24.58	23.39	7.1	9.3	14.2
HB+12	26%	50.17	48.96	46.41	5.5	15.0	13.8
HB+13	25%	35.51	32.65	30.62	5.9	14.2	11.7
HB+14	25%	64.50	59.62	57.73	12.0	16.7	21.6
HB+15	25%	84.29	73.93	71.64	14.7	24.8	27.9
HB+16	25%	98.66	87.32	85.23	8.1	20.0	23.8
HB+17	25%	144.50	138.44	133.73	14.7	20.5	26.4
HB+18	25%	71.86	68.05	66.58	11.3	14.5	14.5
Ratio		1.143	1.059	1.000	0.706	0.944	1.000

Table I gives the experimental results of DeFer, PeF and our algorithm on the HB+ benchmarks. The first two columns show the basic information about each benchmark, including its name and whitespace rate. Columns 3 through 5 present the HPWL results of each floorplanner, in units of 10^6. Columns 6 through 9 give the CPU times of each floorplanner, in seconds. The last row presents the ratio, which is the average ratio of the HPWL (time) result of each benchmark by the respective floorplanners to our HPWL (time) result for the corresponding benchmark.

As can be seen from the table, our floorplanning algorithm achieves HPWL reductions of 14.3% and 5.9% compared to DeFer and PeF, respectively. Furthermore, the HPWL result of each benchmark by our algorithm is better than those of the other two floorplanners for the corresponding benchmark. Regarding the runtime comparison, due to different machines used by the floorplanners, we quote the runtimes directly from the respective references directly for a rough comparison. PeF was performed on a 3.60GHz Intel Core i3-9100 CPU with 4GB of memory and Defer used a Core Duo 1.86GHz CPU with 2GB of memory. Although our machine is the best, we can see that DeFer and PeF are faster than ours, taking only 70% and 94% of our runtime, respectively. In fact, they are actually much faster than ours because of their inferior machines.

B. *ami49_x Benchmarks*

The ami49_x benchmarks are currently the largest benchmarks for floorplanning. Therefore, we use the ami49_x benchmarks as described in [23] to evaluate the effectiveness of our floorplanning algorithm.

TABLE II
RESULTS ON THE AMI49_X BENCHMARKS, WHITESPACE 15%

	HPWL $(\times 10^6)$		time (min)	
Circuit Name	PeF	ours	PeF	ours
ami49_10	2.68	2.33	0.08	0.03
ami49_20	6.80	6.13	0.09	0.06
ami49_40	19.03	15.99	0.22	0.17
ami49_60	32.30	29.25	0.43	0.24
ami49_80	47.30	42.88	1.08	0.51
ami49_100	65.32	59.60	1.57	0.67
ami49_150	113.86	107.17	2.57	1.34
ami49_200	170.02	161.70	5.02	2.43
Ratio	1.11	1.00	1.90	1.00

On the ami49_x benchmarks, we compare our floorplanning algorithm with PeF [1], the experimental results are presented in Table II. The first column of Table II shows the circuit names of the benchmarks. Columns 2-3 give the wirelength results for each floorpanner, in units of 10^6. Columns 4-5 present the runtimes in minutes. The last row shows the ratio calculation, which is the same as in Table I.

From the table, we can see that our floorplanning algorithm reduces the HPWL result by 11% compared to PeF. In addition, the HPWL results of all benchmarks are smaller than the corresponding results of PeF. Regarding the runtime comparison, we have taken the runtimes directly from PeF's paper. It can be seen that PeF has longer runtime than ours. Because of different machines, this comparison is not fair, but may give an intuitive feeling of our algorithm's efficiency. Fig. 8 gives the floorplan of ami49_10 obtained by our floorplanning algorithm. It can be seen that all the modules have been rotated to their desired orientations.

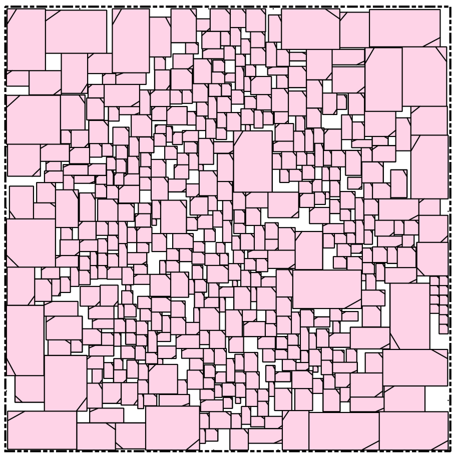

Fig. 8. Result of ami49_10 by our floorplanning algorithm (490 modules, 4521 nets, $HPWL = 2.33 \times 10^6$).

It is noteworthy that our HPWL is 11% less than PeF in this experiment, while this number is 5.9% on the HB+ benchmarks. We believe that this discrepancy is due to the difference between the ami49_x and the HB+ benchmarks. As mentioned earlier, in the HB+ benchmarks, the pins of the modules are all located at the centers of the modules, while in the ami49_x benchmarks, the pins of the modules are almost all located at the edges of the modules. This also indirectly demonstrates the effectiveness of our rotation and sizing.

C. *Effectiveness of Rotation and Sizing*

In this experiment, we aim to demonstrate the effectiveness of our module rotation and module sizing driven by wirelength. Since only the modules in the ami49_x benchmarks have pins that are not at the centers of the modules, we only test the performance of our rotation and sizing on these benchmarks. We compare the results between enabling and disabling module rotation and module sizing in our floorplanning algorithm.

TABLE III
RESULTING HPWL AND CPU TIMES OF OUR ALGORITHM WITHOUT AND WITH MODULE ROTATION AND MODULE SIZING ON THE AMI49_X BENCHMARK CIRCUITS

	Ours w/o R&S		Ours w/ R&S	
Circuit Name	HPWL$(\times 10^6)$	CPU(s)	HPWL$(\times 10^6)$	CPU(s)
ami49_10	2.52	1.6	2.33	1.9
ami49_20	6.48	3.9	6.13	4.2
ami49_40	16.75	9.9	15.99	10.6
ami49_60	30.18	14	28.25	14.7
ami49_80	44.97	28.6	42.88	30.6
ami49_100	61.28	38.9	59.60	40.4
ami49_150	112.28	77.6	107.17	80.5
ami49_200	165.27	143.8	161.70	145.9
Ratio	1.05	0.93	1.00	1.00

Table III presents the benchmark names in the first column, the wirelength (HPWL) and runtime of our algorithm without and with rotation and sizing are given in columns 2-3 and columns 4-5, respectively. The units for HPWL and runtime are 10^6 and seconds, respectively.

As shown in the table, enabling module rotation and module sizing in our algorithm reduces the HPWL by 5%, compared to that disabling module rotation and module sizing. In addition, the runtimes for the two experiments are almost the same, confirming that the time complexity of our density calculation Algorithm 2 is almost the same as that of the original density calculation in [12].

V. CONCLUSION

This paper has proposed an improved fixed-outline floorplanning algorithm based on electrostatics that incorporates module rotation and module sizing driven by wirelength. To accurately calculate the density function after module rotation, a novel density calculation algorithm was introduced based on line drawing algorithm and polygon clipping algorithm in computer graphics. With this algorithm, it is able to calculate the density after module rotation precisely without increasing complexity. A floorplan legalization algorithm has been designed with module sizing and module rotation after constraint graph adjustment. Moreover, a linear programming has been adopted to minimize wirelength and enhance the quality of the results. Experimental results demonstrate that our floorplanning algorithm reduces the average half-perimeter wirelength by 5.9% and 11% on the HB+ and ami49_x benchmark circuits, respectively, compared to state-of-the-art floorplanners. Although our two-step global floorplanning can solve the problem better, an ideal scheme is to use the wirelength gradient and density gradient to simultaneously optimize the coordinates, rotation degrees and widths of modules, which deserves further investigation.

References

[1] X. Li, K. Peng, F. Huang, and W. Zhu, "PeF: Poisson's equation based large-scale fixed-outline floorplanning," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 42, no. 6, pp. 2002–2015, 2023.

[2] D. Wong and C. Liu, "A new algorithm for floorplan design," in *Proc. Des. Autom. Conf. (DAC)*, 1986, pp. 101–107.

[3] S. Adya, S. Chaturvedi, J. Roy, D. Papa, and I. Markov, "Unification of partitioning, placement and floorplanning," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2004, pp. 550–557.

[4] J. Z. Yan and C. Chu, "DeFer: Deferred decision making enabled fixed-outline floorplanning algorithm," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 29, no. 3, pp. 367–381, 2010.

[5] S. Adya and I. Markov, "Fixed-outline floorplanning: enabling hierarchical design," *IEEE Trans. Very Large Scale Integr VLSI Syst.*, vol. 11, no. 6, pp. 1120–1135, 2003.

[6] J.-M. Lin and J.-H. Wu, "F-FM: Fixed-outline floorplanning methodology for mixed-size modules considering voltage-island constraint," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 33, no. 11, pp. 1681–1692, 2014.

[7] J.-M. Lin, T.-T. Chen, H.-Y. Hsieh, Y.-T. Shyu, Y.-J. Chang, and J.-M. Lu, "Thermal-aware fixed-outline floorplanning using analytical models with thermal-force modulation," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 29, no. 5, pp. 985–997, 2021.

[8] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1228–1240, 2008.

[9] J.-M. Lin, P.-Y. Chiu, and Y.-F. Chang, "SAINT: Handling module folding and alignment in fixed-outline floorplans for 3D ICs," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2016, pp. 1–7.

[10] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, "ePlace: Electrostatics-based placement using Fast Fourier Transform and Nesterov's method," *ACM Trans. Design Autom. Electron. Syst.*, vol. 20, no. 2, pp. 1–34, 2015.

[11] W. C. Naylor, R. Donelly, and L. Sha, "Nonlinear optimization system and method for wire length and delay optimization for an automatic electronic circuit placer," U.S. Patent 6 301 693, 2001.

[12] W. Zhu, Z. Huang, J. Chen, and Y.-W. Chang, "Analytical solution of Poisson's equation and its application to VLSI global placement," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.

[13] J. Cong and M. Xie, "A robust mixed-size legalization and detailed placement algorithm," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 27, no. 8, pp. 1349–1362, 2008.

[14] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 10, no. 3, pp. 356–365, 1991.

[15] M.-K. Hsu and Y.-W. Chang, "Unified analytical global placement for large-scale mixed-size circuit designs," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2010, pp. 657–662.

[16] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965.

[17] I. Sutherland and G. W. Hodgman, "Reentrant polygon clipping," *Communications of the ACM*, vol. 17, no. 1, pp. 32–42, 1974.

[18] A. Goralčíková and V. Koubek, "A reduct-and-closure algorithm for graphs," in *Proc. 8th Int. Symp. Math. Found. Comput. Sci.*, Olomouc, Czech Republic, 1979, pp. 301–307.

[19] Y. Dinitz, "Algorithm for solution of a problem of maximum flow in networks with power estimation," *Soviet Math. Dokl.*, vol. 11, pp. 1277–1280, 01 1970.

[20] X. Tang, R. Tian, and M. Wong, "Optimal redistribution of white space for wire length minimization," in *Proc. Asia South Pac. Des. Autom. Conf. (ASP-DAC)*, vol. 1, 2005, pp. 412–417.

[21] IBM Inc. IBM ILOG CPLEX. [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer

[22] A. N. Ng, I. L. Markov, R. Aggarwal, and V. Ramachandran, "Solving hard instances of floorplacement," in *Proc. Int. Symp. Phys. Des. (ISPD)*, 2006.

[23] T.-C. Chen, Y.-W. Chang, and S.-C. Lin, "A new multilevel framework for large-scale interconnect-driven floorplanning," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 27, no. 2, pp. 286–294, 2008.