POLAR: a high performance mixed-size wirelengh-driven placer with density constraints

Tao Lin*, Chris Chu*, Joseph R. Shinnerl†, Ismail Bustany†, Ivailo Nedelchev†
* Iowa State University
† Mentor Graphics Corporation

***Abstract*—Wirelength is one of the most important metrics in placement problem. Minimizing wirelength is not only a beneficial but also fundamental step to optimize other metrics, such as timing, power and routability. In this paper, we propose a high performance mixed-size wirelengh-driven placer called POLAR. POLAR is based on the recent popular look-ahead legalization idea. The goals of our look-ahead legalization are: (1) to achieve a roughly legalized placement; (2) to maintain cells' relative positions of quadratic placement while minimizing cell movements. To achieve these goals, in POLAR, look-ahead legalization is realized in a simple and elegant manner. Firstly, all placement density hotspots (where placement overflow occurs) are detected. Secondly, for each hotspot, an appropriate window is searched to cover it by enumerating many feasible candidates. Finally, cell-to-bin assignment is performed within each window by a fast recursive bisection method. The experimental results verify the efficiency of POLAR over the ISPD 2005 and ISPD 2006 benchmarks.**

Keywords*—placement, look-ahead legalization, placement density hotspot, window, recursive bisection.

I. Introduction

Placement is one of the most fundamental problems in electronic design automation (EDA). Although it has been extensively studied and its solution quality has been improved significantly during the last decade, a high performance placer is still in urgent need to catch up with the continual increase of design scale. Besides, considering varieties of new constraints and objectives introduced due to technology scaling, Alpert et al. [1] indicates that placement is still a "hot topic".

There are many metrics in placement optimization. Wirelength is one of the most important metrics, since minimizing wirelength is not only a beneficial but also fundamental step to optimize other metrics, such as timing, power and routability. Therefore, [2] points out that designing more efficient wirelength-driven placer is the key step to conquer new emerging challenges in placement problem.

To solve placement problem, analytical approach defines a suitable analytical cost function and minimizes the cost function through numerical optimization methods. It is considered the most promising technique. Depending on the cost function, analytical placers can be subdivided into the following two categories: (1) nonlinear placer; (2) quadratic placer.

Nonlinear placer approximates half perimeter wirelength (HPWL) by a nonlinear cost function, e.g., log-sum-exp function [3]. And the placement density constraints are smoothed by differentiable nonlinear function, e.g., bell-shaped function [3] and inverse Laplace transformation [4]. Because solving nonlinear programming is time consuming, nonlinear placers usually apply a multilevel approach [5] [6] to reduce runtime. Examples of nonlinear placers are APlace [7], mPL [4] and NTUPlace [8].

Different from nonlinear placer, quadratic placer approximates HPWL by a convex quadratic function, which is also called quadratic wirelength. Quadratic wirelength can be efficiently minimized by solving linear equations. However, minimizing just quadratic wirelength would lead to considerable cell overlapping. Therefore, many techniques have been proposed to spread out cells while maintaining quadratic nature of optimization.

Among cell spreading techniques for quadratic placer, iterative force-directed approach is the most promising one due to its low runtime and good placement quality. It interprets placement problem as a classical mechanics problem of finding equilibrium configuration for a spring system. In each placement iteration, the equilibrium state of the corresponding spring system is achieved by minimizing the quadratic wirelength. Then a cell spreading technique is applied to generate anchors for movable cells. Based on these anchors, additional spreading forces are added into spring system. This process gradually spreads out cells until the cell distribution is almost even and the wirelength is not improved any more. Examples of quadratic placers which apply force-directed approach are Kraftwerk2 [9], DPlace [10], mFAR [11], FastPlace [12], RQL [13], SimPL [14], ComPLx [15] and MAPLE [16].

The main difference among different force-directed quadratic placers is how they spread out movable cells to generate their anchors. Kraftwerk2 [9] and DPlace [10] are based on density gradient. Kraftwerk2 utilizes a Poisson potential by a generic supply and demand system, while DPlace models the diffusion process by solving a differential equation relating to cell density. mFAR [11] achieves the spreading forces by moving cells from those bins with overflow to those without. FastPlace [12] and RQL [13] move the cells from high density bins to the low density adjacent bins by cell shifting. Recently, SimPL [14] proposed a new cell spreading technique called look-ahead legalization. The key idea of look-ahead legalization is that almost legal placement is used to guide the anchor generation. Many placers [14–18] adopt this idea and produce high quality placements. In SimPL [14], the look-ahead legalization is implemented by top-down geometric partitioning and non-linear scaling. In ComPLx [15], the entire placement process is modelled by subgradient primal-

dual Lagrange optimization, whereas look-ahead legalization is modelled by a feasibility projection. In MAPLE [16], the look-ahead legalization of SimPL is combined with multilevel clustering [5] and improvement of iterative local refinement [12]. Besides, both SimPLR [17] and Ripple [18] extend SimPL to handle routing congestion.

In this paper, we propose a new force-directed quadratic placer called POLAR. POLAR also adopts the look-ahead legalization idea and the look-ahead legalization is realized in a simple and elegant manner. We notice that while the placement solution by quadratic based wirelength minimization may have considerable overlaps, the relative positions of cells can be trusted in producing a legal placement with good wirelength. Hence, during look-ahead legalization, our goal is to maintain relative positions of cells as best as we can while minimizing cell movements. To achieve this goal, firstly, all placement density hotspots are detected. Secondly, for each hotspot, an appropriate window (which is also called expansion region) is searched to cover it by enumerating many feasible candidates. Finally, cell-to-bin assignment is performed within each window by a fast recursive bisection method and then cells within each bin are spread out.

Comparing with other SimPL-like placers (which adopt look-ahead legalization such as [14–16], there are two main differences in POLAR's look-ahead legalization approach. The first one is how to find window for placement density hotspot. POLAR enumerates many feasible windows in order to maintain quadratic placement maximally. The second one is how to perform cell spreading in each window. POLAR formulates this step into cell-to-bin assignment so that it can have better control on the placement density of each bin. The experimental results over ISPD 2005 and ISPD 2006 benchmarks show that POLAR outperforms other SimPL-like wirelength-driven placers. Besides, POLAR's look-ahead legalization approach can be easily extended to consider routing congestion in [19], which outperforms all the other academic routability-driven placers both on runtime and quality over ICCAD 2012 routability-driven placement contest benchmarks [20] so far.

The rest of this paper is organized as follows. Section II presents the preliminary. Section III elaborates the POLAR's algorithm. Section IV presents some implementation details. Section V shows the experimental results. Finally, the conclusions are made in Section VI.

II. Preliminary

A circuit can be represented by a hypergraph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_{|V|}\}$ is the set of cells and $E = \{e_1, e_2, \ldots, e_{|E|}\}$ is the set of nets. Global placement tries to determine physical positions of cells without violating placement density constraints. We denote the x-coordinates of cells by a vector $\mathbf{x} = (x_1, x_2, \cdots, x_{|V|})$, and the y-coordinates by $\mathbf{y} = (y_1, y_2, \cdots, y_{|V|})$, the objective is to minimize the HPWL, which is measured by Formula (1).

$$\text{HPWL}(\mathbf{x}, \mathbf{y}) = \Sigma_{e \in E}[\max_{i \in e} x_i - \min_{i \in e} x_i + \max_{i \in e} y_i - \min_{i \in e} y_i] \quad (1)$$

A. Quadratic optimization

Assuming that all the nets only connect two different cells in the circuit. For any net, as shown in Formula (1), the HPWL is given by Manhattan distance between the two connected cells. In quadratic placer, the Manhattan distance is approximated by squared Euclidean distance of the two connected cells, so the cost function ϕ of global placement can be defined in Formula (2).

$$\phi = \frac{1}{2}\mathbf{x}^T Q_x \mathbf{x} + \mathbf{c}_\mathbf{x}^T \mathbf{x} + \frac{1}{2}\mathbf{y}^T Q_y \mathbf{y} + \mathbf{c}_\mathbf{y}^T \mathbf{y} + const \quad (2)$$

where the connection matrices Q_x and Q_y are both sparse symmetric positive definite. Minimizing ϕ is equal to solving the linear system in Eq. (3).

$$Q_x \mathbf{x} + \mathbf{c}_\mathbf{x} + Q_y \mathbf{y} + \mathbf{c}_\mathbf{y} = 0 \quad (3)$$

In POLAR, preconditioned conjugate gradient (PCG) method with incomplete Cholesky decomposition [21] is used to solve Eq. (3).

B. Bound-to-bound net model

In real circuit, lots of nets have more than two pins. To get the quadratic cost function (2), every multi-pin net should be decomposed into a set of 2-pin nets by a net model, e.g., clique model [22], hybrid model [23] or Bound-to-bound (B2B) model [9]. The net model determines the connection matrices Q_x and Q_y, which have big impact on the runtime and placement quality of quadratic placer. Therefore, it is important to choose suitable net model. In POLAR, we use B2B net model, which has been shown to both accurate and efficient in practice.

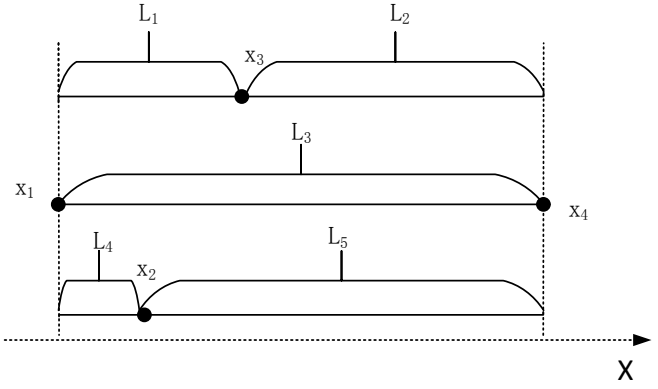


Fig. 1. B2B net model [9]. In this example, there are four pins in the given net. The left most pin is x_1 and the right most pin is x_4. So the quadratic wirelength of this net in x-direction is $w^x_{1,3}(x_1 - x_3)^2 + w^x_{3,4}(x_3 - x_4)^2 + w^x_{1,4}(x_1 - x_4)^2 + w^x_{1,2}(x_2 - x_1)^2 + w^x_{2,4}(x_2 - x_4)^2$, where $w^x_{1,3} = \frac{1}{2L_1}$, $w^x_{3,4} = \frac{1}{2L_2}$, $w^x_{1,4} = \frac{1}{2L_3}$, $w^x_{1,2} = \frac{1}{2L_4}$ and $w^x_{2,4} = \frac{1}{2L_5}$.

The B2B net model is based on the idea of removing all inner two-pin connections and utilizing only connections to the boundary pins. With this, the boundary pins span the net, and the property of the HPWL being the distance between the boundary pins is emulated. In x-direction, the two-pin

connection weight $w_{p,q}^x$ of B2B net model is determined in Formula (4) [9]. In y-direction, the weight is calculated in the similar way with x-direction. B2B net model should be updated once cells' positions are changed in each global placement iteration. Fig. 1 gives an example of B2B net model.

$$w_{p,q}^x = \begin{cases} 0 & \text{if pin } p \text{ and } q \text{ are inner pins} \\ \frac{2}{P-1}\frac{1}{|x_p - x_q|} & \text{otherwise} \end{cases} \quad (4)$$

where P is the number of pins in the net.

It is proved that (2) based on B2B net model is completely equal to (1) if the positions of cells are finally converged using B2B net model in [9].

C. Spreading force realization

To reduce cell overlapping, spreading forces are added to guide cells toward their anchors (some papers also call them target positions). [11] proposed a simple way (which is called fixed-point technique) to add spreading force by pseudo net connecting cell's original position to its anchor, as shown in Fig. 2. Then the connection matrices Q_x and Q_y are updated and linear system (3) is solved again. In POLAR, we also use the fixed-point technique.

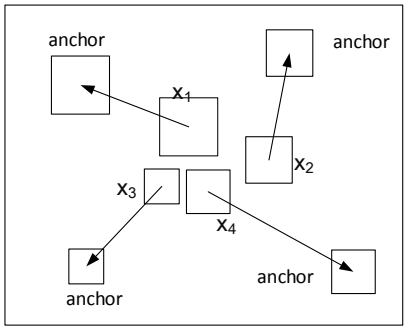


Fig. 2. An example of the fixed-point technique [11]. In this example, the objective function is formulated as a quadratic penalty function: $\rho\left[(x_1 - x_1^{anch})^2 + (x_2 - x_2^{anch})^2 + (x_3 - x_3^{anch})^2 + (x_4 - x_4^{anch})^2\right]$, where ρ is the weight of pseudo net. In POLAR, the weight of all pseudo nets are the same. This penalty function is added to Formula (2).

III. POLAR'S ALGORITHM

A. Algorithm outline

As shown in Fig. 3, POLAR is composed of three stages: initial placement, density-driven placement and post-global placement.

In the initial placement stage, a good wirelength-driven seed placement without considering cell overlapping is generated. Firstly, the hybrid net model [23] is responsible to the initial connection matrices, and linear system (3) is solved by PCG to get the initial placement. Next, the B2B net model updates the connection matrices to further optimize the wirelength iteration by iteration. Usually, three iterations are enough.

In the density-driven placement stage, POLAR adopts the iterative look-ahead legalization framework [14]. In each iteration, the look-ahead legalization is used to generate upper bound wirelength, and the quadratic wirelength achieved by solving linear system (3) is considered as lower bound wirelength[1]. To realize look-ahead legalization, firstly, all placement density hotspots are detected. Secondly, for each hotspot, a minimal window is searched to cover it by enumerating the feasible candidates. Finally, the movable cells are evenly assigned to each bin within the window by a recursive bisection method and then the cells within each bin are spread out. After look-ahead legalization is finished, the cells' new positions are used as anchors to generate spreading forces and the connection matrices Q_x and Q_y are also updated based on anchors' positions by B2B net model [9]. The density-driven placement runs iteratively until it satisfies the convergence condition, which is defined as that the gap between lower bound wirelength and upper bound wirelength is less than 8%.

Finally, in the post-global placement stage, look-ahead legalization is applied once more. And then legalization and detailed placement are performed by FastDP [24] (using the same setting as [24]'s) to get a legal placement.

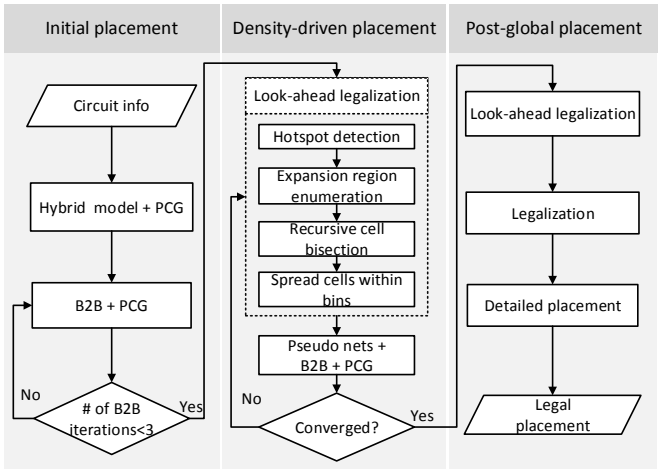


Fig. 3. The overview of POLAR.

B. Placement density estimation

One of the goals of look-ahead legalization is to achieve a roughly legalized placement. To evaluate whether a placement is roughly legalized, a method to estimate placement density is necessary.

A popular and widely used approach is to split the placement region into a set of $p \times q$ uniform bins denoted by $B = \{b_{1,1}, b_{1,2}, \ldots, b_{1,q}, b_{2,1}, b_{2,2}, \ldots, b_{p,q}\}$. To simplify placement density estimation, we define the following concepts.

[1]This is not a real lower bound on the wirelength of the placement problem because spreading forces that are generated by heuristics are added to the linear system.

Definition 1: For any movable cell v_i, if its geometric center is located in bin $b_{x,y}$, we say that v_i is **belonged** to $b_{x,y}$. And the placement density of bin, $d_{i,j}$, is defined as Formula (5), where $o_{i,j}$ and $a_{i,j}$ are the total area of movable cells belonged to $b_{i,j}$ and the available area of $b_{i,j}$ respectively. For mixed-size placement problem, $a_{i,j}$ can be calculated offline, while $o_{i,j}$ should be calculated online.

$$d_{i,j} = \frac{o_{i,j}}{a_{i,j}} \tag{5}$$

Definition 2: A bin $b_{i,j}$ is considered density **overflow** if the following condition (6) is satisfied.

$$d_{i,j} > \lambda \times \theta \tag{6}$$

where λ is set to 1.05 in POLAR and θ is the density target of the circuit. Otherwise, we call bin $b_{i,j}$ **underflow**.

The size of bin is determined as follows. Suppose the average area of standard cell is $\overline{area}$ and the expected number of standard cells in a bin is $\overline{n}$, then the bin is a square whose width (height) is calculated by Formula (7). In POLAR, the default value of $\overline{n}$ is 30, and the bin size is fixed during the global placement.

$$Grid_h = Grid_w = \sqrt{\frac{\overline{n} \times \overline{area}}{\theta}} \tag{7}$$

C. Hotspot detection

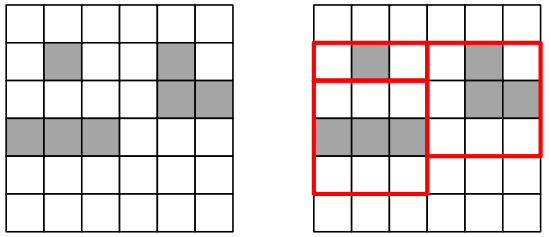
The first step of look-ahead legalization is to recognize placement density hotspots. Similar to SimPL [14], the placement density hotspot is defined as a cluster of overflow bins. Its formal definition is given as follows.

Definition 3: The **grid graph** for the uniform bin grid is the graph in which (i) each bin represents a vertex, and (ii) two vertices are joined by a graph edge if and only if the two bins for those vertices are directly adjacent, either horizontally or vertically. That is, referring to a bin by its (row,column) coordinates in the uniform bin grid, bins (i, j) and (k, l) are adjacent if and only if $|k - i| + |l - j| = 1$.

Definition 4: A **placement density hotspot** is a spatially contiguous collection of overflow bins, i.e., a connected subgraph of overflow bins in the grid graph. A hotspot is also called a "clump." For any pair of bins in the clump, there is a path in the grid graph connecting them. The edges of the path can only be vertical or horizontal, and this path cannot go through the bins outside of the clump.

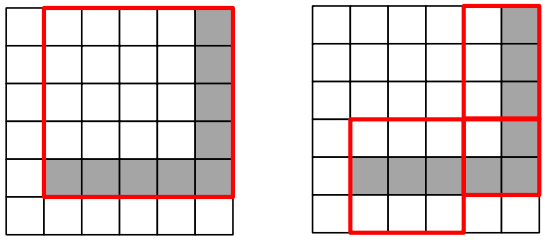
Fig. 4(a) gives an example of placement density hotspots. The shadowed bins are overflow, so there are three placement density hotspots according to our definitions.

The algorithm of hotspot detection is presented in Algorithm 1. Breadth first search (BFS) is used to traverse all overflow bins. Once the number of overflow bins in currently constructing density hotspot exceeds 3, a new overflow bin is set as a new start for BFS, the reason is explained in Section III.D. The time complexity of Algorithm 1 is O(pq), since each bin is visited at most twice.



(a) Placement density hotspots. (b) Window cover hotspots.

Fig. 4. Placement density hotspots and windows.



(a) Ill-shaped hotspot and corresponding window. (b) Ill-shaped hotspot decomposition and windows.

Fig. 5. Ill-shaped hotspot and its decomposition.

D. Window enumeration

To spread out the cells in placement density hotspot, we define window as follows.

Definition 5: The **window** of a hotspot is a set of bins which completely cover the hotspot and has enough available space to accommodate the movable cells within it while satisfying the density constraints.

As shown in Fig. 4(b), the three placement density hotspots in Fig. 4(a) are fully covered by three windows, respectively. Therefore, if the cells are evenly distributed within all those windows, a roughly legalized placement is expected.

As mentioned before, one goal of look-ahead legalization is to maintain the cells' relative positions of quadratic placement while minimizing the cell movements. Generally speaking, for any placement density hotspot, smaller window means less cell movements so it is preferred. To avoid unnecessarily big window for ill-shaped hotspot, we simply constrict the number of overflow bins in each placement density hotspot, as shown in Algorithm 1. As shown in Fig. 5, if the number of overflow bins in each hotspot is constricted to 3, the ill-shaped hotspot is decomposed into three smaller ones.

For any placement density hotspot, there are many candidates of window. According to the definition of window, the number of candidates is O(p^2q^2). Enumerating all of the candidates is time consuming. To trade off runtime and quality, we define τ-enumerated window and the minimal τ-enumerated window is chosen to spread out placement density hotspot.

Algorithm 1 Placement density hotspot detection

Require: The density of bins are already calculated. The bin grid is $p \times q$. The number of bins in each hotspot is constricted to 3.
Ensure: The set of placement density hotspots, denoted by Ω.

for $i = 1 \rightarrow p$ **do**
  for $j = 1 \rightarrow q$ **do**
    visited[i][j]=0;
  end for
end for
for $i = 1 \rightarrow p$ **do**
  for $j = 1 \rightarrow q$ **do**
    if visited[i][j]==1 $\parallel$ $d_{i,j} \leq 1.05 \times \theta$ **then**
      continue;
    end if
    $H = \{b_{i,j}\}$; $count = 0$; visited[i][j]=1;
    Initialize an empty queue Q; push bin $b_{i,j}$ into Q;
    while $Q \neq \emptyset$ **do**
      pop a bin $b_{x,y}$ from Q;
      for each $b_{x,y}$'s adjacent bin $b_{x1,y1}$ **do**
        if visited[x_1][y_1]=0 && $d_{i,j} > 1.05 \times \theta$ **then**
          if $count > 3$ **then**
            $\Omega = \Omega \cup \{H\}$; break;
          end if
          $H = H \cup \{b_{x_1,y_1}\}$; $count = count + 1$;
          visited[x_1][y_1]=1; push b_{x_1,y_1} into Q;
        end if
      end for
    end while
  end for
end for

Definition 6: For any placement density hotspot, a **τ-enumerated window** is a rectangular set of bins, whose geometric center is also the gravity center of corresponding placement density hotspot. Besides, its aspect ratio is within the range $\left[\frac{1}{\tau}, \tau\right]$.

To calculate the geometric center of window and gravity center of placement density hotspot, we use the bin coordinate system. A rectangular-shaped window can be represented by a quadruple (l_x, l_y, u_x, u_y), where (l_x, l_y) is the coordinate of its lower-left bin, and (u_x, u_y) is the coordinate of its upper-right bin. Its geometric center is defined as $\left(floor(\frac{l_x+l_y}{2}), floor(\frac{u_x+u_y}{2})\right)$. The gravity center (g_x, g_y) of placement density hotspot H is defined as Formula (8) and (9).

$$g_x = floor(\frac{\sum_{b_{i,j} \in H} o_{i,j} * i}{\sum_{b_{i,j} \in H} o_{i,j}}) \tag{8}$$

$$g_y = floor(\frac{\sum_{b_{i,j} \in H} o_{i,j} * j}{\sum_{b_{i,j} \in H} o_{i,j}}) \tag{9}$$

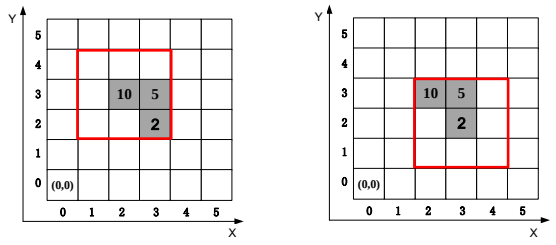
For instance, as shown in Fig. 6, the red rectangular-shaped window is represented by quadruple $(1, 2, 3, 4)$, and its geometric center is $(2, 3)$. The corresponding placement density hotspot is composed of three overflow bins, the total area of movable cells belonged to those bins are 10, 5 and 2 respectively, its gravity center is $(2, 3)$ according to Formula (8)-(9). Therefore, the red rectangular-shaped window in Fig. 6(a) is a τ-enumerated window, whose aspect ratio τ is equal to 1, while the red rectangular-shaped window in Fig. 6(b) is not a valid τ-enumerated window, since its geometry center is not the same as the gravity center of the density hotspot. Note that, by only enumerating τ-enumerated windows, POLAR would miss some candidates of windows. However, the CPU runtime consumption is reduced significantly.

A window whose aspect ratio is either excessively high or excessively low is not beneficial to wirelength. Because the x-direction wirelength would be sacrificed if using excessively

Algorithm 2 Window enumeration for a placement density hotspot

Require: $p \times q$ bins of placement region, a hotspot H, target utilization θ.
Ensure: window (l_x, l_y, r_x, r_y) for hotspot H.

Calculate the gravity center (g_x, g_y) of H according to Formula (8) and (9);
$\Gamma = \emptyset$;
$found = false$;
$\tau = 2.5$;
while $not found$ **do**
  for $radius_x = 1 \rightarrow \max\{g_x, p - g_x\}$ **do**
    for $radius_y = 1 \rightarrow \max\{g_y, q - g_y\}$ **do**
      $l_x = \max\{0, g_x - radius_x\}$;
      $l_y = \max\{0, g_y - radius_y\}$;
      $r_x = \min\{p - 1, g_x + radius_x\}$;
      $r_y = \min\{q - 1, g_y + radius_y\}$;
      if window (l_x, l_y, r_x, r_y) does not cover H **then**
        continue;
      end if
      Calculate the space utilization ratio $\gamma = \frac{\sum_{b_{i,j} \in H} o_{i,j}}{\sum_{b_{i,j} \in H} a_{i,j}}$ of window (l_x, l_y, r_x, r_y);
      if $\gamma < \theta$ && $\frac{r_y - l_y}{r_x - l_x} \in \left[\frac{1}{\tau}, \tau\right]$ **then**
        Push window (l_x, l_y, r_x, r_y) into Γ;
        break;
      end if
    end for
  end for
  if $\Gamma = \emptyset$ **then**
    $\tau += 0.5$;
  else
    $found = true$;
  end if
end while
return the (area) minimal window from Γ;



(a) τ-enumerated window. (b) Non τ-enumerated window.

Fig. 6. Bin coordinate system and τ-enumerated window.

low aspect ratio window, while the y-direction wirelength would be sacrificed if using excessively high aspect ratio window. For example, as shown in Fig. 7, suppose that the total occupied area of each overflow bin is 9 and the total occupied area of underflow bins are all 0. If the available area of each bin is 3 and density target θ is 1, the window in Fig. 7(a) is expected to produce better wirelength than the window in Fig. 7(b). (The y-direction wirelength of Fig. 7(b) may be a little better than that of Fig. 7(a), but the x-direction wirelength of Fig. 7(b) may be much worse than that of Fig. 7(a).)

The window enumeration method is presented in Algorithm 2. The initial value of τ is set to 2.5. All the τ-enumerated windows are checked to find the minimal one. If no τ-enumerated window has enough available area to accommodate the cells within it while satisfying the density constraint, the value of τ is gradually increased by 0.5.

Algorithm 2 guarantees to return a window for the given hotspot. Its time complexity is $O(\tau pq)$ as line 15 (calculating the space utilization ratio of window) can be computed in constant time by a look-up table method, which we will show next.

In a $p \times q$ grid, for any rectangular-shaped window denoted by $(0, 0, x, y)$ whose lower-left bin is $(0, 0)$ and upper-right bin is (x, y), its available area is denoted by $z_{x,y}$. Then the available area of window (l_x, l_y, u_x, u_y) which is denoted by $a\_(l_x, l_y, u_x, u_y)$, can be calculated according to Formula (10).

$$a\_(l_x, l_y, u_x, u_y) = z_{u_x,u_y} - z_{l_x,u_y} - z_{u_x,l_y} + z_{l_x,l_y} \quad (10)$$

The same method can be applied to calculate the occupied area of windows. Therefore, if we maintain two 2-D arrays to store available area and occupied area, the space utilization γ of window can be computed in constant time. Besides, this kind of 2-D array can be constructed and updated by dynamic programming based on the following recursive relation (11).

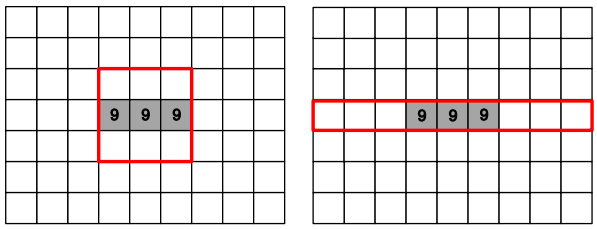
$$z_{x,y} = z_{x-1,y} + z_{x,y-1} - z_{x-1,y-1} + a_{x,y} \quad (11)$$

Only the look-up table of occupied area should be updated once the placement is changed. Once the cell distribution of a window denoted by (l_x, l_y, u_x, u_y) is changed and the cell distribution outside of this window is untouched, the occupied area of three regions should be updated. They are respectively (1) $x \in [l_x, u_x], y \in [l_y, u_y]$, (2) $x \in [u_x + 1, p), y \in [l_y, u_y]$ and (3) $x \in [l_x, u_x], y \in [l_y + 1, q)$. Note that the boundary conditions (e.g. $l_x = 0$ or $l_y = 0$) and update order should be taken care of. The update of look-up table for occupied area is presented in Algorithm 3. Algorithm 3 should be invoked each time when a window is roughly legalized.

Algorithm 3 Update of look-up table for occupied area

Require: Cell distribution of a window denoted by (l_x, l_y, u_x, u_y) is changed. The occupied area in bin $b_{i,j}$ is denoted by $o_{i,j}$. The grid size is $p \times q$. The occupied area in window $(0, 0, x, y)$ whose lower-left bin is $(0, 0)$ and upper-right bin is (x, y) is denoted by $O_{x,y}$.
Ensure: Look-up table is update.

```
1: if l_x == 0 || l_y == 0 then
2:     O_{0,0} = o_{0,0};
3:     for i = 1; i < p; ++i do
4:         O_{i,0} = O_{i-1,0} + o_{i,0};
5:     end for
6:     for j = 1; j < q; ++j do
7:         O_{0,j} = O_{0,j-1} + o_{0,j};
8:     end for
9: end if
10: for i = max{l_x, 1}; i <= u_x; ++i do
11:     for j = max{l_y, 1}; j <= u_y; ++j do
12:         O_{i,j} = O_{i-1,j} + O_{i,j-1} - O_{i-1,j-1} + o_{i,j};
13:     end for
14: end for
15: for i = max{l_x, 1}; i <= u_x; ++i do
16:     for j = u_y + 1; j < q; ++j do
17:         O_{i,j} = O_{i-1,j} + O_{i,j-1} - O_{i-1.j-1} + o_{i,j}
18:     end for
19: end for
20: for i = max{u_x, 1} + 1; i < p; ++i do
21:     for j = max{l_y, 1}; j <= u_y; ++j do
22:         O_{i,j} = O_{i-1,j} + O_{i,j-1} - O_{i-1,j-1} + o_{i,j};
23:     end for
24: end for
```



(a) Reasonable aspect ratio. (b) Unreasonable aspect ratio.

Fig. 7. τ-enumerated window.

E. Recursive bisection based cell spreading

Once the window for placement density hotspot is determined, the cells (within window) are evenly assigned to each bin (within window) in order to get a roughly legalized placement. During this cell-to-bin assignment, it is almost impossible to maintain the x- and y-directed relative cell positions simultaneously, due to the irregular distribution of placement density. To balance the loss of x and y-directed relative positions, the horizontal and vertical cut are applied alternatively similar to the slicing tree of [25]. When vertical cut is applied, the cells are sorted by their x positions, and the cells on left part are assigned to left child, the other cells are assigned to right child. While the horizontal cut is applied, the cells are sorted by their y positions, and cells on the bottom and top part are respectively assigned to left and right child. We define partition tree as follows.

Definition 7: For each window, its partition tree is a binary tree. Its node can be represented by four fields: a split line, a left child, a right child and an associated rectangle. Partition tree of a window satisfies the following conditions:

- The whole window is the associated rectangle of root.
- For each inner node, the split line should be one of the

horizontal/vertical lines that divide the placement region into uniform bins.

- For each inner node, its associated rectangle is split into two parts by its split line. If its split line is horizontal, the above/below part is associated to its left/right child. Similarly, if its split line is vertical, the left/right part is associated to its left/right child.
- For each leaf node, its split line is invalid and its associated rectangle is one-to-one mapped to a bin.

For each inner node, we try to balance the sizes of associated rectangles of its two children by choosing the split line which is closest to the middle horizontal or vertical line of its associated rectangle. A simple implementation of this cell-bin assignment is presented in Algorithm 4. At the beginning, only the root of partition tree is in the queue Q. In each iteration, a tree node is popped from Q and partitioned into two by a horizontal or vertical cut. During the partition, the space utilization ratios of its two children's corresponding rectangles are closed to each other by properly allocating movable cells. The Algorithm 4 stops until the queue Q is empty, which means that the partition tree is constructed completely. For example, Fig. 8 is the partition tree for an 3 $\times$ 3 grid window.

Comparing with [25], there are several differences in our recursive bisection approach. Firstly, cut line should be strictly one of the split lines that divide placement region into uniform bins. Secondly, the leaf node should be strictly one-to-one mapped to a bin in order to control the placement density of each bin, which means that partition tree may not be a complete binary tree. Thirdly, the slice tree proposed by [25] is used to adjust cut line to handle with routing congestion, while our partition tree is served to achieve roughly legal placement where placement density of each bin is closed to design target.

Algorithm 4 Cell-to-bin assignment within window

Require: The set of cells S and window F (l_x, l_y, r_x, r_y).
Ensure: Each cell is assigned to exactly one bin.
1: respectively get the x and y-directed cell ordering;
2: determine the initial cut type T, it is vertical when $r_x - l_x > r_y - l_y$, otherwise horizontal;
3: create a root node R, push the quadruple (R, F, T, S) into a queue Q;
4: **while** Q is not empty **do**
5: pop (R, F, T, S) from Q;
6: partition the window and cell set, the results are $(F1, S1)$ and $(F2, S2)$, where $F = F1 \cup F2$ and $S = S1 \cup S2$;
7: create two children $C1, C2$ for inner node R;
8: change cut type T;
9: **if** $F1$ is not a bin and not empty **then** push ($C1$, $F1$, T, $S1$) into Q
10: **end if**
11: **if** $F2$ is not a bin and not empty **then** push ($C2$, $F2$, T, $S2$) into Q
12: **end if**
13: **end while**

1) Speedup technique: The runtime of Algorithm 4 is dominated by line 6 (partition inner tree node). For each inner tree node, sorting should be performed to partition cells. Therefore, the time complexity of Algorithm 4 is O($nlog^2n$), where n is the number of cells within window. Next, we will show a clever implementation of Algorithm 4. We can reduce the time complexity to O($(p+q)k+nlog(pq)$) for a $p \times q$ grid window, where k is a small constant.

There are two techniques to speedup Algorithm 4. Firstly, cells are not really moved during the construction of partition tree. For each cell, a path from the root to the leaf (the bin to which it is finally assigned) is maintained. So once the partition tree is constructed, each cell could be assigned to a bin by going through its path. For the convenience of computation, a bit sequence is maintained to denote this path. Secondly, the construction of partition tree is strictly level-by-level. To generate a new level of partition tree, the whole sorted cell list of window is just scanned once. Therefore, the total number of sorting is reduced to 2 [14].

Besides, we can take advantage of that the ranges of x and y coordinates of cells are known when the window is given. If we split every bin into k horizontal or vertical stripes like [14], the x and y-directed sorting can be done by bucket sort only losing little accuracy. And our experimental results show that the placement quality is not suffered if k is set to a reasonable value, which is 100 in POLAR.

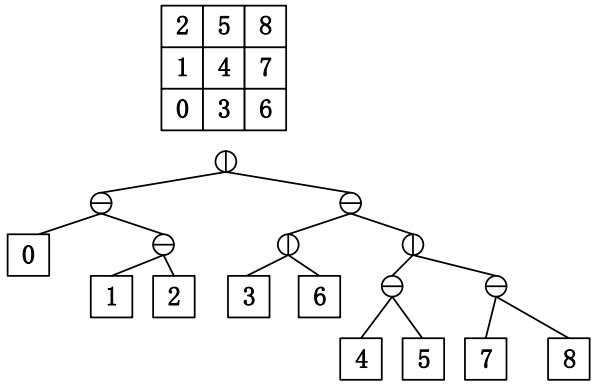


Fig. 8. An example of partition tree for 3 $\times$ 3 window, the vertical cut line is first applied.

The detail of our speedup technique is presented in Algorithm 5. The inner nodes that have been partitioned are called **dead nodes**, otherwise **live nodes**. For each window, at the beginning, there is only one live node. The partition tree is constructed level by level, and a queue Q is used to maintain current live nodes. In each level, the same type of cut line is applied to all the live nodes in lines 8-19. For any sub window F (l_x, l_y, u_x, u_y), it is divided into two subregions L (left subregion by vertical cut $\frac{l_x+u_x}{2}$, or bottom subregion by horizontal cut $\frac{l_y+u_y}{2}$) and R (right subregion by vertical cut $\frac{l_x+u_x}{2}$, or top subregion by vertical cut $\frac{l_y+u_y}{2}$). The partition pivot is computed based on Formula (12).

$$\frac{A_L}{(A_L + A_R)} \times A_M \tag{12}$$

where A_L and A_R are the available area of L and R respec-

tively, A_M is the total area of moveable cells within F. There is a special case during the partition. If the sub window is a 1×2 (2×1) grid, the vertical (horizontal) cut line can not be applied apparently, since each leaf of partition tree is one-to-one mapped to a bin. Therefore, partition of this sub windows is delayed to the next level where the cut type is changed, we call this situation **level delay**.

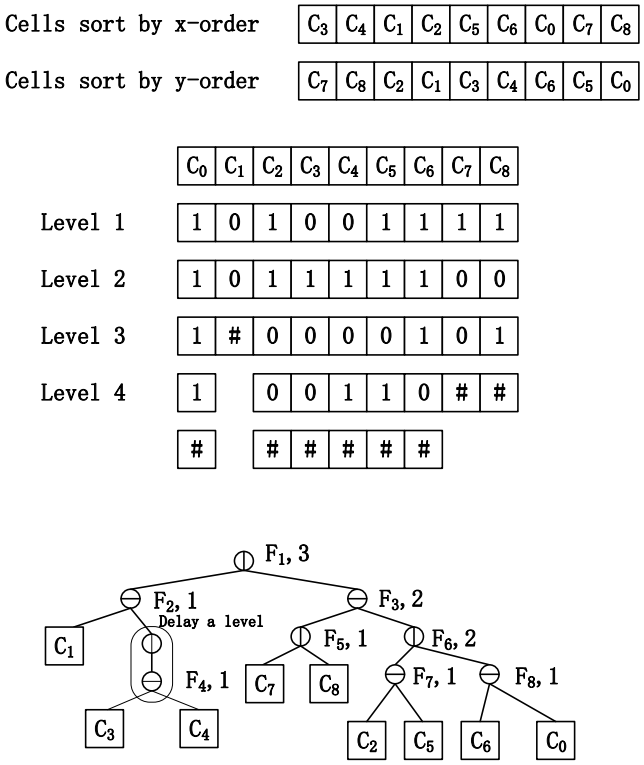


Fig. 9. An example for recursive bisection based cell spreading with speedup technique. The value on the right side of tree node is the pivot for partition.

An example is illustrated in Fig. 9. Assuming that the given window F_1 is a 3×3 grid, each bin has the same available space. The nine equal sized cells within F_1 are $\{C_1, C_2, \cdots, C_9\}$, the area of each cell is 1. The cells are sorted in ascending order according to their x- and y-coordinates respectively, the x-directed order is $(C_3, C_4, C_1, C_2, C_5, C_6, C_0, C_7, C_8)$, while the y-directed order is $(C_7, C_8, C_2, C_1, C_3, C_4, C_5, C_6, C_0)$.

In the first level, the vertical cut line is applied, according to Formula (12), the pivot is 3. The cell list is scanned in x-directed order, then C_3, C_4, C_1 should be assigned to left sub-window, others should be assigned to right sub-window, the partial path of each cell is updated. In the second level, the horizontal cut line is applied, the partition pivots for F_2 and F_3 are respectively 1 and 2. Then the cell list is scanned in y-directed order. The first one is C_7. Its partial path is "1", which means that it should be assigned to F_3 in the last level, and the current pivot of F_3 is 2, so it should be assigned to F_5 and its partial path is "10". The partial paths of other cells are updated similarly. In the third level, the vertical cut line is applied and the cell list is scanned in x-directed order. The first cell is C_3, its current partial path is "01". Since the associated rectangle of the node to which C_3 is belonged is F_4, and F_4 a 1×2 grid, a level delay happens and we update the partial path of C_3 to "010". The next cell is C_4. The level delay happens again and its partial path is updated accordingly. The next cell is C_1 and its partial path is "00". Since the associated rectangle of the node to which C_1 is belonged is already a bin, we do not update its partial path. The next cell is C_2 and its partial path is "11". In this case, we fetch the pivot of the node to which it is belonged and this pivot is 2. Therefore, C_2 should be assigned to left child and its partial path is updated to "110". The partial paths of other cells are updated similarly.

Algorithm 5 Speed up technique for cell-to-bin assignment

Require: The set of cells S and window F (l_x, l_y, r_x, r_y)
Ensure: Each cell is assigned to exactly one bin.
1: perform x and y-directed cell sorting by bucket sort;
2: determine the initial cut type T, it is vertical when $r_x - l_x > r_y - l_y$, otherwise horizontal;
3: create a root node R, push the (R, F, $pivot$) into a queue Q;
4: **while** true **do**
5: $\Phi = \emptyset$; ▷ Φ stores the next level sub windows
6: **if** Q is empty **then** break; ▷ all the leaf nodes have been generated
7: **end if**
8: **while** Q is not empty **do**
9: pop (R, F, $pivot$) from Q;
10: **if** F is a 1×2 (2×1) grid && the cut is vertical (horizontal) **then**
11: $\Phi = \Phi \cup F$;
12: **else if** F is not bin **then**
13: partition F into $F1$ and $F2$ based on $pivot$;
14: create two children $C1$ and $C2$ for R;
15: calculate the pivot for $F1$ and $F2$, denoted by $pivot1$, $pivot2$;
16: $\Phi = \Phi \cup \{(C1, F1, pivot1), (C2, F2, pivot2)\}$;
17: update the bit sequence for each cell;
18: **end if**
19: **end while**
20: change the cut type T;
21: push each element of Φ into Q;
22: **end while**
23: **for** each cell in S **do**
24: cur_node := root of partition tree;
25: **for** each bit in the bit sequence of the cell **do**
26: **if** current bit is 0 **then**
27: cur_node: = cur_node's left child;
28: **else**
29: cur_node: = cur_node's right child;
30: **end if**
31: **end for**
32: assign this cell to the bin associated to cur_node;
33: **end for**

For any cell, once we know its path from root to leaf in partition tree, the bin which it is finally assigned to can be determined according to line 24-32 in Algorithm 5. For example, as shown in Fig. 9, the path of C_8 is "101" and cur_node in Algorithm 5 is set to root (represented by F_1)

initially. The first bit of the path is "1", so cur_node is set to F_3 which is the right child of F_1. The second bit is "0", so cur_node is set to F_5 which is the left child of F_3. Finally, the last bit is "1", so it should be assigned to bin 6 in Fig. 8.

We analyze the time complexity of Algorithm 5 as follows. Line 1 needs O($(p+q)k+n$). The level of partition tree is the same as its height, which is O($log(pq)$). In each level, the whole cell list is scanned once by either x-directed or y-directed order, so the line 4-22 needs O($nlog(pq)$). For each cell, O($log(pq)$) is needed to scan its path (bit sequence). Therefore, the total time complexity of Algorithm 5 is O($(p+q)k+nlog(pq)$).

F. Cell spreading within bins

After recursive bisection based cell spreading, the information that which cell belongs to which bin is known. However, the concrete positions of cells in each bin are still undetermined. Here, similar with [14], we use a scaling method [24] to handle with this issue. To calculate x-coordinates of the cells within a bin, these cells are sorted in ascending order according to their original x-coordinates before look-ahead legalization. Then all these cells are put side by side in x-direction based on the above ordering. Since the total cell width may be longer than the width of bin, the x-coordinates of cells are scaled to make sure the centers of all the cells are between bin's left boundary and right boundary. Similarly, y-coordinates of the cells within a bin can be calculated. The details are presented in Algorithm 6.

Algorithm 6 Cell spreading in a bin

Require: The cells belong to the bin.
Ensure: The positions of cells.
1: Get the coordinate of bin's lower-left corner, denoted by (bin_x, bin_y) ;
2: Sort the cells in ascending order based on original x-coordinates before look-ahead legalization
3: $total\_width = 0$;
4: **for** each cell v **do**
5: $total\_width += v$'s width;
6: **end for**
7: $cur\_pos_x = 0$;
8: **for** each cell v **do**
9: v's x-coordinate = $bin_x + \frac{cur\_pos_x}{total\_width}$;
10: $cur\_pos_x += v$'s width;
11: **end for**
12: Calculate the y-coordinate of each cell by similar method;

G. Complete algorithm of look-ahead legalization

Algorithm 7 combines all the above together. The input could be any uneven placement. Firstly, the density estimation of given placement is performed to update the loop-up table mentioned in section III.D. Then Algorithm 1 is applied to find all the density hotspots. Density hotspots are sequentially handled according to their density. For each hotspot H, Algorithm 2 is used to find its window, within which Algorithm 5 finishes the cell-to-bin assignment. Since some bins of hotspot H may have been already covered by previous windows and are not overflow any more, they are subtracted from H. Note that the cell distribution of placement is changed after applying Algorithm 5, so the look-up table should be updated by Algorithm 3. At the end, cells within each bin are simply spread out by Algorithm 6. To illustrate our look-ahead legalization, placement migration of circuit adaptec1 is shown in Fig. 10, where both lower bound and upper bound placement are presented.

Algorithm 7 Look-ahead legalization of POLAR

Require: A placement and target utilization θ.
Ensure: The density of each bin is closed to θ.
1: Build up the look-up table (see III.D formula (10)).
2: Placement density hotspot detection (see Algorithm 1).
3: Sort all the density hotspots in descending order of density (the ratio of cell area and available area within hotspot).
4: **for** each density hotspot H **do**
5: In H, subtract the bins that have been covered by previous windows.
6: Find an expansion window F for H (see Algorithm 2).
7: Cell-to-bin assignment within F (see Algorithm 5).
8: Update the look-up table because the cell distribution of F is changed (see Algorithm 3).
9: **end for**
10: Spread the cells within bins (see Algorithm 6).

H. Force modulation

After all the anchors of cells are determined, the linear system (3) is updated and solved again. Firstly, the B2B net model is refreshed based on anchor positions of cells. Then, for each movable cell, a two-pin pseudo net connecting it to its anchor is added into the spring system. The weight ρ of pseudo net is calculated according to Formula (13), where i means the ith iteration of density driven placement.

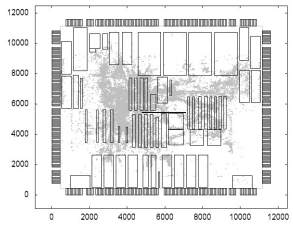
$$\rho = \begin{cases} \varepsilon & \text{if } i = 0 \\ \varepsilon * \alpha^{i-1} & \text{if } 1 \le i \le 20 \\ \varepsilon * \alpha^{19} * \beta^{i-20} & \text{if } i > 20 \end{cases} \tag{13}$$

It is a two-stage force modulation, where ε is a small value and $\beta > \alpha$. At the early stage, ρ is small in order to avoid significant change of placement. While at the latter stage, ρ is increased more quickly to speedup the convergence. For ISPD 2005 and 2006 benchmark suites, the default value of $\varepsilon, \alpha, \beta$ are respectively 1e-5, 1.05, 1.15.

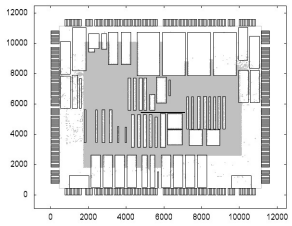
I. Handling with movable macros

To handle with movable macros, there are many existent methods such as [15, 26–29]. We used a light-weight approach which can be easily integrated into Algorithm 7 as shown in Algorithm 8.

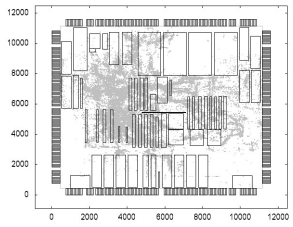
The idea originates from [26] and [15]. Each movable macro is sliced into a set of equal-sized pieces. Similar with [15], these sliced pieces share the same central position as the movable macro which they are from just before look-ahead legalization. Different from [26], these sliced pieces are not



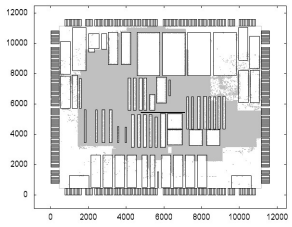
(a) The lower bound placement in the 5th iteration.



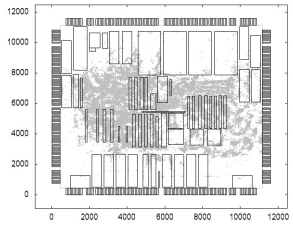
(b) The upper bound placement in the 5th iteration.



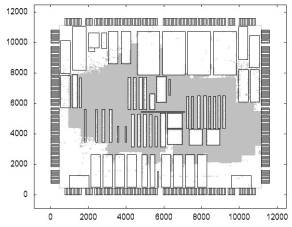
(c) The lower bound placement in the 15th iteration.



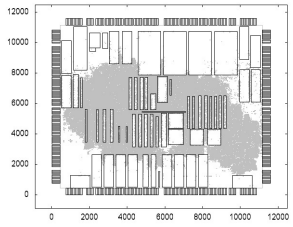
(d) The upper bound placement in the 15th iteration.



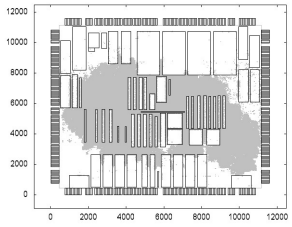
(e) The lower bound placement in the 30th iteration.



(f) The upper bound placement in the 30th iteration.



(g) The lower bound placement in the last iteration.



(h) The upper bound placement in the last iteration.

Fig. 10. Placement migration of circuit adaptec1.

connected by fake nets, so this approach does not touch matrix generation. In other words, exactly the same as standard cell, each movable macro is treated as an entity rather than a set of sliced pieces during matrix generation. The size of each piece is about the average size of standard cells. Since POLAR tries to maintain the relative positions of cells, most of the sliced pieces belonged to the same macro would be very close to each other. And the position of movable macro is the gravity center of its sliced pieces just after look-ahead legalization. Therefore, even few of its sliced pieces maybe a little far away from the majority of others, it has little impact on the final position of movable macro.

Algorithm 8 Handle with movable macros

1: Shredding moveable macros into slices;
2: Apply Algorithm 7;
3: Resembling the slices into macros;

IV. Experimental results

POLAR is implemented in C++. The binaries of several modern academic placers such as NTUPlacer3[8], mPL6 [4], FastPlace3 [12], SimPL [14] and ComPLx [15] are also obtained. However, MAPLE [16] is not available since it is a commercial tool. All results (excepts those of MAPLE) are generated in the same platform, which is a Linux PC with 16GB of memory and Intel Core-i3 3.3GHz CPU. Note that some academic placers such as SimPL and ComPLx have paralleled implementation, while some other do not. To make the comparison fair, we only allow to use one core for all placers. The ISPD 2005 benchmark suite [30] and ISPD 2006 benchmark suite [31] are used to verify the efficiency of POLAR.

A. benchmark characteristics

The characteristics of ISPD 2005 benchmark suite and ISPD 2006 benchmark suite are listed in Table I and Table II. For the ISPD 2005 benchmark suite, the density target θ is set to 1, which means that the whitespace in each bin could be occupied by movable cells completely. However, for the ISPD 2006 benchmark suite, the density target of each benchmark is fixed by contest organizers. The range of number of cells (including both movable cells and fixed macros/pins) in those benchmarks is from 0.2 millions to 2.5 millions.

TABLE I. Characteristics of ISPD 2005 benchmark suite.

benchmark	# of V	# of E	design utility	density target
adaptec1	211447	221142	57.34%	100%
adaptec2	255023	266009	44.32%	100%
adaptec3	451650	466758	33.52%	100%
adaptec4	496045	515951	27.14%	100%
bigblue1	278164	284479	44.67%	100%
bigblue2	557866	577235	37.78%	100%
bigblue3	1096812	1123170	56.48%	100%
bigblue4	2177353	2229886	42.29%	100%

TABLE II. Characteristics of ISPD 2006 benchmark suite.

benchmark	# of V	# of E	design utility	density target
adaptec5	843128	867798	49.97%	50%
newblue1	330474	338901	83.20%	80%
newblue2	441516	465219	61.66%	90%
newblue3	494011	552199	26.30%	80%
newblue4	646139	637051	46.45%	50%
newblue5	1233058	1284251	49.55%	50%
newblue6	1255039	1288443	38.78%	80%
newblue7	2507954	2636820	49.31%	80%

B. Comparison on ISPD2005 benchmark suite

For the ISPD 2005 benchmark suite, the quality of placement is measured by wirelength. The experimental results are presented in Table III. On average, POLAR achieves the best quality (respectively improve the wirelength by 7.87%, 2.68%, 6.67%, 2.32%, 1.30% and 0.14% compared with NTUPlacer3, mPL6, FastPlace3, SimPL, ComPLx and MAPLE) and it is also very fast. Comparing with SimPL which is fastest one, POLAR improves the wirelengh by 2.32% at the cost of 6% increase in the runtime. Comparing with MAPLE which produces similar wirelength with POLAR, we get 6.73× speedup [2].

C. Comparison on ISPD2006 benchmark suite

In the ISPD 2006 benchmark suite, the circuit newblue1 has several movable macros. The quality of placement is measured by scaled wirelength. The scaled wirelength is composed of two parts: wirelength and the penalty of overflow. The contest official script [31] is used to calculate the scaled wirelength, and the experimental results are presented in Table IV and Table V. On average, POLAR achieves improvement of 3.54%, 5.74%, 11.68%, 0.79% and 0.44% on scaled wirelength versus NTUPlace3, mPL6, FastPlace3, ComPLx and MAPLE. We do not have the results of SimPL, since it currently does not support to run on ISPD 2006 benchmark suite. For the runtime, on average, POLAR is 2.59×, 8.91×, 1.00×, 1.05× faster than NTUPlacer3, mPL6, FastPlace3 and ComPLx. The runtime of MAPLE on ISPD 2006 benchmark suite was not reported in [16], so we cannot compare POLAR with it either directly or indirectly.

D. Runtime analysis

The runtime breakdown of POLAR is shown in Table VI. It is divided into three components: global placement legalization, and detailed placement. The runtime of global placement is further divided into three parts: PCG, look-ahead legalization (which includes hotspot detection, window enumeration and recursive bisection based cell spreading) and others (e.g. B2B net model update, wirelength calculation and I/O).

On average, global placement takes about 76% of total runtime, while legalization and detailed placement respectively take about 7% and 17% of total runtime. In global placement stage, PCG takes about 50% of total runtime, look-ahead legalization takes 14% (window enumeration uses 2% and recursive bisection based cell spreading uses 9%) and others take 12% of total runtime.

V. Conclusions

In this paper, we have proposed a high performance mixed-size wirelength-driven placer called POLAR. It adopts the popular framework of legalization. An elegant and effective algorithm for look-ahead legalization is proposed. The experimental results on ISPD 2005 benchmark suite and ISPD 2006 benchmark suite verify that our placer is very comparable to the state-of-the-art placers in both runtime and placement quality.

[2]The binary of MAPLE is not released, so MAPLE's runtime is scaled according to [16] which show that it is 7.14 × slower than SimPL

TABLE V. Runtime comparison on ISPD 2006 benchmark suite.

benchmark	NTUPlace3	mPL6	FastPlace3	ComPLx	POLAR
adaptec5	45.2	118.2	16.6	15.0	13.7
newblue1	8.9	27.1	4.2	3.6	6.9
newblue2	17.4	66.4	6.7	8.6	6.9
newblue3	15.1	102.4	8.6	7.7	7.1
newblue4	26.4	89.1	9.2	9.6	9.0
newblue5	58.1	161.8	21.4	23.6	21.5
newblue6	50.0	133.9	21.1	19.1	17.8
newblue7	120.3	377.1	33.2	47.4	38.5
Norm.	2.59×	8.91×	1.00×	1.05×	1.00×

References

[1] C. Alpert, Z. Li, G.-J. Nam, C. N. Sze, N. Viswanathan, and S. I. Ward, "Placement: hot or not?," ICCAD '12, pp. 283–290.

[2] I. L. Markov, J. Hu, and M.-C. Kim, "Progress and challenges in VLSI placement research," ICCAD '12, pp. 275–282.

[3] W. Naylor, R. Donelly, and L. Sha, "Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer," U.S.Patent 6 301 693, 2001.

[4] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie, "mPL6: enhanced multilevel mixed-size placement," ISPD '06, pp. 212–214.

[5] G.-J. Nam, S. Reda, C. J. Alpert, P. G. Villarrubia, and A. B. Kahng, "A fast hierarchical quadratic placement algorithm," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 4, pp. 678–691, 2006.

[6] J. Z. Yan, C. Chu, and W.-K. Mak, "SafeChoice: a novel clustering algorithm for wirelength-driven placement," ISPD '10, pp. 185–192.

[7] A. B. Kahng and Q. Wang, "A faster implementation of aplace," ISPD '06, pp. 218–220.

[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. on Computer-Aided Design of Integrated Circuits and System*, vol. 27, no. 7, pp. 1228–1240, 2008.

[9] P. Spindler, U. Schilichtmann, and F. M. Johannes, "Kraftwerk2 - a fast force-directed quadratic placement approach using an accurate net model," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and System*, vol. 27, no. 8, pp. 1398–1411, 2008.

[10] T. Luo and D. Z. Pan, "DPlace2.0: a stable and efficient analytical placement based on diffusion," ASP-DAC '08, pp. 346–351.

[11] B. Hu, Y. Zeng, and M. Marek-Sadowska, "mFAR: fixed-

TABLE III. COMPARISON ON ISPD 2005 BENCHMARK SUITE.

benchmark	NTUPlace3		mPL6		FastPlace		SimPL		ComPLx		MAPLE		POLAR	
	HPWL	runtime	HPWL	runtime	HPWL	runtime	HPWL	runtime	HPWL	runtime	HPWL	runtime	HPWL	runtime
adaptec1	79.83	5.11	77.29	19.3	78.66	2.61	77.53	2.43	77.79	2.64	76.36	19.3	76.54	2.43
adaptec2	90.08	6.73	90.02	20.1	94.06	3.57	91.11	3.50	88.97	3.27	88.97	24.2	86.57	3.37
adaptec3	232.72	13.5	207.0	62.7	214.13	8.10	203.79	6.75	203.57	7.58	209.78	50.7	202.8	7.17
adaptec4	215	16.4	188.87	59.2	197.5	7.43	184.75	5.18	183.22	6.65	179.91	49.2	182.93	7.13
bigblue1	96.94	9.72	96.18	24.4	96.67	3.73	95.59	4.46	95.3	5.20	93.74	24.7	94.36	3.40
bigblue2	158.26	24.2	148.91	68.1	155.74	6.50	145.87	5.61	145.39	7.03	144.45	43.8	143.89	8.13
bigblue3	343.57	27.8	335.53	93.2	365.16	18.8	351.65	17.7	337.96	18.1	323.05	89.5	323.05	16.5
bigblue4	825.48	81.0	814.13	212	836.2	34.7	791.29	26.9	788.8	35.8	775.71	231	791.27	34.9
Norm.	+7.87%	2.26×	+2.68%	7.28×	+6.67%	1.04×	+2.32%	0.94×	+1.30%	1.07×	+0.14%	6.73×	+0.00%	1.00×

TABLE IV. PLACEMENT QUALITY COMPARISON ON ISPD 2006 BENCHMARK SUITE.

benchmark	NTUPlace3		mPL6		FastPlace3		ComPLx		MAPLE		POLAR	
	sWL	overflow	sWL	overflow	sWL	overflow	sWL	overflow	sWL	overflow	sWL	overflow
adaptec5	444.41	28.51	428.31	1.03	472.72	8.17	415.77	1.93	407.33	4.76	411.91	6.42
newblue1	61.01	0.70	72.62	9.02	74.11	1.04	64.75	1.02	69.25	1.05	67.2	1.11
newblue2	194.8	1.82	201.91	1.44	206.04	1.00	193.06	1.05	191.66	1.01	192.8	1.18
newblue3	275.08	0.05	285.26	0.66	297.46	0.55	274.64	0.93	268.07	0.77	270.58	1.01
newblue4	296.62	13.6	298.2	1.70	308.35	4.22	292.82	1.45	282.49	5.86	282.67	3.31
newblue5	537.92	20.3	535.8	1.47	621.47	7.21	507.74	1.76	515.04	4.05	502.96	5.36
newblue6	534.96	0.28	523.47	1.41	549.87	1.02	501.05	1.14	494.82	1.08	497.86	1.39
newblue7	1096.16	2.01	1085.68	1.19	1105.43	1.30	1041.21	1.40	1032.6	1.70	1025.4	1.01
Norm.	+3.54%	2.07	+5.74%	1.62	+11.68%	1.01	+0.79%	0.73	+0.44%	1.01	+0.00%	1.00

TABLE VI. RUNTIME BREAKDOWN OF POLAR.

benchmark	global placement					legalization	DP	total runtime
	PCG	LAL			others			
		window enumeration	cell spreading	others				
adaptec1	66	3	11	5	22	7	32	146
adaptec2	86	3	15	7	25	7	59	202
adaptec3	208	14	29	9	42	21	107	430
adaptec4	203	8	35	11	41	19	111	428
bigblue1	95	3	16	6	27	6	51	204
bigblue2	279	9	30	13	49	32	76	488
bigblue3	436	12	102	29	87	61	261	988
bigblue4	1072	46	187	50	215	122	400	2092
adaptec5	431	15	91	26	90	103	63	819
newblue1	260	1	52	7	55	23	19	417
newblue2	196	5	42	10	43	89	27	412
newblue3	175	29	31	28	40	26	98	427
newblue4	273	7	55	18	58	48	82	541
newblue5	630	20	173	27	127	94	217	1288
newblue6	610	15	120	41	127	64	92	1069
newblue7	1253	129	242	80	245	232	128	2309
Normalize	0.50	0.02	0.09	0.03	0.12	0.07	0.17	1.00

points-addition-based VLSI placement algorithm," ISPD '05, pp. 239–241.

[12] N. Viswanathan, M. Pan, and C. Chu, "FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," ASP-DAC '07, pp. 135–140.

[13] N. Viswanathan, G.-J. Nam, C. J. Alpert, P. Villarrubia, H. Ren, and C. Chu, "RQL: global placement via relaxed quadratic spreading and linearization," DAC '07, pp. 453–458.

[14] M.-C. Kim, D. Lee, and I. L. Markov, "SimPL: An effective placement algorithm," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 31, no. 1, pp. 50–60, 2012.

[15] M.-C. Kim and I. L. Markov, "ComPLx: A competitive primal-dual lagrange optimization for global placement," DAC '12, pp. 747–752.

[16] M.-C. Kim, N. Viswanathan, C. J. Alpert, I. L. Markov, and S. Ramji, "MAPLE: multilevel adaptive placement for mixed-size designs," ISPD '12, pp. 193–200.

[17] M.-C. Kim, J. Hu, D.-J. Lee, and I. L. Markov, "A SimPLR method for routability-driven placement," ICCAD '11, pp. 67–73.

[18] X. He, T. Huang, L. Xiao, H. Tian, G. Cui, and E. F. Y. Young, "Ripple: an effective routability-driven placer by iterative cell movement," ICCAD '11, pp. 74–79.

[19] T. Lin and C. Chu, "POLAR 2.0: An effective routability-driven placer," in *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, DAC '14, pp. 123:1–123:6, 2014.

[20] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei, "ICCAD-2012 CAD contest in design hierarchy aware

routability-driven placement and benchmark suite," ICCAD '12, pp. 345–348.
[21] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2nd ed., 2003.
[22] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 3, pp. 356–365, 1991.
[23] N. Viswanathan, M. Pan, and C. C.-N. Chu, "FastPlace: An analytical placer for mixed-mode designs," ISPD '05, pp. 221–223.
[24] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," ICCAD '05, pp. 48–55.
[25] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden, "Routability-driven placement and white space allocation," ICCAD '04, pp. 394–401.
[26] S. N. Adya and I. L. Markov, "Consistent placement of macro-blocks using floorplanning and standard-cell placement," ISPD '02, pp. 12–17.
[27] H.-C. Chen, Y.-L. Chuang, Y.-W. Chang, and Y.-C. Chang, "Constraint graph-based macro placement for modern mixed-size circuit designs," ICCAD '08, pp. 218–223, 2008.
[28] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang, and D. Liu, "Mp-trees: A packing-based macro placement algorithm for mixed-size designs," DAC '07, pp. 447–452, 2007.
[29] J. Z. Yan, N. Viswanathan, and C. Chu, "Handling complexities in modern large-scale mixed-size placement," DAC '09, pp. 436–441, 2009.
[30] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz, "The ISPD2005 placement contest and benchmark suite," ISPD '05, pp. 216–220.
[31] G.-J. Nam, C. J. Alpert, and P. Villarrubia, "The ISPD2006 placement contest and benchmark suite," ISPD '06, pp. 216–220.