High Performance Dummy Fill Insertion With Coupling and Uniformity Constraints

Yibo Lin, Bei Yu, *Member, IEEE*, and David Z. Pan, *Fellow, IEEE*

***Abstract*—In deep-submicron very large scale integration manufacturing, dummy fills are widely applied to reduce topographic variations and improve layout pattern uniformity. However, the introduction of dummy fills may impact the wire electrical properties, such as coupling capacitance. Traditional tile-based method for fill insertion usually results in very large number of fills, which increases the cost of layout storage. In advanced technology nodes, solving the tile-based dummy fill design is more and more expensive. In this paper, we propose a high performance dummy fill insertion framework based on geometric properties to optimize multiple objectives simultaneously, including coupling capacitance, density variations and gradient. The experimental results for ICCAD 2014 contest benchmarks demonstrate the effectiveness of our methods.**

***Index Terms*—Chemical mechanical polishing (CMP), design for manufacture, dummy fill insertion.**

I. Introduction

IN CURRENT very large scale integration manufacturing process, chemical mechanical polishing (CMP) is a planarizing technique widely used to satisfy the planarity requirements. Both mechanical and chemical methods are adopted in the CMP process. In spite of its popularity, the CMP-induced design challenge is its dependence on the features of device and interconnect in deep-submicron technology [1]. The quality of CMP patterns is highly-related to the uniformity of density distribution, and a predictable layout is desired for good CMP performance. To achieve uniform density distribution in a layout, dummy fills are inserted to increase the density of sparse regions. Even though there are specific design rules to restrict the side effects from the addition of dummy fills, it is still not enough to resolve all the problems in density variation or coupling capacitance. Hence powerful CAD tools for multiobjective fill insertion are still in great demand.

Manuscript received December 3, 2015; revised July 5, 2016; accepted November 1, 2016. Date of publication December 12, 2016; date of current version August 18, 2017. The preliminary version has been presented at Design Automation Conference in 2015. This paper was recommended by Associate Editor E. Young.

Y. Lin and D. Z. Pan are with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78731 USA (e-mail: yibolin@utexas.edu).

B. Yu was with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78731 USA. He is now with CSE Department, Chinese University of Hong Kong, Hong Kong.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2016.2638452

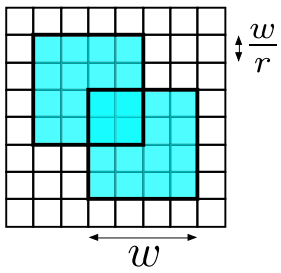

Fig. 1. Example of $w \times w$ windows with r^2 tiles.

The flow for layout density optimization can be generalized as two phases: 1) density analysis and 2) fill synthesis [2]. Density analysis first collects information of wire density and available fill regions and then calculates the amount of fills for the layout. In density analysis, regions with violations of density rules (lower/upper bound) are identified. It is usually based on fixed dissection where a layout is divided into windows with dimension of $w \times w$. Each window consists of $w/r \times w/r$ tiles as shown in Fig. 1. The extension to multiwindow and multilayer analysis was also proposed [3]. Fill synthesis determines how many fills to be inserted into the layout. Traditional methods usually aim at minimizing both the density variation and the number of fills through linear programming (LP) formulation [4], [5]. In these methods, layout is divided into windows and each window is further split into tiles for fill insertion. As the advancement of technology node, circuits become more and more complicated that LP-based method reaches their limitation due to problem sizes. In both [1] and [6], analysis of an example layout with 200 μm $\times$ 200 μm windows results in over 160K variables, thus the runtime becomes the bottleneck for LP-based method. Alternative approaches based on Monte Carlo or heuristic algorithms have been proposed. However, they are still lacking in either performance or speed [7]–[9]. The recent study from Liu *et al*. [10], [11] proposed an ultrafast fill insertion engine with an objective of minimizing density variation and total amount of fills.

Typically the density uniformity is measured with density variation, but it is highlighted that density gradient also plays an important role in post-CMP metal thickness [12] and mask distortion [13]. Chen *et al*. [14] mentioned that in modern process, the polishing pad during CMP can dynamically adjust pressure and rotation speed according to the density distribution. If the density changes slowly across neighboring windows, the CMP system is able to perform local adjustment

for better quality. In addition, the minimization of density variation does not necessarily contribute to small gradient because it focuses more on global uniformity rather than local uniformity. Chen *et al*. [12] gave an example of two density distributions with the same variation but yield totally different gradients. Therefore, gradient-based minimization should also be incorporated into the density uniformity optimization.

Furthermore, the introduction of dummy fills will incur additional coupling capacitance, causing performance degradation. The first integer LP (ILP)-based approach considering coupling capacitance was proposed by Chen *et al*. [15]. Xiang *et al*. [16] also studied the fill-induced coupling effects and proposed another ILP-based coupling constrained dummy fill algorithm to handle coupling capacitance. Their methods efficiently analyze density distribution and conduct fill optimization based on slots.

Besides runtime, another problem for traditional tile-based approaches lies in the requirement of large amount of fills for good uniformity, resulting in the difficulty for layout storage. Although current layout file standard like graphic database system II (GDSII) and open artwork system interchange standard (OASIS) can achieve good reduction in data volume, the problem is not solved due to the increasing complexity of circuits. In addition, large file size often leads to usability limitation and also increases data transfer time [17]. Ellis *et al*. [18] and Chen *et al.* [19] made early trials for data compression in GDSII and OASIS for tile-based fills. They try to utilize those array tokens in the file formats to describe those regular fill matrices so that the data is compressed with a hierarchical structure, such as AREF in GDSII. This approach is suitable to tile-based fills because those tiles are regularly aligned. But in our problem, as fills are described as arbitrary rectangles, which is not regular, it is difficult to wrap them in hierarchy.

To motivate the development of more effective dummy fill algorithms, ICCAD 2014 held a dummy fill contest [17] and released a suite of industrial benchmarks. Many conventional issues and emerging concerns were holistically modeled with *layer overlay*, *density variation*, *line hotspots*, *outlier hotspots*, and *file size*. A dummy fill optimizer with comprehensive optimization is desired. The definitions of these concepts will be discussed in detail in the next section (Section II). Besides metrics based on density variation from ICCAD 2014 contest, the density gradient should also be considered for the reason of better CMP quality and mask distortion as mentioned above. This is a brand new challenge for multiobjective optimization that includes density, coupling, and file size for arbitrary rectangular shapes of fills.

In this paper, we develop a high-performance framework for dummy fill insertion. Our main contributions can be summarized as follows.

1) The dummy fill insertion is based on geometric properties instead of tiles. Target density planning is done at window level and then candidate fills are generated under the guidance of target density.
2) The proposed algorithm optimizes for multiple objectives including density variation, density gradient, overlay, and so on.
3) We propose an ILP formulation and then its dual min-cost flow formulation to optimize overlay and layout density efficiently.
4) Experimental results show that our algorithms outperform the top teams by over 10% on ICCAD 2014 DFM contest benchmarks [17].

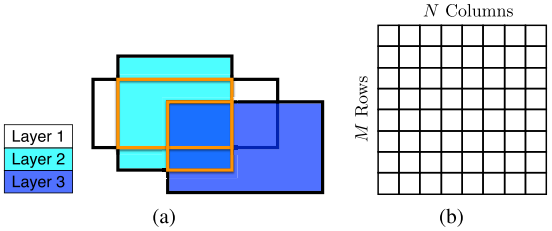

Fig. 2. Example of (a) overlay between layers (b) square windows for density analysis.

The rest part of this paper is organized as follows. Section II shows the definitions of related concepts and problem formulation. Section III explains the optimization algorithms in detail. Section IV presents the experimental results in different scores. In the end, we conclude this paper in Section V.

II. Preliminaries

A. Overlay Between Layers

The spatial overlaps between neighboring layers will lead to coupling capacitance, which is not desired in the layout design. Dummy fills are actually metal tiles, and their interaction with other signal wires will inevitably introduce additional capacitance, resulting in performance degradation. Therefore, it is necessary to avoid coupling capacitance during dummy fill insertion. In this paper, (as in ICCAD 2014 contest), coupling capacitance is evaluated with the amounts of overlay area between fills and their neighboring layers (including signal wires and fills) [20], [21]. As shown in Fig. 2(a), the enclosure regions of orange lines are counted to overlay area.

B. Layout Density

The performance of CMP is highly related to the layout density of a given window, and uniform density distribution contributes to high CMP quality [6]. In other words, density variation along windows is very critical, and is also the purpose of introducing dummy fills.

The distribution of window densities in this paper is evaluated with three scores: variation, line hotspots, and outlier hotspots [20], [21]. The whole layout is divided into $N \times M$ square windows as shown in Fig. 2(b). Variation is standard deviation of window layout densities, represented with σ. It aims at capturing the uniformity at layout level. Line hotspots are summation of column-based variation. For a layout shown in Fig. 2(b), we compute line hotspots as follows:

$$\text{lh} = \sum_{i=1}^{N} \sum_{j=1}^{M} \left| d(i,j) - \frac{\sum_{j=1}^{M} d(i,j)}{M} \right| \tag{1}$$

where $d(i,j)$ stands for the layout density at window (i,j). This score is used to verify variations along each column.

Outlier hotspots are summation of outlier deviations. This score is designed to evaluate the deviation of window densities outside 3σ range

$$\text{oh} = \sum_{i=1}^{N}\sum_{j=1}^{M} \max\left(0, \left|d(i,j) - \overline{d}\right| - 3\sigma\right) \tag{2}$$

where $\overline{d}$ denotes the average density over the layout, and σ indicates the variation.

All the three scores above evaluate different perspectives of the density distribution, and all are used in ICCAD 2014 contest. Variation may only provide a general view to the density map, while line hotspots and outlier hotspots collect more concrete information about it.

Besides the metrics on density variations, the uniformity should also include density gradient whose importance has been addressed in previous work [12], [13]. We define the density gradient $g(i,j)$ for a window (i,j) as follows:

$$g(i,j) = \max(|d(i,j) - d(i+1,j)|, |d(i,j) - d(i-1,j)| \\ |d(i,j) - d(i,j+1)|, |d(i,j) - d(i,j-1)|) \tag{3a}$$

$$\text{avg. grad.} = \sum_{i=1}^{N}\sum_{j=1}^{M} g(i,j) \tag{3b}$$

$$\text{max. grad.} = \max\ g(i,j),\ \ \forall i \in [1,N], j \in [1,M]. \tag{3c}$$

It describes the maximum density gap between current window and its four neighboring windows. The density gradient metric for the whole layout is measured by average density gradient in (3b) and max density gradient in (3c) across all the windows.

C. Problem Formulation

The addition of dummy fills needs a comprehensive view of the layout. That means both performance degradation and CMP quality should be taken into consideration. Hence, in this paper, the optimization will focus on a combined objective of layout overlay and density uniformity.

In ICCAD 2014 contest, all the metrics on layout overlay and density uniformity are normalized to scores. But density gradient is not included in the metrics. Therefore, we will show the values of density gradient separately without normalization.

Given an input layout with initial fill regions and wire densities across each window, we insert dummy fills to maximize the following score and minimize density gradient:

$$\text{score} = \alpha_{\text{ov}} s_{\text{ov}} + \alpha_{\sigma} s_{\sigma} + \alpha_{\text{lh}} s_{\text{lh}} + \alpha_{\text{oh}} s_{\text{oh}} + \alpha_{\text{fs}} s_{\text{fs}} \tag{4}$$

where $s_{\text{ov}} = f_{\text{ov}}(\sum_{l\in L} \text{ov}(l))$ denotes total overlay score for all layers in set L; $s_{\sigma} = f_{\sigma}(\sum_{l\in L} \sigma(l))$ stands for total variation score for all layers; $s_{\text{lh}} = f_{\text{lh}}(\sum_{l\in L} \text{lh}(l))$ means total line hotspot score for all layers; $s_{\text{oh}} = f_{\text{oh}}(\sum_{l\in L} \sigma(l) \cdot \sum_{l\in L} \text{oh}(l))$ represents total outlier score for all layers; $s_{\text{fs}} = f_{\text{fs}}(\text{fs})$ is file size score. Function f shows how the score is calculated with its corresponding variables and it can be generalized to

$$f(x) = \max\left(0, 1 - \frac{x}{\beta}\right) \tag{5}$$

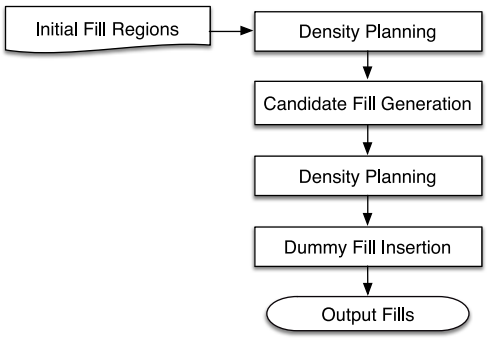

Fig. 3. Our dummy fill insertion flow.

TABLE I
NOTATIONS USED IN FILL INSERTION PROBLEM

s_m, w_m, a_m	DRC rule for min. spacing, width and area
a_w	Window area
d_g	Density gap (normalized to area)
$d_t(l)$	Target density on layer l
$d_t(i,j)$	Target density for window (i,j)
ov	Overlay
w_i, h_i	width and height of a candidate fill
L	Set of layers
$F(l)$	Set of fills on layer l
$O(l)$	Set of fill overlays between layer l and $l+1$
$P(l)$	Set of fill pairs with spacing rule violations
$e(i,j)$	Euclidean distance between fills i and j, $\forall(i,j) \in P(l)$
x_i^l, x_i^h	left and right bound of fill i
y_i^l, y_i^h	lower and upper bound of fill i

where α_{ov}, α_{σ}, α_{lh}, α_{oh}, α_{fs}, and β are benchmark-related parameters in the contest, which are defined in Table II of Section IV.

We can see that overlay, variation, line hotspots and outlier hotspots in each layer are added up for scores. File size score s_{fs} is introduced to reduce the difficulty for layout storage. In ICCAD 2014 contest, GDSII format is used as a standard I/O format. As mentioned above, the metric of density gradient does not apply to the normalizing function in (5).

III. DUMMY FILL INSERTION ALGORITHMS

The flow of our algorithm is summarized in Fig. 3. After reading the input fill regions, we need to perform polygon decomposition and assign fill regions to each window. After the available fill regions for each window are calculated, the density distribution is ready for density planning and target densities can be obtained. Then we generate candidate fills according to density demands and overlay cost. After this, another round of density planning is performed due to the inconsistency between candidate fills and initial plans. In the end, dummy fills are inserted with proper sizes to improve overlay and density variations. Table I gives the definitions of symbols used in the following explanation.

A. Polygon Decomposition

It is very important to decompose input rectilinear polygons into rectangles, because rectangles are usually easier to process and final fills should also be rectangles. The quality of final fills is also related to the performance of polygon decomposition. Our polygon decomposition algorithm is extended from the efficient polygon-to-rectangle conversion algorithm from [22]. The procedure can also be explained as a scanning line move from bottom to top and cut rectangles one-by-one. It is different from the edge-based decomposition in [10] since we do not need to differentiate the corners for duplicated points. We extend the algorithm with one horizontal scanning line and another vertical scanning line. During each cut, two candidate rectangles will be generated from the scanning lines where the better rectangle is chosen. There are various criteria in choosing the rectangles, such as area and aspect ratio. We find that a suitable criterion is essential to work together with candidate fill generation in Section III-D for fewer fills. Intuitively larger and fewer rectangles in general lead to better performance in terms of file size. It is also observed that fewer rectangles in this step do not always contribute to fewer fills due to density requirement. More details will be discussed empirically in Section IV-B.

B. Target Density Planning

Due to the complexity of objective function and large amounts of windows in a layout, direct optimization over the locations and sizes of dummy fills across all windows is very expensive and thus time consuming. Density planning is rather important, since it can serve as a guidance for candidate fill region generation (Section III-D) and final fill insertion (Section III-E) in each window. Good density plan is also capable of reducing the problem size, and eventually contributes to the reduction of run-time.

With the information of feasible fill regions, it is possible to calculate the density bound of each window. The lower bound for the window density is the wire density in that window, while its upper bound is usually related to the area of its fill regions.

During this step, we do not consider overlay penalty, so the goal of density planning is to maximize the density score, the summation of variation score, the line hotspot score, and the outlier hotspot score. It is a function of the density of each window in each layer. Actually this objective considers multiple windows in multiple layers. To simplify the analysis, we assume that in all the following notations, the information of layers is implicitly included. For each window, its density $d(i,j)$ is bounded by existing wire density and available fill regions. Let the lower and upper bound of $d(i,j)$ be $l(i,j)$ and $u(i,j)$, respectively.

Definition 1 (Target Layout Density): A density value for one layer represented by d_t. Its relation with $d(i,j)$ can be shown as follows:

$$d(i,j) = \begin{cases} l(i,j), & \text{if } d_t < l(i,j) \\ u(i,j), & \text{if } d_t > u(i,j) \\ d_t, & \text{other.} \end{cases} \tag{6}$$

The density planning problem is now transformed to find the best d_t for maximum density score. To find the best d_t, we can analyze the following two cases.

Case I: If the ranges of $d(i,j)$ for all windows are large enough, we can get a trivial solution that is optimal by setting

$$d_t = \max\left(l(k,n)\right), \quad \forall k \in 1,2,\ldots,N, n \in 1,2,\ldots,M. \tag{7}$$

It means that the target density d_t is equal to the largest wire density throughout the layout. In this way, an ideal uniform distribution is achieved since densities in all windows are the same.

Case II: Not all windows are able to get to target layout density. For example, the largest wire density may be 0.9, while a window can only achieve a density as high as 0.7. This kind of situation will occur when $\exists(i,j)$ satisfies

$$u(i,j) < \max\left(l(k,n)\right). \tag{8}$$

Then the target density for window (i,j) can only be set to $u(i,j)$ instead of $\max(l(k,n))$. In this case, we simply search all combinations of target layout densities for all layers with small steps and then choose the best one. The search range for d_t can be limited to values between $\max(l(k,n))$ and $\min(u(k,n))$. The run-time for this step is still very fast due to the simplicity of each calculation and limited number of layers.

C. Density Gradient Adjusting

The target density planning approach in Section III-B aims at maximizing total density score, but it does not take density gradient into consideration. In order to keep the planned density score and meanwhile optimize density gradient, we choose to locally adjust gradient for each layer without large perturbation to the existing results. The problem is formulated into a quadratic program as follows:

$$\min \sum_{\substack{1\le i\le N,\\ 1\le j\le M}} d_g(i,j)\cdot d_g(i,j) + \varepsilon \sum_{\substack{1\le i\le N,\\ 1\le j\le M}} g(i,j)\cdot g(i,j) \tag{9a}$$

$$\text{s.t. } d_g(i,j) = d(i,j) - d_t \tag{9b}$$

$$g(i,j) \ge d(i,j) - d(i\pm 1,j) \tag{9c}$$

$$g(i,j) \ge d(i\pm 1,j) - d(i,j) \tag{9d}$$

$$g(i,j) \ge d(i,j) - d(i,j\pm 1) \tag{9e}$$

$$g(i,j) \ge d(i,j\pm 1) - d(i,j) \tag{9f}$$

$$l(i,j) \le d(i,j) \le u(i,j). \tag{9g}$$

The objective consists of two parts. The first summation computes the quadratic density gap between current window density and target layout density. The second summation computes the quadratic gradient for all the windows. We want to minimize total density gradient and the amounts of deviation from target layout density. Coefficient ε is used to balance the priority between two parts, and it is set to 10 in the experiment. The density gap for each window is calculated in constraint (9b) where d_t denotes the target layout density obtained from Section III-B. Constraints (9c) to (9f) compute the gradient defined in (3a). Since the density gradient for a window is defined as the maximum density gap between the

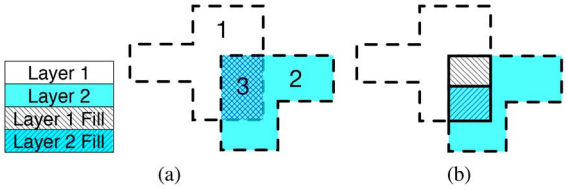

Fig. 4. Case I zero overlay. Example of (a) fill regions in neighboring layers in a window (b) fill solution without overlay.

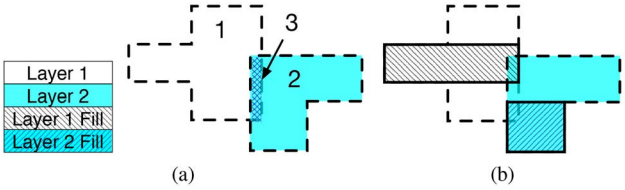

Fig. 5. Case II nonzero overlay. Example of (a) fill regions in neighboring layers in a window (b) fill solution with overlay.

window and its four neighbors, it introduces four sets of constraints for each window. At the same time, it is necessary to enumerate all the combinations of $a-b$ and $b-a$ to mimic the absolute operation $|a-b|$ in the density gap computation. So each set contains two constraints. The constraints from constraints (9c) to (9f) represent the eight constraints for each window.

Problem (9) is a mathematical program with a quadratic objective function subject to linear equality and inequality constraints; in other words, it is a quadratic program. If we rewrite the objective as a general expression, $(1/2)d^T\Phi d$, we can see that the Hessian matrix Φ is positive semidefinite, which means the quadratic program (9) is a convex quadratic programming problem. It can be solved in polynomial time by interior point method and the constraints can be integrated into the objective by barrier functions. By solving the quadratic program (9), we can obtain locally adjusted density distribution with tradeoffs between density gradient and density scores. The solution of $d(i,j)$ for each window will be used as the target density for each window (i,j).

D. Candidate Fill Region Generation

In this section, we generate candidate fills in each window under the guidance of target density and at the same time minimize overlay. After this step, with all the candidate fills, the density in a window will be no less than its target density. So the output of this step is an upper bound of fills which needs further optimization in the next section to reduce density variation. For convenience, we only use two layers to explain our strategies.

To optimize overlay area, it is necessary to consider fill regions in multiple layers simultaneously. The problem can be analyzed from two cases.

Case I (Zero Overlay): Fig. 4(a) shows one case of fill regions for two neighboring layers. We can divide these fill regions into three parts, marked as 1, 2, and 3 in the figure. In region 1, there is only one empty space in layer 1 and the same space in layer 2 should contain signal wires. If dummy fills are inserted to layer 1 in this region, it is likely to have overlay with wires in layer 2, since in this region there is wire distribution with a certain density. In region 2, the condition is similar to region 1 and layer 1 contains signal wires, while it is empty in layer 2. In region 3, spaces in both layers are free of signal wires. There is no need to consider overlay with signal wires any more in this region. But we should be aware of overlay between fills in different layers. If we only insert fills to region 3, it is possible to achieve zero overlay, as shown in Fig. 4(b).

Algorithm 1 Candidate Fill Region Generation

Require: Feasible fill regions of all layers in a window.
Ensure: Generate candidate fills with small overlay under density constraints.

Assume layer numbers in layer set L are 1, 2, ..., N;
Define $fr(l)$ as the fill region in Layer l;
Define $d_t(l)$ as the target density in Layer l;
Define $d_w(l)$ as the wire density in Layer l;
Define $d_g(l)$ as the density gap in Layer l;
Define $d(l)$ as the layout density of Layer l;
Define a_w as the area of the window;
Define λ as a parameter, $\lambda \geq 1$;
for l = odd number of layers in L **do**
    $fr_s = intersect(fr(l), fr(l+1))$;
    $d_g(l) = d_t(l) - d_w(l)$;
    $d_g(l+1) = d_t(l+1) - d_w(l+1)$;
    if $area(fr_s) \geq (d_g(l) + d_g(l+1)) \cdot a_w$ **then**
        Assign fills to Layer l until $d(l) \geq \lambda \cdot d_t(l)$;
    else
        Sort fills in $fr(l)$ by area;
        Assign fills to Layer l until $d(l) \geq \lambda \cdot d_t(l)$;
    end if
end for
for l = even number of layers in L **do**
    $d_g(l) = d_t(l) - d_w(l)$;
    Sort fills in $fr(l)$ by quality score q;
    Assign fills to Layer l until $d(l) \geq \lambda \cdot d_t(l)$;
end for

Case II (Nonzero Overlay): Fig. 5(a) shows that region 3 may be too small to meet the density requirements. In this case, if we still limit fills inside region 3, there will be inevitable deterioration in density variation. So the extension of fills to regions 1 and 2 becomes a necessity and small fill-to-fill overlay is also allowed for better density distribution. We evaluate the quality of a candidate fill using a score considering its overlay with fills in lower and upper layers and its area, shown as

$$q = -\frac{\text{fill overlay}}{\text{fill area}} + \gamma \cdot \frac{\text{fill area}}{\text{window area}} \tag{10}$$

where γ is a parameter, and we set it to 1 in the experiment. Fills with high quality scores have priority in candidate fill selection.

Since the number of layers is usually larger than 2, the actual implementation combines the previous ideas, which is summarized in Algorithm 1. It is impossible to calculate the overlay between fills before any fill is inserted to the

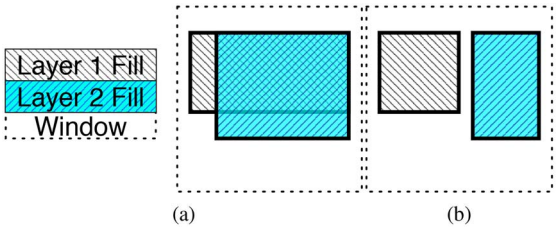

Fig. 6. (a) Example of candidate fills with large overlay in a window (b) resolved by splitting candidate fills.

lower/upper layer, so we determine candidate fills in odd layers first as a reference for even layers. Function *intersect* returns the shared fill region between $fr(l)$ and $fr(l+1)$. If the shared fill region fails to meet the density requirements in layer l and layer $l+1$, fill qualities are simply evaluated with the size of the fill. After the generation of candidate fills in odd layers, we select fills in even layers according to their quality score q with (10). λ is a parameter to control how many fills to generate for each layer. It is no less than 1 because the amount of fills should be large enough to achieve target density.

Although Algorithm 1 considers both density benefit and overlay cost, it cannot handle some special cases, shown as Fig. 6(a). Sometimes there are very few choices for candidate fills in a window and all the rectangular regions have large overlay with neighboring layers. In such a case, it is difficult to generate candidate fills with high quality through a selection scheme like Algorithm 1. The overlay impacts from such cases become unignorable when the fill regions in a window consists of only several *giant fills* (occupy more than a_t of the window area). Actually a better strategy is to split those candidate fills and choose their partial components to avoid overlay, shown as Fig. 6(b).

Such observation results in a post refinement step in Algorithm 2. This step takes candidate fills in a window generated from previous step as input and aims at reducing overlay for those giant fills by splitting them. The algorithm starts from even layers in line 5 and iterates through all the giant candidate fills in the layer from lines 6 to 20. For a giant candidate fill $f(l)$ in layer l, its overlay area between its neighboring layer $l+\delta$ is computed in line 11. We first check the overlay with layer $l-1$ and then layer $l+1$, so δ can be -1 and 1. If more ov_t of the candidate fill has overlap with fills in neighboring layers, consider to split it for better overlay, which corresponds to lines 12 to 18. Simply splitting fill $f(l)$ might not be enough to remove enough overlay regions. The fill $f(l+\delta)$ of largest overlay area with $f(l)$ in layer $l+\delta$ is extracted as the other candidate for splitting. Then both $f(l)$ and $f(l+\delta)$ are split equally by half and we can replace them with the parts with smallest overlay.

Algorithm 2 performs very well for overlay removal, but it reduces the available area of candidate fills. The parameters a_t and ov_t are two threshold values for triggering overlay refinement, which are set to 0.05 and 0.6, respectively. We will discuss the effects of these parameters in Section IV. Sometimes it results in the drop of density upper bound below target density and eventually deteriorates final density uniformity. This side effect is more severe when it comes to density gradient that focuses on local uniformity. Special optimization is needed for density gradient. However, as the gradient of a window is computed from both its own density and that of its neighboring windows, it is difficult to determine the neighboring window densities before the candidate fill generation of all windows. To solve this issue, we perform an incremental optimization after all windows finish Algorithms 1 and 2, i.e., extract top $\kappa\%$ windows with worst density gradient and regenerate fills with Algorithm 1 to satisfy target density. We observe that this approach is very effective to reduce maximum gradient across all windows. The tradeoffs for the value of κ will be further discussed in Section IV.

Algorithm 2 Overlay Refinement for Candidate Fills

Require: Candidate fills of all layers in a window.
Ensure: Refine candidate fills to reduce overlay

```
1:  Assume layer numbers in layer set L are 1, 2, ... N;
2:  Define F(l) as the set of fills generated in Layer l;
3:  Define f(l) as a fill in Layer l;
4:  Define OV(l1, l2) as the overlay of Layer l1 and Layer l2;
5:  for l = even number of layers in L do
6:      for f(l) in F(l) do
7:          if area(f(l)) < a_t × a_w then
8:              Continue;
9:          end if
10:         for δ ∈ {−1, 1} do
11:             OV(l, l + δ) = intersect(f(l), F(l + δ));
12:             if OV(l, l + δ) ≥ ov_t × area(f(l)) then
13:                 Find f(l+δ) with largest overlay with f(l);
14:                 Split f(l) into two equal parts;
15:                 Split f(l + δ) into two equal parts;
16:                 Choose parts of f(l) and f(l + δ) for
17:                 minimum overlay and replace original fills;
18:             end if
19:         end for
20:     end for
21: end for
```

E. Dummy Fill Sizing

In this section, we will determine the sizes of fills to further reduce density variation and overlay in an efficient way.

1) Mathematical Formulation: So far we have a set of candidate fills as an upper bound, but there are still DRC errors and maybe large deviations between fill density and target density. Further steps are necessary to fix spacing rule violations and optimize density variation together with overlay. In this step, the final sizes of fills are determined by shrinking candidate fills.

Given a set of candidate fills in a window, determine the dimension of fills under DRC constraints to minimize overlay area and density variation. Our problem can be described with

$$\min_{\substack{x_i^l, x_i^h \\ y_i^l, y_i^h}} \sum_{l \in L} d_g(l) + \eta \cdot \sum_{l \in L} \text{ov}(l, l+1) \tag{11a}$$

$$\text{s.t.} \quad d_g(l) = \left| \sum_{i \in F(l)} w_i \cdot h_i - d_t(l) \cdot a_w \right|, \ l \in L \tag{11b}$$

$$\text{ov}(l, l+1) = \sum_{i \in O(l)} w_i \cdot h_i, \ l \in L \tag{11c}$$

$$w_i = x_i^h - x_i^l, h_i = y_i^h - y_i^l \tag{11d}$$

$$w_i \geq w_m, h_i \geq w_m \tag{11e}$$

$$w_i \cdot h_i \geq a_m \tag{11f}$$

$$e(i,j) \geq s_m, \quad \forall (i,j) \in P(l), l \in L \tag{11g}$$

$$x_i^l, x_i^h, y_i^l, y_i^h \in Z$$

where η is a weight for overlay cost, which is 1 in the experiment. The objective tries to minimize a combination of density gap and weighted overlay. In Constraint (11b), density gap d_g is defined as the difference between the area of fills and the target fill area (derived from target fill density). Constraint (11c) defines the overlay area ov. Constraints (11e)–(11g) state required DRC rules, such as minimum width, minimum area and minimum spacing. $e(i,j)$ is the Euclidean distance between fills i and j. Equation (11) defines a nonconvex problem, so it is very expensive to solve it.

2) Problem Relaxation: Previous formulation contains multiplication operations between variables in two directions, which results in nonconvex features. We can alternatively fix one direction when optimizing the other one, and then the problem is relaxed to a linear program. Without loss of generality, we set vertical direction fixed and all the variables related to that direction become constants. Then constraints (11b) and (11c) can be relaxed to

$$d_g(l) = \left| \sum_{i \in F(l)} w_i \cdot h_{i0} - d_t(l) \cdot a_w \right|, \ l \in L \tag{12}$$

$$\text{ov}(l, l+1) = \sum_{i \in O(l)} w_i \cdot h_{i0}, \ l \in L \tag{13}$$

where h_{i0} is the initial height of candidate fills from Section III-D.

Constraints (11e) to (11f) can be merged into one equation

$$w_i \geq \max\left(w_m, \frac{a_m}{h_{i0}}\right). \tag{14}$$

Constraint (11g) will only exist for pairs of fills that are very close to each other. For these fill pairs, following constraint will force fills to keep enough space in horizontal direction:

$$e^h(i,j) \geq s_m. \tag{15}$$

With (12)–(15), a relaxed problem solvable with ILP is formed for horizontal direction as follows:

$$\min \ \sum_{l \in L} d_g(l) + \eta \times \sum_{l \in L} \text{ov}(l, l+1) \tag{16a}$$

$$\text{s.t.} \ \ d_g(l) = \left| \sum_{i \in F(l)} w_i \times h_{i0} - d_t(l) \cdot a_w \right|, \quad l \in L \tag{16b}$$

$$\text{ov}(l, l+1) = \sum_{i \in O(l)} w_i \times h_{i0}, \quad l \in L \tag{16c}$$

$$x_i^h - x_i^l \geq \max\left(w_m, \frac{a_m}{h_{i0}}\right), \quad \forall i \in F(l), l \in L \tag{16d}$$

$$e(i,j) \geq s_m, \quad \forall (i,j) \in P(l), l \in L \tag{16e}$$

$$l_i^l \leq x_i^l \leq u_i^l, \ l_i^h \leq x_i^h \leq u_i^h \tag{16f}$$

$$x_i^l, x_i^h \in Z$$

where $d_g(l)$ denotes the density gap in layer l between current fill area and target fill area as defined in Table I; $\text{ov}(l, l+1)$ denotes the overlay area between layer l and $l+1$; $e(i,j)$ denotes the Euclidean distance between fills i and j that result in spacing rule violations. We alternatively optimize the problem in horizontal and vertical directions. In other words, ILP will be run iteratively. During each iteration, variables are bounded to a certain range to ensure performance, i.e., $l_i^l \leq x_i^l \leq u_i^l$ and $l_i^h \leq x_i^h \leq u_i^h$. These ranges need to be updated according to the results of each iteration.

3) Dual Min-Cost Flow Formulation: The relaxed problem in previous section may still suffer from high run-time penalty when the problem size is large, as ILP problem is generally NP-hard to solve. Here we show that the formulation is able to achieve further speedup with dual min-cost flow.

Equation (12) contains an absolute operation which ensures the fill density will converge to target density. Since in this stage, fills can only shrink in each iteration. It is possible to relax the problem by removing the absolute operation. We are always able to calculate the upper bound of fill area by taking the current sizes of fills. If the upper bound is smaller than $d_t(l) \cdot a_w$, then $|\sum_{i \in F(l)} w_i \cdot h_{i0} - d_t(l) \cdot a_w|$ is equivalent to $d_t(l) \cdot a_w - \sum_{i \in F(l)} w_i \cdot h_{i0}$. On the other hand, if current fill density is larger than target density, we can still remove it by reducing the shrinking steps for fills in each iteration. It should be noted that after current iteration fill density drops below target density, we will switch to the first case and hence the fill density cannot keep getting away from target density.

Then the relaxed problem in Section III-E2 can be written in a generalized manner without any absolute operation

$$\min_{x_i} \ \sum_{i=1}^{N} c_i x_i \tag{17a}$$

$$\text{s.t.} \ \ x_i - x_j \geq b_{ij}, \ (i,j) \in E \tag{17b}$$

$$l_i \leq x_i \leq u_i, \ i = 1, 2, \ldots, N \tag{17c}$$

$$x_i \in Z.$$

Equation (17) is a linear program with only differential constraints and bounded variables, which can be transformed to a dual min-cost flow problem [23]. Min-cost flow problem is a relative mature field with very fast algorithms. Thus it is often adopted in the physical design flow [24]–[26].

Our problem can be transformed to the following typical dual min-cost flow format:

$$\min_{y_i} \ \sum_{i=0}^{N} c_i' y_i \tag{18a}$$

$$\text{s.t.} \ \ y_i - y_j \geq b_{ij}', \ (i,j) \in E' \tag{18b}$$

$$y_i \in Z$$

where

$$x_i = y_i - y_0, \quad i = 1, 2, \ldots, N \tag{19a}$$

TABLE II
ICCAD 2014 BENCHMARK STATISTICS

Design	#P	#L	File size	Overlay*		Variation*		Line*		Outlier*		Size*		Run-time*		Memory*	
				α	β	α	β	α	β	α	β	α	β	α	β	α	β
s	382K	3	48M	0.2	79154	0.2	0.077	0.2	11.758	0.15	0.014	0.05	32	0.15	60	0.05	1024
b	8.1M	3	1.1G	0.2	6111303	0.2	0.517	0.2	3578	0.15	22.801	0.05	2048	0.15	600	0.05	32768
m	31.8M	3	2.2G	0.2	10276835	0.2	0.53	0.2	6052	0.15	27.56	0.05	1536	0.15	1200	0.05	32768

$$c'_i = \begin{cases} c_i, & i = 1, 2, \ldots, N \\ -\sum_{i=1}^{N} c_i, & i = 0 \end{cases} \tag{19b}$$

$$b'_{ij} = \begin{cases} b_{ij}, & (i,j) \in E \\ l_i, & i = 1, 2, \ldots, N, j = 0 \\ -u_i, & i = 0, j = 1, 2, \ldots, N. \end{cases} \tag{19c}$$

Lemma 1: Equations (17) and (18) are equivalent.

The proof is omitted here for brevity. The key idea is to introduce a new variable y_0 to convert lower and upper bound constraints to differential constraints, which is a common technique in convex optimization [27].

The corresponding min-cost flow problem can be written as follows:

$$\min \sum_{(i,j)\in E'} -b'_{ij} f_{ij}, \tag{20a}$$

$$\text{s.t.} \sum_{(j,i)\in E'} f_{ji} - \sum_{(i,k)\in E'} f_{ik} = -c'_i, i = 0, \ldots, N \tag{20b}$$

$$f_{ij} \geq 0, (i,j) \in E'. \tag{20c}$$

So far the dual min-cost flow problem can be mapped to a graph with $N + 1$ nodes. The supply of node i is c'_i and for each (i, j) pair in E', an edge starting from node i to node j with a cost of $-b'_{ij}$ is inserted to the graph. The edge capacity is set to infinity. Final solution of y can be obtained from the potential of each node and x can be derived from (19a).

IV. EXPERIMENTAL RESULTS

Our algorithms were implemented in C++ and tested on an 8-Core 3.40 GHz Linux server with 32GB RAM. The results of ICCAD 2014 contest top three teams are tested on a 2.6 GHz machine with 64GB RAM. Related results and executables are released at link (http://www.cerc.utexas.edu/utda/download/DFI/index.html). LEMON [28] is used as the min-cost flow solver. GUROBI [29] is used as the quadratic programming solver. Our solutions have been verified by the contest organizer [17]. Statistics about benchmarks are listed in Table II. The total amount of input polygons for fill insertion is shown as "#P." The total number of layers is shown as "#L." It also contains coefficients to calculate following scores. Since all the metrics are represented in scores, we add "*" to differentiate them from their original values. Overlay* denotes the overlay score stated in Section II; Variation* represents variation score; Line* is the score for line hotspots; Outlier* is the score for outlier hotspots; Size* is the score for the volume of solution GDSII files, which is the standard input and output format in the contest; Run-time* stands for run-time score; Memory* denotes memory score and it measures the peak memory usage during the execution. All the scores above are calculated with (5). According to ICCAD 2014 contest [17], Score is the weighted summation of all the scores above. Quality is similar to testcase score but excludes run-time score and memory score. It measures the quality of solutions. α and β respect to the coefficients in (4) and (5). Run-time score and memory score are also calculated with (5). The fills inserted must lie in specific regions given inputs and subject to DRC rules, i.e., minimum metal width and space of 32 nm and minimum metal area of 4800 nm^2. The window size for fill insertion is set to 20×20 μm^2 according to the contest. All the scores are obtained from the contest organizer [17] except run-time and peak memory usage which are measured in the server. To evaluate the effectiveness of our algorithm, we compare our results with top three teams in the contest. In the result tables, our dummy fill insertion algorithm for variation minimization is shown as "DFI." Our algorithm with gradient minimization is shown as "G-DFI."

Table III lists the evaluation results for our algorithm and top three teams from the ICCAD 2014 contest. It is shown that our fill insertion engine produces both the highest quality scores and overall scores for all the testcases. On average, the quality score of DFI is 13% better and the overall score is about 10% higher than the top team in the contest. G-DFI further improves the quality and efficiency of DFI such that eventually it outperforms the top team by 22% in quality score and 24.6% in overall score.

According to score calculation, density related scores (variation, line hotspots and outlier hotspots) take 45%, and overlay score takes 20%. From Table III we can see that our overall density scores are among the highest. For example, for benchmark *m* we get the best density scores, though the overlay score is a little bit lower than top three teams. We ascribe it to the comprehensive density analysis and simultaneous control over density and overlay during fill insertion. Our density planning directly works on the density score metrics including variation, line hotspots and outlier hotspots, for optimizing any of metrics individually may not result in a good overall score. Furthermore, we consider overlay in both candidate fill generation and dummy fill sizing under the guidance of density planning. So the overlay can be reduced without degrading the density scores too much. We can also see that our size score is high, which means the number of fills inserted is much smaller than others. Although the contest 1st team gets even higher size score, their solutions suffer from larger density variation. When the size of layout file increases, it takes longer time for reading and writing. In summary, the results demonstrate that our algorithms produce more balanced solutions which can handle overlay and density requirements simultaneously.

We further compare our gradient aware algorithm G-DFI with DFI together with average and maximum density gradient metrics, shown as Table IV. *Grad. Avg/Max* is computed

TABLE III
EXPERIMENTAL RESULTS ON ICCAD 2014 BENCHMARK FOR DFI

Design	Team	Overlay*	Variation*	Line*	Outlier*	Size*	Run-time*	Memory*	Quality	Score
s	1st	0.743	0.636	0.733	1.000	0.976	0.877	0.885	0.621	0.797
	2nd	0.743	0.909	0.967	0.975	0.103	0.846	0.831	0.675	0.844
	3rd	0.613	0.985	0.990	1.000	0.158	0.842	0.429	0.676	0.823
	DFI	0.723	0.948	0.979	0.994	0.887	0.872	0.818	0.724	0.895
	G-DFI	0.719	0.977	0.989	1.000	0.938	0.969	0.931	**0.734**	**0.926**
b	1st	0.748	0.368	0.364	0.871	0.924	0.515	0.891	0.473	0.594
	2nd	0.841	0.381	0.534	0.000	0.053	0.513	0.828	0.354	0.472
	3rd	0.576	0.485	0.601	0.000	0.568	0.554	0.339	0.361	0.461
	DFI	0.685	0.499	0.470	0.953	0.765	0.351	0.852	0.512	0.607
	G-DFI	0.521	0.675	0.670	0.998	0.837	0.886	0.949	**0.564**	**0.745**
m	1st	0.598	0.462	0.486	0.204	0.941	0.556	0.845	0.387	0.513
	2nd	0.668	0.460	0.618	0.000	0.000	0.780	0.761	0.349	0.504
	3rd	0.510	0.509	0.689	0.000	0.807	0.748	0.772	0.382	0.533
	DFI	0.493	0.643	0.766	0.088	0.905	0.750	0.786	0.439	0.591
	G-DFI	0.425	0.607	0.593	0.967	0.896	0.932	0.917	**0.515**	**0.701**

TABLE IV
COMPARISON OF GRADIENT BETWEEN DFI AND G-DFI

	Team	Grad. Avg/Max
s	DFI	0.004/0.023
	G-DFI	0.001/0.006
b	DFI	0.212/1.127
	G-DFI	0.096/0.539
m	DFI	0.141/1.395
	G-DFI	0.097/0.658

TABLE V
COMPARISON BETWEEN G-DFI AND [11]

	Team	Variation*	Line*	Outlier*	Size*	Quality
s	[11]	1.000	1.000	1.000	0.952	**0.776**
	G-DFI	0.977	0.987	1.000	0.938	0.768
b	[11]	0.528	0.701	0.000	0.778	0.426
	G-DFI	0.675	0.670	0.998	0.837	**0.547**
m	[11]	0.876	0.939	0.838	0.385	0.548
	G-DFI	0.607	0.593	0.967	0.896	**0.554**

through the summation of average and maximum gradient values across all layers. Since the official evaluation script does not contain gradient metrics and it is not released either, we try our best to estimate our results as close as possible to the official values. Comparing the scores in Table III with that in Tables IV and V for G-DFI, all the density related metrics are very close. The slight difference comes from different starting coordinates of windows which are determined by the leftmost lowest shape in the official script. The leftmost lowest shape can be either a dummy fill or an initial routing segment. As the initial routing is not accessible, we use (0, 0) as the leftmost lowest coordinates as the starting coordinates of windows. The overlay includes fill-to-fill and fill-to-routing overlaps in the official script. Again, we are not able to access initial routing and what we have is the density distribution of routing layers in each window. So we assume a uniform distribution of routing segments in each window for overlay calculation. While it is true that our evaluation script produces different result from the official one, the fidelity and trends are maintained. So the comparison is fair according to the same baseline and script.

In Table IV, we can see the maximum gradient of G-DFI is 59.6% better than DFI and the average gradient also has 53.6% improvement. The local adjustment for gradient minimization in Section III-C helps to reduce both average and maximum gradients, while the density scores are maintained. Actually even the density scores turn out to be slightly better, we ascribe the reason to the incremental optimization process after Algorithm 2 which contributes to the removal of large outliers. But this process also results in more overlay because candidate fills with large overlay are allowed in those critical windows for better gradient. This explains the drop of overlay score. Eventually, the testcase quality scores are improved by 8.2%. We also reduce runtime and memory usage by replacing smart pointers with primitive pointers so that faster access is possible. The new runtime values for benchmark *s*, *b*, and *m* are 1.8, 68.1, and 81.4 s including file IO, respectively. The peak memory usage values are 70.7, 1662.1, and 2732.7 MB, respectively. According to the results in Table III, the absolute overlay areas for the three benchmarks in G-DFI are 0.022, 2.9, and 5.9 mm^2, respectively.

For the comparison with [11] in Table V, we can see that G-DFI outperforms in quality scores about 6.8%. Our quality scores on benchmark *b* and *m* are much better than theirs, while the scores on benchmark *s* are very close. In benchmark *b* and *m*, our outlier scores are much higher than theirs. We ascribe the improvement to the incremental optimization mentioned at the end of Section III-D that often helps remove outliers by regenerating candidate fills for windows with poor gradients.

A. Runtime and Memory Scalability of Algorithm

We also study the scalability of our algorithm in runtime and memory, as shown in Fig. 7. We create large benchmarks by duplicating benchmark *b* from 1× to 16×, so the file size of the input benchmarks grows from 1.1 to 17GB. It can be seen that both runtime and memory increase linearly with the sizes of input benchmarks. In the 1× benchmark, the runtime

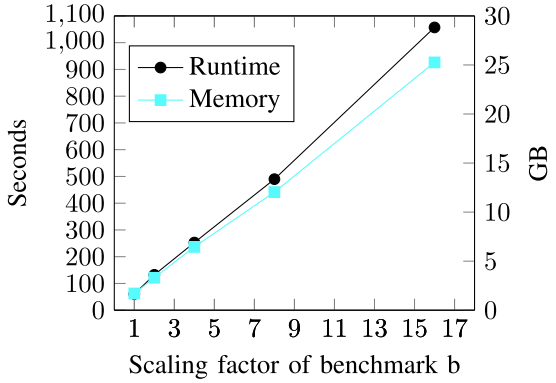

Fig. 7. Experiment on runtime and memory scalability by scaling benchmark b from 1× to 16×.

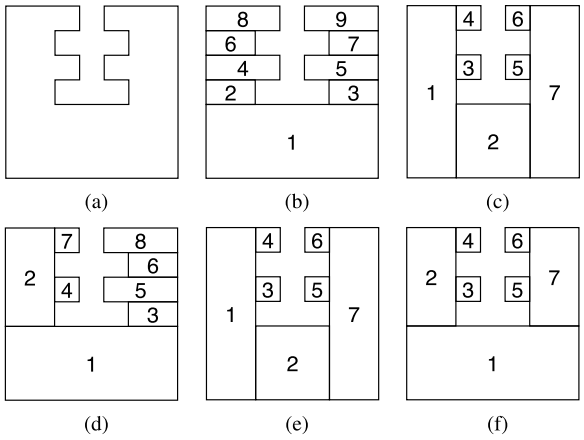

Fig. 8. Example of polygon decomposition solutions with various criteria (a) original polygon and solution of (b) *H*, (c) *V*, (d) LA, (e) SA, and (f) AR.

is less than 100 s and peak memory is around 1.7GB, while in the 16× benchmark, the runtime is less than 1100 s and peak memory is around 25GB. For benchmark b, the runtime distribution of each step is as follows: polygon decomposition takes around 16% of runtime, two rounds of density planning take 37% of runtime, and dummy fill sizing takes 34% of runtime.

B. Criteria for Polygon Decomposition

As mentioned in Section III-A, various heuristics can be applied to polygon decomposition to facilitate candidate fill generation. Since we have both horizontal and vertical scanning lines, the following five criteria are compared.

1) *H:* Always choose the rectangle cut by the horizontal scanning lines.
2) *V:* Always choose the rectangle cut by the vertical scanning lines.
3) *LA:* Always choose the rectangle with larger area during each cut.
4) *SA:* Always choose the rectangle with smaller area during each cut.
5) *AR:* Always choose the rectangle with an aspect ratio closer to one during each cut.

Fig. 8 shows an example of polygon decomposition solutions from these criteria. The numbers of rectangles denote the order

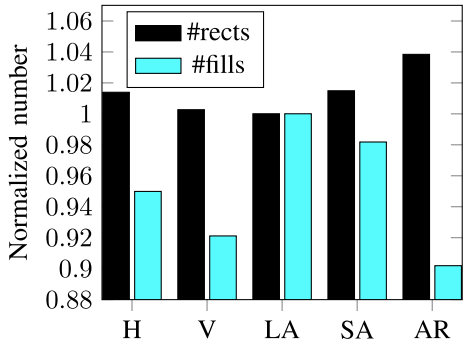

Fig. 9. Comparison in the amounts of decomposed rectangles and generated fills between different selection criteria for odd layers in polygon-to-rectangle conversion. The values are normalized by LA for easier comparison.

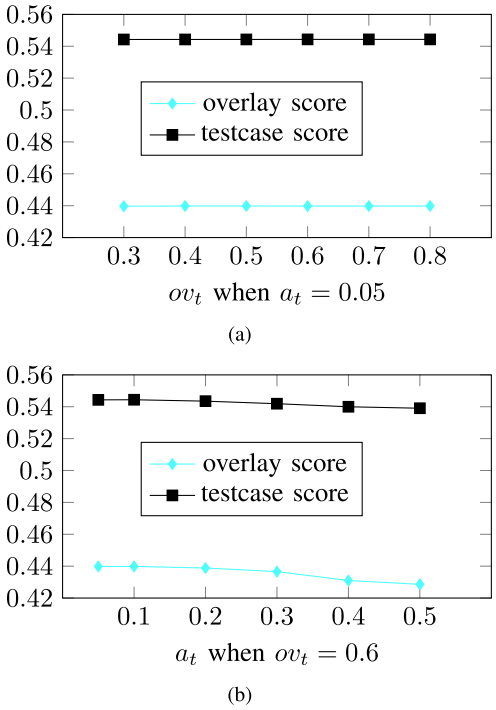

Fig. 10. Trends of overlay and testcase scores with a_t and ov_t. (a) Sweep ov_t when $a_t = 0.05$. (b) Sweep a_t when $\text{ov}_t = 0.6$.

of rectangles generated. The first two criteria *H* and *V* are faster than rest since they only need to keep one scanning line. The area related criteria LA and SA are proposed to generate fewer and larger rectangles. The criterion LA is more straight-forward since larger rectangles are selected greedily. The intuition of SA comes from the fact that if smaller rectangles are selected at the beginning, the rest ones are likely to be large. The aspect ratio criterion AR prefers square rectangles to slim ones so that it is less likely to have DRC rule violation, i.e., minimum width and area rules.

Since the fill generation procedures for odd and even layers are different in Section III-D, we also compare the heuristics separately. In the fill generation of odd layers, we keep collecting largest rectangles until the target density is satisfied. This procedure prefers the total area of largest rectangles is able to meet the target density requirement, while it does not matter for the rest ones to be small. The results for benchmark b

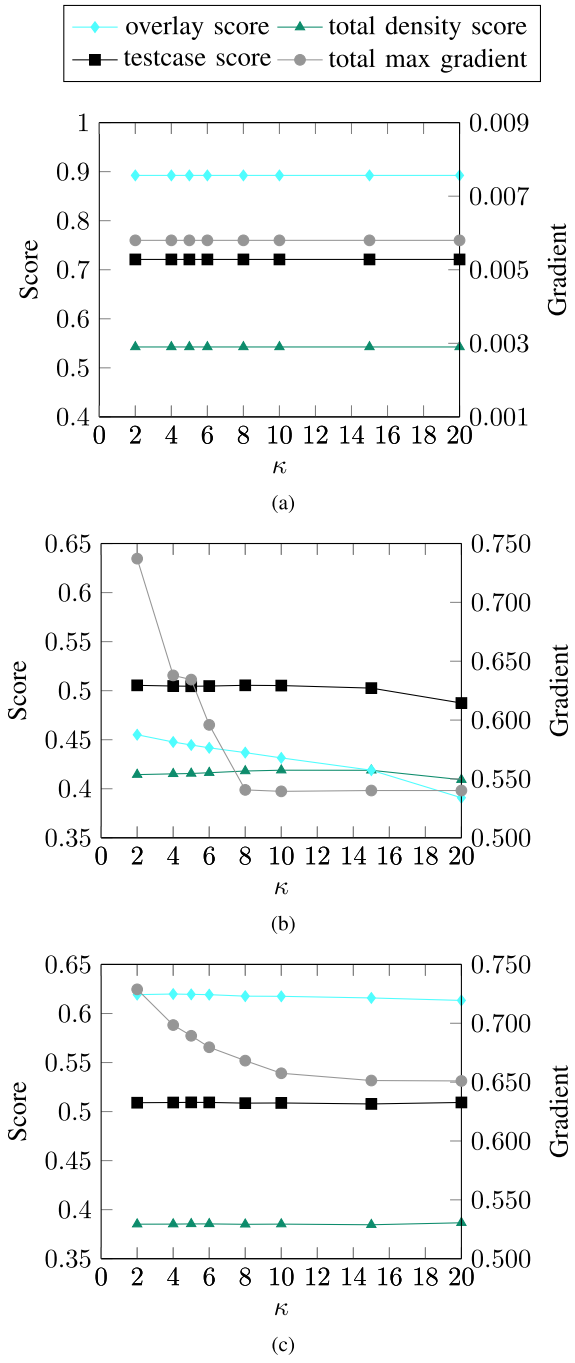

Fig. 11. Tradeoffs of percentage κ between gradient, density, and overlays for benchmark (a) *s*, (b) *b*, and (c) *m*.

with different criteria on odd layers are shown in Fig. 9 where even layers are decomposed with default criterion *H*. The total amount of rectangles generated in polygon decomposition is denoted by "#rects." The total number of fills generated is denoted by "#fills." The values are normalized by the corresponding values of LA for easier comparison. It can be seen that fewer decomposed rectangles do not always lead to fewer fills. Criterion LA generates least decomposed rectangles but most fills in the end. On the other hand, criterion AR generates most decomposed rectangles but least fills which 10% less than that of LA. The performance of other criteria like *H* and *V* lies in the middle.

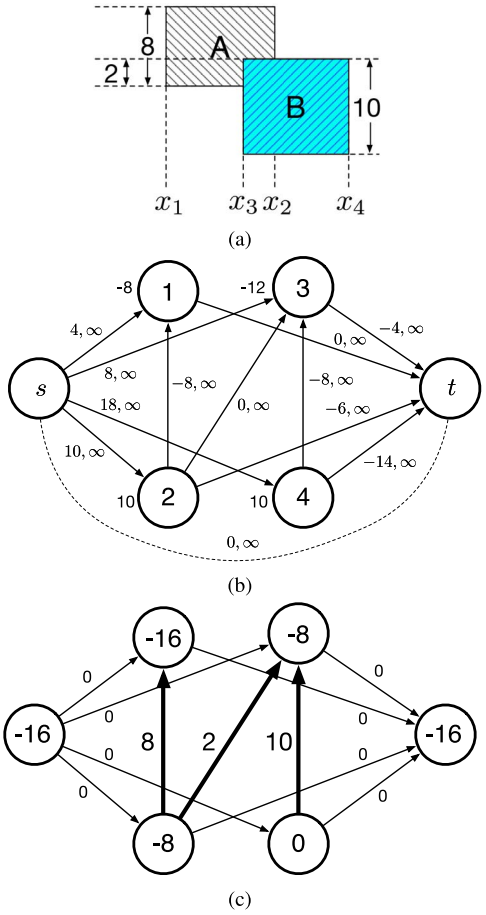

Fig. 12. (a) Example of fill shrinking problem, (b) dual min-cost flow graph, and (c) corresponding solution graph: edges are marked with flow values and nodes are marked with potential values.

The fill generation of even layers is related to overlay cost, which is more difficult to be addressed by simple criteria in polygon decomposition. Therefore, we empirically set that for even layers to default criterion *H* that maintains efficient and stable performance across different benchmarks. But for odd layers we adopt AR criteria for fewer fills in general.

C. Influences of Parameters

In Algorithm 2 of Section III-D, two parameters: 1) a_t and 2) ov_t define the area and overlay threshold to trigger overlay refinement for large candidate fills. We sweep a_t from 0.05 to 0.5 and ov_t from 0.3 to 0.8 to study the influences to overlay and final testcase scores. Eventually, we set $a_t = 0.05$ and $\text{ov}_t = 0.6$ for best testcase scores. Fig. 10 shows two examples of the trending. Fig. 10(a) gives the trend of two scores with the increase of ov_t when a_t is set to 0.05 for benchmark *b*. The effects of ov_t to overlay and testcase scores are very small. The scores do not change much when ov_t increases. Fig. 10(b) gives the trend of two scores with the increase of a_t when ov_t is set to 0.6 for benchmark *b*. The parameter a_t has more impacts on the overlay score, as it drops with the increase of a_t. But the testcase score drops much slower than overlay score, because performing overlay refinement to more fills in general worsen density scores, such as variation.

As mentioned in Section III-C, the percentage κ in the incremental optimization for candidate fill regeneration is very critical to both density gradient and variations. Although larger κ contributes to smaller gradient and better density distribution, it inevitably leads to more overlays and runtime. Fig. 11 plots the trending of total maximum gradient, total density score and testcase quality score with the growth of κ for benchmark *s*, *b*, and *m*. We can see that with the increase of κ from 2 to 10, total maximum gradient drops quickly and then saturates afterwards for benchmark *b* and *m*. The total density score and overlay score grow in an opposite direction in general from 2 to 20. To tradeoff gradient, density and overlay scores, we use $\kappa = 10$ in our implementation.

V. Conclusion

This paper proposes a new methodology for the holistic fill optimization problem in which file size is included to the objective along with other cost functions. Experimental results show the effectiveness of our algorithms in optimizing multiple objectives including overlay, density variation, density gradient, and file size. In the future work, we plan to consider exact locations of signal nets to reduce coupling capacitances rather than only metal density of signal nets from the benchmarks. Future work would also include evaluation on lithography related impacts and methodologies considering lithograph-friendliness during dummy fill insertion.

Appendix
Example of Dual Min-Cost Flow in Section III-E3

Fig. 12(a) gives an example of two candidate fills where *A* is in layer 1 and *B* is in layer 2. Assume in an iteration the relaxed ILP formulation for horizontal direction can be written as follows:

$$\min \quad (x_2 - x_1) \cdot 8 + (x_4 - x_3) \cdot 10 + (x_2 - x_3) \cdot 2 \tag{21a}$$

$$\text{s.t.} \quad x_2 - x_1 \geq 8 \tag{21b}$$

$$x_4 - x_3 \geq 8 \tag{21c}$$

$$x_2 - x_3 \geq 0 \tag{21d}$$

$$0 \leq x_1 \leq 4 \quad 6 \leq x_2 \leq 10, \quad 4 \leq x_3 \leq 8 \quad 14 \leq x_4 \leq 18. \tag{21e}$$

The objective (21a) consists of two terms for density variations and one term for overlay. Equations (21b) to (21c) honor DRC rules such as minimum width rules. Equation (21d) makes sure the overlay computation in the objective is correct. It is also assumed that initially total area of fills is larger than target density, so the absolute operations in the objective can be removed with tighter bound constraints to each variable.

We can construct the dual min-cost flow graph as Fig. 12(b). Variables x_1, x_2, x_3, and x_4 correspond to nodes 1, 2, 3, and 4, respectively. Each node is labeled with node supply which comes from coefficient in the objective (21a). For each differential constraint $x_i - x_j \geq b_{ij}$, an edge from node i to j is inserted with cost of $-b_{ij}$. For each lower bound constraint $x_i \geq l_i$, an edge from node i to t is inserted with cost of $-l_i$. For each upper bound constraint $x_i \leq u_i$, an edge from node s to i is inserted with cost of u_i. The capacity for all edges is infinite. Node s and t corresponds to additional variable y_0 for bound constraints added by (18). They are regarded as virtually connected by an undirected edge with zero cost and infinite capacity. Fig. 12(c) shows the solution graph in which edges are marked with flow values and nodes are marked with node potentials. The final solution for x_i is the difference between the potential of node i and node s/t. So eventually $x_1 = 0, x_2 = 8, x_3 = 8, x_4 = 16$.

Acknowledgment

The authors would like to thank Dr. R. Topaloglu for the evaluation of experimental results and helpful comments.

References

[1] A. B. Kahng and K. Samadi, "CMP fill synthesis: A survey of recent studies," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 1, pp. 3–19, Jan. 2008.

[2] C. Feng, H. Zhou, C. Yan, J. Tao, and X. Zeng, "Efficient approximation algorithms for chemical mechanical polishing dummy fill," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 3, pp. 402–415, Mar. 2011.

[3] A. B. Kahng, G. Robins, A. Singh, and A. Zelikovsky, "New multilevel and hierarchical algorithms for layout density control," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, Hong Kong, 1999, pp. 221–224.

[4] A. B. Kahng, G. Robins, A. Singh, and A. Zelikovsky, "Filling algorithms and analyses for layout density control," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 4, pp. 445–462, Apr. 1999.

[5] R. Tian, M. D. F. Wong, and R. Boone, "Model-based dummy feature placement for oxide chemical-mechanical polishing manufacturability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 7, pp. 902–910, Jul. 2001.

[6] C. Feng, H. Zhou, C. Yan, J. Tao, and X. Zeng, "Provably good and practically efficient algorithms for CMP," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2009, pp. 539–544.

[7] Y. Chen, A. B. Kahng, G. Robins, and A. Zelikovsky, "Monte-Carlo algorithms for layout density control," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, Yokohama, Japan, 2000, pp. 523–528.

[8] Y. Chen, A. B. Kahng, G. Robins, and A. Zelikovsky, "Practical iterated fill synthesis for CMP uniformity," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Los Angeles, CA, USA, 2000, pp. 671–674.

[9] X. Wang, C. C. Chiang, J. Kawa, and Q. Su, "A min-variance iterative method for fast smart dummy feature density assignment in chemical-mechanical polishing," in *Proc. IEEE Int. Symp. Qual. Electron. Design (ISQED)*, San Jose, CA, USA, 2005, pp. 258–263.

[10] C. Liu *et al.*, "An effective chemical mechanical polishing filling approach," in *Proc. IEEE Annu. Symp. VLSI (ISVLSI)*, Montpellier, France, 2015, pp. 44–49.

[11] C. Liu *et al.*, "An effective chemical mechanical polishing fill insertion approach," *ACM Trans. Design Autom. Electron. Syst.*, vol. 21, no. 3, p. 54, 2016.

[12] H.-Y. Chen, S.-J. Chou, and Y.-W. Chang, "Density gradient minimization with coupling-constrained dummy fill for CMP control," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, San Francisco, CA, USA, 2010, pp. 105–111.

[13] P. Wu, H. Zhou, C. Yan, J. Tao, and X. Zeng, "An efficient method for gradient-aware dummy fill synthesis," *Integr. VLSI J.*, vol. 46, no. 3, pp. 301–309, 2013.

[14] Y. Chen, A. B. Kahng, G. Robins, and A. Zelikovsky, "Closing the smoothness and uniformity gap in area fill synthesis," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, Del Mar, CA, USA, 2002, pp. 137–142.

[15] Y. Chen, P. Gupta, and A. B. Kahng, "Performance-impact limited area fill synthesis," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Anaheim, CA, USA, 2003, pp. 22–27.

[16] H. Xiang, L. Deng, R. Puri, K.-Y. Chao, and M. D. F. Wong, "Fast dummy-fill density analysis with coupling constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 4, pp. 633–642, Apr. 2008.
[17] R. O. Topaloglu, "ICCAD-2014 CAD contest in design for manufacturability flow for advanced semiconductor nodes and benchmark suite," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2014, pp. 367–368.
[18] R. B. Ellis, A. B. Kahng, and Y. Zheng, "Compression algorithms for dummy-fill VLSI layout data," in *Proc. SPIE*, vol. 5042. Santa Clara, CA, USA, 2003, pp. 233–245.
[19] Y. Chen, P. Gupta, and A. B. Kahng, "Performance-impact limited-area fill synthesis," in *Proc. SPIE*, vol. 5042. Santa Clara, CA, USA, 2003, pp. 75–86.
[20] R. O. Topaloglu, "Energy-minimization model for fill synthesis," in *Proc. IEEE Int. Symp. Qual. Electron. Design (ISQED)*, San Jose, CA, USA, 2007, pp. 444–451.
[21] A. B. Kahng and R. O. Topaloglu, "A DOE set for normalization-based extraction of fill impact on capacitances," in *Proc. IEEE Int. Symp. Qual. Electron. Design (ISQED)*, San Jose, CA, USA, 2007, pp. 467–474.
[22] K. D. Gourley and D. M. Green, "Polygon-to-rectangle conversion algorithm," *IEEE Comput. Graph. Appl.*, vol. 3, no. 1, pp. 31–36, Jan. 1983.
[23] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, 2005.
[24] J. Vygen, "Algorithms for detailed placement of standard cells," in *Proc. IEEE/ACM Design Autom. Test Eurpoe (DATE)*, Paris, France, 1998, pp. 321–324.
[25] X. Tang, R. Tian, and M. D. F. Wong, "Optimal redistribution of white space for wire length minimization," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, Shanghai, China, 2005, pp. 412–417.
[26] S. Ghiasi, E. Bozorgzadeh, P.-K. Huang, R. Jafari, and M. Sarrafzadeh, "A unified theory of timing budget management," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 11, pp. 2364–2375, Nov. 2006.
[27] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[28] *LEMON version 1.3.1*. Accessed on Nov. 1, 2016. [Online]. Available: http://lemon.cs.elte.hu/trac/lemon
[29] *Gurobi Optimizer Reference Manual*, Gurobi Optim. Inc., Houston, TX, USA, 2014. [Online]. Available: http://www.gurobi.com

Yibo Lin received the B.S. degree in microelectronics from Shanghai Jiaotong University, Shanghai, China, in 2013. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA.

His current research interests include physical design and design for manufacturability.

Mr. Lin was a recipient of Franco Cerrina Memorial Best Student Paper Award at SPIE Advanced Lithography Conference 2016, and National Scholarship at Shanghai Jiaotong University in 2012. He has interned at IMEC, Cadence, and Oracle.

Bei Yu (S'11–M'14) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Assistant Professor with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong.

Dr. Yu was a recipient of the three Best Paper Awards at SPIE Advanced Lithography Conference 2016, International Conference on Computer Aided Design (ICCAD) 2013, and Asia and South Pacific Design Automation Conference (ASPDAC) 2012, three other Best Paper Award Nominations at Design Automation Conference 2014, ASPDAC 2013, ICCAD 2011, and three ICCAD Contest Awards in 2015, 2013, and 2012. He has served in the editorial boards of *Integration, the VLSI Journal* and *IET Cyber-Physical Systems: Theory & Applications*. He has also received European Design and Automation Association (EDAA) Outstanding Dissertation Award in 2014, Chinese Government Award for Outstanding Students Abroad in 2014, SPIE Scholarship in 2013, and IBM PhD Scholarship in 2012.

David Z. Pan (S'97–M'00–SM'06–F'14) received the B.S. degree from Peking University, Beijing, China, and the M.S. and Ph.D. degrees from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA.

From 2000 to 2003, he was a Research Staff Member with the IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He is currently the Engineering Foundation Endowed Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA. He has published over 250 papers in refereed journals and conferences, and is the holder of eight U.S. patents. His current research interests include cross-layer nanometer IC design for manufacturability, reliability, security, new frontiers of physical design, and CAD for emerging technologies.

Dr. Pan was a recipient of a number of awards, including the SRC 2013 Technical Excellence Award, the ACM/IEEE Design Automation Conference (DAC) Top 10 Author in Fifth Decade, the DAC Prolific Author Award, the ASPDAC Frequently Cited Author Award, 13 Best Paper Awards and several international CAD contest awards, the Communications of the ACM Research Highlights in 2014, the ACM/SIGDA Outstanding New Faculty Award in 2005, the NSF CAREER Award in 2007, the SRC Inventor Recognition Award three times, the IBM Faculty Award four times, the UCLA Engineering Distinguished Young Alumnus Award in 2009, and the UT Austin RAISE Faculty Excellence Award in 2014. He has served as a Senior Associate Editor for *ACM Transactions on Design Automation of Electronic Systems*, an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS, *Science China Information Sciences*, the *Journal of Computer Science and Technology*. He has served as the Program/General Chair of ACM International Symposium on Physical Design 2007/2008, TPC Chair for ASPDAC 2017, Vice Program Chair for ICCAD 2017, Tutorial Chair for DAC 2014, among others.