High Performance Dummy Fill Insertion with Coupling and Uniformity Constraints

Yibo Lin , Bei Yu , and David Z. Pan

ECE Department, University of Texas at Austin, Austin, TX, USA
{yibolin, bei}@cerc.utexas.edu, dpan@ece.utexas.edu

ABSTRACT

In deep-submicron VLSI 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 and sizing framework, where the coupling capacitance issues and density variations are considered simultaneously. The experimental results for ICCAD 2014 contest benchmarks demonstrate the effectiveness of our methods.

1. INTRODUCTIONS

In current VLSI 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 multi-objective fill insertion are still in great demand.

The flow for layout density optimization can be generalized as two phases: density analysis and 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 multi-window and multi-layer 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, 6]. 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 larger and larger that LP-based method reaches their limitation due to problem sizes. In both [1] and [7], analysis of an example layout with $200\mu m \times 200\mu m$ 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 [8, 9, 10].

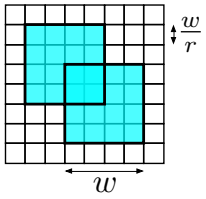

Figure 1: An example of $w \times w$ windows with r^2 tiles.

Furthermore, the introduction of dummy fills will incur additional coupling capacitance, causing performance degradation. The first integer linear programming (ILP) based approach considering coupling capacitance is proposed by Chen et al. [11]. Xiang et al. [12] 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 GDSII and 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 [13].

To motivate the development of more effective dummy fill algorithms, ICCAD 2014 held a dummy fill contest [13] 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 2). This is a brand new challenge since file size was not taken into consideration before

DAC'15, June 07–11 2015, San Francisco, USA.

http://dx.doi.org/10.1145/2744769.2744850

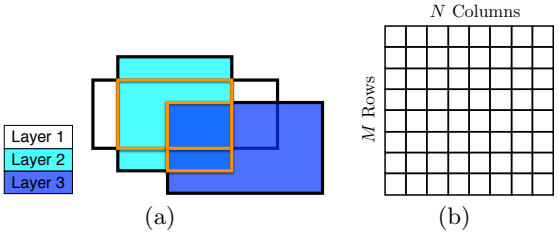

Figure 2: Example of (a) overlay between layers (b) square windows for density analysis.

and previous tile-based method is not suitable.

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. We propose an ILP formulation and then its dual min-cost flow formulation to optimize overlay and layout density efficiently.

3. Experimental results show that our algorithms outperform the top teams by over 10% on ICCAD 2014 DFM contest benchmarks [13].

The rest part of this paper is organized as follows: Section 2 shows the definitions of related concepts and problem formulation. Section 3 explains the optimization algorithms in detail. Section 4 presents the experimental results in different scores. In the end, we conclude our work in Section 5.

2. PRELIMINARIES

2.1 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 work (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) [14, 15]. As shown in Fig. 2(a), the enclosure regions of orange lines are counted to overlay area.

2.2 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 [7]. In other words, density variation along windows is very critical, which is also the purpose of introducing dummy fills.

The distribution of window densities in this work is evaluated with 3 scores: variation, line hotspots, and outlier hotspots [14, 15]. 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,

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

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,

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

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

All the 3 scores above evaluate different perspectives of the density distribution, which is 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.

2.3 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 work, the optimization will be focused on a combined objective of layout overlay and density variation.

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

$$\text{score} = \alpha_{ov}s_{ov} + \alpha_{\sigma}s_{\sigma} + \alpha_{lh}s_{lh} + \alpha_{oh}s_{oh} + \alpha_{fs}s_{fs}, \tag{3}$$

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

$$f(x) = \max(0, 1 - \frac{x}{\beta}), \tag{4}$$

where α and β are benchmark-related parameters in the contest.

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 IO format.

3. DUMMY FILL INSERTION ALGORITHMS

The flow of our algorithm is summarized in Fig. 3. After reading the input fill regions, we need to convert polygons to rectangles[16] 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.

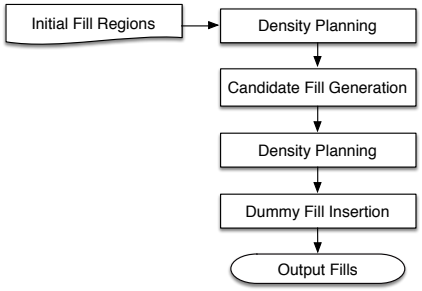

Figure 3: Our dummy fill insertion flow.

3.1 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 3.2) and final fill insertion (Section 3.3) 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 density score, the summation of variation score, line hotspot score and 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 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 t_d. Its relation with $d(i,j)$ can be shown as follows,*

$$d(i,j) = \begin{cases} l(i,j), & \text{if } t_d < l(i,j) \\ u(i,j), & \text{if } t_d > u(i,j) \\ t_d, & \text{other} \end{cases} \quad (5)$$

The density planning problem is now transformed to find the best t_d for maximum density score. To find the best t_d, 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

$$t_d = \max\left(l(k,n)\right), \forall k \in 1,2,...,N, n \in 1,2,...,M, \quad (6)$$

It means the target density t_d 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

$$u(i,j) < \max\left(l(k,n)\right), \quad (7)$$

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 t_d 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.

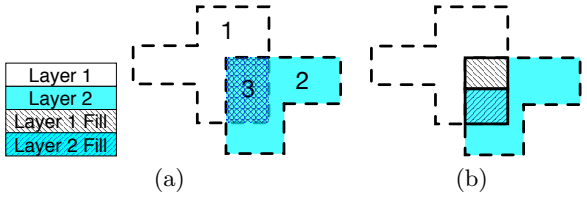

Figure 4: Case I Zero Overlay: Example of (a) Fill Regions in Neighboring Layers in a Window (b) Fill Solution without Overlay.

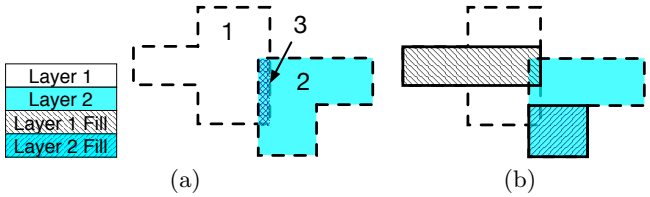

Figure 5: Case II Non-zero Overlay: Example of (a) Fill Regions in Neighboring Layers in a Window (b) Fill Solution with Overlay.

3.2 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 2 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 2 neighboring layers. We can divide these fill regions into 3 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).

Case II Non-zero 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 Region 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 Eqn. (8),

$$q = -\frac{fill\ overlay}{fill\ area} + \gamma \cdot \frac{fill\ area}{window\ area}, \quad (8)$$

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.

Algorithm 1 Candidate Fill Region Generation

Input: Feasible fill regions of all layers in a window.
Output: 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

Since the number of layers is usually larger than 2, the actual implementation combines the previous ideas, which is summarized in Alg. 1. It is impossible to calculate the overlay between fills before any fill is inserted to the 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 Eqn. (8). 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.

3.3 Dummy Fill Insertion

In this section, we will determine the sizes of fills to further reduce density variation and overlay in an efficient way. Table 1 gives the definitions of symbols used in the following explanation.

3.3.1 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 mathematical Eqn. (9),

$$\min_{\substack{x_i^l, x_i^h \\ y_i^l, y_i^h}} \sum_{l \in L} d_g(l) + \eta \cdot \sum_{l \in L} ov(l, l+1) \tag{9a}$$

$$\text{s.t. } d_g(l) = |\sum_{i \in F(l)} w_i \cdot h_i - d_t(l) \cdot a_w|, \ l \in L \tag{9b}$$

$$ov(l, l+1) = \sum_{i \in O(l)} w_i \cdot h_i, \ l \in L \tag{9c}$$

$$w_i = x_i^h - x_i^l, h_i = y_i^h - y_i^l, \tag{9d}$$

$$w_i \geq w_m, h_i \geq w_m, \tag{9e}$$

$$w_i \cdot h_i \geq a_m, \tag{9f}$$

$$e(i,j) \geq s_m, \tag{9g}$$

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

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 (9b), 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 (9c) defines the overlay area ov. Constraints (9e) to (9g) state required DRC rules, such as minimum width, minimum area and minimum spacing. Eqn. (9) defines a non-convex problem, so it is very expensive to solve it.

3.3.2 Problem Relaxation

Previous formulation contains multiplication operations between variables in two directions, which results in non-convex 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 Constraint (9b) and (9c) can be relaxed to

$$d_g(l) = |\sum_{i \in F(l)} w_i \cdot h_{i0} - d_t(l) \cdot a_w|, \ l \in L, \tag{10}$$

$$ov(l, l+1) = \sum_{i \in O(l)} w_i \cdot h_{i0}, \ l \in L, \tag{11}$$

where h_{i0} is the initial height of candidate fills from Section 3.2.

Constraint (9e) to (9f) can be merged into one equation,

$$w_i \geq \max(w_m, \frac{a_m}{h_{i0}}). \tag{12}$$

Constraint (9g) 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,

$$e^h(i,j) \geq s_m. \tag{13}$$

With Eqn. (10), (11), (12) and (13), a relaxed problem solvable with ILP is formed.

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.

3.3.3 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

Table 1: 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	Target density
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 fill 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

problem is generally NP-hard to solve. Here we show that the formulation is able to achieve further speedup with dual min-cost flow.

Eqn. (10) 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 3.3.2 can be written in a generalized manner without any absolute operation,

$$\min_{x_i} \sum_{i=1}^{N} c_i x_i \tag{14a}$$

$$\text{s.t. } x_i - x_j \geq b_{ij},\ (i,j) \in E, \tag{14b}$$

$$l_i \leq x_i \leq u_i,\ i = 1, 2, ..., N, \tag{14c}$$

$$x_i \in Z,$$

Eqn. (14) is a linear program with only differential constraints and bounded variables, which can be transformed to a dual min-cost flow problem [17]. Min-cost flow problem is a relative mature field with very fast algorithms, which is often adopted in the physical design flow [18, 19, 20].

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

$$\min_{y_i} \sum_{i=0}^{N} c_i' y_i, \tag{15a}$$

$$\text{s.t. } y_i - y_j \geq b_{ij}',\ (i,j) \in E', \tag{15b}$$

$$y_i \in Z,$$

where

$$x_i = y_i - y_0, \quad i = 1, 2, ..., N \tag{16a}$$

$$c_i' = \begin{cases} c_i & i = 1, 2, ..., N \\ -\sum_{i=1}^{N} c_i & i = 0 \end{cases} \tag{16b}$$

$$b_{ij}' = \begin{cases} b_{ij} & (i,j) \in E \\ l_i & i = 1, 2, ..., N, j = 0 \\ -u_i & i = 0, j = 1, 2, ..., N \end{cases} \tag{16c}$$

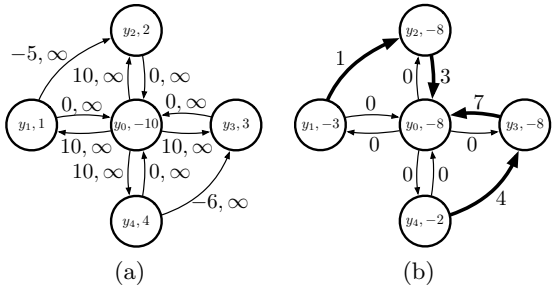

Figure 6: (a) Example of min-cost flow graph: edges are marked with cost/capacity pairs and node is marked with name/supply pairs (b) Corresponding solution graph: edges are marked with flow values and nodes are marked with name/potential pairs.

Lemma 1. *Eqn.* (14) *and Eqn.* (15) *are equivalent.*

The proof is omitted due to space limitations.

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 Eqn. (16a).

For example, we want to minimize $x_1 + 2x_2 + 3x_3 + 4x_4$ with constraints $x_1 - x_2 \geq 5$ and $x_4 - x_3 \geq 6$ and all variables are integers bounded with $[0, 10]$. The min-cost flow graph can be constructed as Fig. 6(a). The nodes in the graph are directly named with y_0, y_1, y_2, y_3 and y_4. The corresponding solution graph is given by Fig. 6(b). We are able to calculate x with Eqn. (16a): $5, 0, 0, 6$.

4. EXPERIMENTAL RESULTS

Our algorithms were implemented in C++ and tested on an 8-Core 3.40 GHz Linux server with 32 GB RAM. The results of ICCAD 2014 contest top three teams are tested on a 2.6 GHz machine with 64 GB RAM. LEMON [21] is used as the min-cost flow solver. Our solutions have been verified by the contest organizer [13]. Statistics about benchmarks are listed in Table 2. It contains coefficients to calculate following scores. **Overlay*** denotes the overlay score stated in Section 2; **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 Eqn. (4). According to ICCAD 2014 contest [13], **Testcase Score** is the weighted summation of all the scores above. **Testcase 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 Eqn. (3) and (4). Run-time score and memory score are also calculated with Eqn. (4). All the scores are obtained from the contest organizer [13] except run-time and memory usage. To evaluate the effectiveness of our algorithm, we compare our results with top three teams in the contest.

Table 3 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, our quality score is 13% better and the overall score is about 10% higher than the

Table 2: 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

Table 3: Experimental Results on ICCAD 2014 Benchmark

Design	Team	Overlay*	Variation*	Line*	Outlier*	Size*	Run-time*	Memory*	Testcase Quality	Testcase 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
	ours	0.723	0.948	0.979	0.994	0.887	0.872	0.818	**0.724**	**0.895**
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
	ours	0.685	0.499	0.470	0.953	0.765	0.351	0.852	**0.512**	**0.607**
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
	ours	0.493	0.643	0.766	0.088	0.905	0.750	0.786	**0.439**	**0.591**

top team in the contest.

According to score calculation, density related scores (variation, line hotspots and outlier hotspots) take 45%, and overlay score takes 20%. From Table 3 we can see that our overall density scores are among the highest. e.g. 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. 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. This impact on runtime becomes very serious especially in benchmark b where more than 40% of the total runtime is spent on file IO. In summary, the results demonstrate that our algorithms produce more balanced solutions which can handle overlay and density requirements simultaneously.

5. CONCLUSION

This work 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 and file size. Future work would include evaluation on lithography related impacts and methodologies considering lithograph-friendliness during dummy fill insertion.

Acknowledgment

Thanks to Dr. Rasit Topaloglu for the evaluation of experimental results and helpful comments.

6. REFERENCES

[1] A. B. Kahng and K. Samadi, "CMP fill synthesis: A survey of recent studies," *IEEE Transactions on CAD*, vol. 27, no. 1, pp. 3–19, 2008.

[2] C. Feng, H. Zhou, C. Yan, J. Tao, and X. Zeng, "Efficient approximation algorithms for chemical mechanical polishing dummy fill," *IEEE Transactions on CAD*, vol. 30, no. 3, pp. 402–415, 2011.

[3] A. B. Kahng, G. Robins, A. Singh, and A. Zelikovsky, "New multilevel and hierarchical algorithms for layout density control," in *ASPDAC*, 1999, pp. 221–224.

[4] A. B. Kahng, G. Robins, A. Singh, and A. Zelikovsky, "Filling algorithms and analyses for layout density control," *IEEE Transactions on CAD*, vol. 18, no. 4, pp. 445–462, 1999.

[5] R. Tian, D. Wong, and R. Boone, "Model-based dummy feature placement for oxide chemical-mechanical polishing manufacturability," *IEEE Transactions on CAD*, vol. 20, no. 7, pp. 902–910, 2001.

[6] H. Xiang, L. Deng, R. Puri, K.-Y. Chao, and M. D. Wong, "Fast dummy-fill density analysis with coupling constraints," *IEEE Transactions on CAD*, vol. 27, no. 4, pp. 633–642, 2008.

[7] C. Feng, H. Zhou, C. Yan, J. Tao, and X. Zeng, "Provably good and practically efficient algorithms for CMP," in *DAC*, 2009, pp. 539–544.

[8] Y. Chen, A. B. Kahng, G. Robins, and A. Zelikovsky, "Monte-Carlo algorithms for layout density control," in *ASPDAC*, 2000, pp. 523–528.

[9] Y. Chen, A. B. Kahng, G. Robins, and A. Zelikovsky, "Practical iterated fill synthesis for CMP uniformity," in *DAC*, 2000, pp. 671–674.

[10] 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 *ISQED*, 2005, pp. 258–263.

[11] Y. Chen, P. Gupta, and A. B. Kahng, "Performance-impact limited area fill synthesis," in *DAC*, 2003, pp. 22–27.

[12] H. Xiang, L. Deng, R. Puri, K.-Y. Chao, and M. D. Wong, "Dummy fill density analysis with coupling constraints," in *ISPD*, 2007, pp. 3–10.

[13] R. O. Topaloglu, "ICCAD-2014 CAD contest in design for manufacturability flow for advanced semiconductor nodes and benchmark suite," in *ICCAD*, 2014, pp. 367–368.

[14] R. O. Topaloglu, "Energy-minimization model for fill synthesis," in *ISQED*, 2007, pp. 444–451.

[15] A. B. Kahng and R. O. Topaloglu, "A DOE set for normalization-based extraction of fill impact on capacitances," in *ISQED*, 2007, pp. 467–474.

[16] K. D. Gourley and D. M. Green, "Polygon-to-rectangle conversion algorithm." *IEEE COMP. GRAPHICS & APPLIC.*, vol. 3, no. 1, pp. 31–32, 1983.

[17] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall/Pearson, 2005.

[18] J. Vygen, "Algorithms for detailed placement of standard cells," in *DATE*, 1998, pp. 321–324.

[19] X. Tang, R. Tian, and M. D. Wong, "Optimal redistribution of white space for wire length minimization," in *ASPDAC*, 2005, pp. 412–417.

[20] S. Ghiasi, E. Bozorgzadeh, P.-K. Huang, R. Jafari, and M. Sarrafzadeh, "A unified theory of timing budget management," *IEEE Transactions on CAD*, vol. 25, no. 11, pp. 2364–2375, 2006.

[21] "LEMON," http://lemon.cs.elte.hu/trac/lemon.