Area Fill Synthesis for Uniform Layout Density

Yu Chen, Andrew B. Kahng, Gabriel Robins, and Alexander Zelikovsky

***Abstract*—*Chemical–mechanical polishing (CMP)* and other manufacturing steps in very deep submicron very large scale integration have varying effects on device and interconnect features, depending on local characteristics of the layout. To improve manufacturability and performance predictability, the authors seek to make a layout uniform with respect to prescribed density criteria, by inserting "area fill" geometries into the layout. In this paper, they make the following contributions. First, the authors define the flat, hierarchical, and multiple-layer filling problems, along with a unified density model description. Secondly, for the flat filling problem, they summarize current linear programming approaches with two different objectives, i.e., the Min–Var and Min–Fill objectives. They then propose several new Monte Carlo-based filling methods with fast dynamic data structures. Thirdly, they give practical *iterated methods* for layout density control for CMP uniformity based on linear programming, Monte Carlo, and greedy algorithms. Fourthly, to address the large data volume and inherent lack of scalability of flat layout density control, the authors propose practical methods for hierarchical layout density control. These methods smoothly trade off runtime, solution quality, and output data volume. Finally, they extend the linear programming approaches and present new Monte Carlo-based methods for the multiple-layer filling problem. Comparisons with previous filling methods show the advantages of the new iterated Monte Carlo and iterated greedy methods for both flat and hierarchical layouts and for both density models (*spatial* density and *effective* density). The authors achieve near-optimal filling for flat layouts with respect to each of these objectives. Their experiments indicate that the hybrid hierarchical filling approach is efficient, scalable, accurate, and highly competitive with existing methods (e.g., linear programming-based techniques) for hierarchical layouts.**

***Index Terms*—Area fill, chemical–mechanical polishing, hierarchical layout, layout density control, layout planarization, Monte Carlo methods.**

I. Introduction

As PREDICTED by the Semiconductor Industry Association's Technology Roadmap [21], very large scale integration (VLSI) technology has entered deep submicron regimes, where the manufacturing process increasingly constrains physical layout design and verification [16]. Many process layers, including diffusion and thin-ox, have associated density rules that are satisfied by postprocessing steps which add *area fill* geometries to the layout. Historically, only foundries or specialized technology computer-aided design (TCAD) tool companies performed the layout postprocessing necessary to achieve layout uniformity. Today, however, electronic computer-aided design (ECAD) tools for physical design and verification cannot remain oblivious to such postprocessing phases.

Literature on area fill has focused on chemical–mechanical polishing (CMP) of spin-on glass (SOG) interlayer dielectrics (ILD) [14], [18], [27]. Postpolish ILD thickness variation is kept within acceptable limits by controlling local feature density, relative to a process-specific "window size" (on the order of 1–3 mm), that depends on CMP pad material, slurry composition, and other factors [7].[1]

Application of area fill to device layers (diffusion, poly, thin-ox) is equally (or even more) critical. Isolated transistors are susceptible to contact overetch in reactive ion etch (RIE) process steps, which results in leakage. Chemical vapor deposition (CVD) steps are also subject to iso-dense variations. CVD and etch process variation are particularly troublesome with respect to today's lightly doped drain (LDD) device properties. The bottom-line performance effects of these process variations are well known, e.g., Garofalo *et al.* [8] document 10% variation in interline capacitance resulting from 5% variation in linewidth and 12% error in ring oscillator frequency solely from proximity effects. At the same time, it is also well known that the uniformity of feature density obtained via area fill can mitigate macroscopic process proximity effects such as contact etch variation in reactive ion etch, and nonuniformity of chemical vapor deposition.

With respect to the potential negative effects of area fill insertion, certainly the fill geometries can affect interconnect capacitance, signal delay, and crosstalk. The exact change in interconnect capacitance depends mainly on the size of the fill geometries and proximity to interconnect lines. However, Grobman *et al.* have recently given detailed experimental data [10] pointing out that capacitance of dense lines is not significantly affected by floating fill geometries on neighboring layers, since this capacitance is mostly dominated by same-layer neighbor coupling. Furthermore, smaller fill geometries reduce crosstalk to distant neighbors and lead to a smaller increase in total capacitance. The conclusion from such analyses is that the first-order performance concern remains to

Manuscript received October 8, 2000; revised December 24, 2001. This work was supported in part by a Packard Foundation Fellowship, by the MARCO Gigascale Silicon Research Center, by the National Science Foundation under Young Investigator Award MIP-9457412, by the National Science Foundation under Grant CCR-9988331, Award MM@—3018 of MRDA & CRDF, by the State of Georgia's Yamacraw intitiative, and by a grant from Cadence Design Systems, Inc. This paper was recommended by Associate Editor M. Pedram.

Y. Chen is with the Department of Computer Science, University of California at Los Angeles, Los Angeles, CA 90095-1596 USA (e-mail: yuchen@cs.ucla.edu).

A. B. Kahng is with the Departments of Computer Science and Engineering and Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: abk@cs.ucsd.edu).

G. Robins is with the Department of Computer Science, University of Virginia, Charlottesville, VA 22903-2442 USA (e-mail: robins@cs.virginia.edu).

A. Zelikovsky is with the Department of Computer Science, Georgia State University, Atlanta, GA 30303 USA (e-mail: alexz@cs.gsu.edu).

Digital Object Identifier 10.1109/TCAD.2002.802278.

[1]We observe that the 1999 International Technology Roadmap for Semiconductors [22] added copper interconnect dishing to the fundamental roadmap parameters for Interconnect. (The 2000 ITRS also added copper interconnect thinning in CMP to the fundamental parameters.) Density-mediated process variation has therefore become a first-order concern for interconnects.

0278-0070/02$17.00 © 2002 IEEE

improve planarization and uniformity of geometry via area fill insertion.

Finally, we note that current industry tools appear to be slow in handling detailed physical models of CMP, such as those addressed by our work. To the best of our knowledge, most current industrial tools such as Cadence Assura 2.0 perform fill insertion as part of physical verification, using rule-based methods. The underlying geometry engines are tuned to Boolean operations on layout layers and to local (e.g., width/spacing rule) checks. A typical use of such infrastructure is to simply insert area fill geometries to increase local density wherever there exist large enough slack areas. This is usually done with Boolean operations to find the slack areas and fill them with geometries of a prescribed density. The main problem with this method is that the spread between minimum and maximum densities is usually fairly large, and it is unclear how the fill insertion approach is related to known analytical models for the relationship between local density and ILD thickness. Comparisons with industry tools have not been possible, as no commercial tools of which we are aware offer hierarchical fill insertion capability, and the related methods of [25] are Motorola-internal and not publicly available [9].

A. Organization of the Paper

The remainder of our paper reviews the range of local density models and density control objectives, then proposes several new approaches to a flat and hierarchical density control for CMP. Section II defines the single-layer filling problem for both flat and hierarchical layouts. Both formulations are based on the practical industry standard *fixed dissection* density analysis regime [12]. Relevant objectives include the Min–Var and Min–Fill objectives. Though hierarchical filling can speed up verification of filled layout and decrease data volume, there is an obvious conflict between honoring the layout hierarchy and achieving high-quality filling results. The filling problem for multiple-layer layouts is then discussed, where the cumulative density effect is considered. Section III gives a unified description of existing models for density calculation for CMP. We review a standard model for oxide planarization via CMP and describe *spatial local density* and *effective local density* models. Section IV first reviews several linear programming (LP)-based approaches that determine the optimal fill amounts to be inserted into the layout, with respect to the Min–Var and Min–Fill objectives. Then, because LP approaches tend to require too much memory in practice, we propose new Monte Carlo-based approaches for flat filling, which are as accurate and yet faster than LP approaches. Section V analyzes the difficulties inherent in hierarchical filling, as well as the reasons why the LP approach is sometimes inapplicable. We then propose a new Monte Carlo filling approach and a hybrid hierarchical/flat filling approach which is scalable, efficient, and highly competitive with flat filling. Section VI discusses the extensions of LP and Monte Carlo approaches to a multiple layer model. Sections VII and VIII describe our implementation testbed and computational experience, and Section IX concludes with directions for future research.

II. The Filling Problem

Layout density control consists of two phases: *density analysis* and *fill synthesis*. Density analysis determines the area available for filling. Fill synthesis then computes the amount of fill feature area which should be added into each part of the layout in order to achieve uniformity, then generates the required fill geometries. In this paper, we address the main problem of the area fill synthesis phase.

A. Flat Filling

Given a design rule-correct layout in an $n \times n$ *layout region*, along with a *window size* $w < n$, and upper (U) and lower (L) bounds on the feature density in any window, add *area fill* geometries to create a *filled layout* such that either:

- *(Min–Var Objective)* the *variation* in window density (i.e., maximum window density minus minimum window density) is minimized while the window density does not exceed the given upper bound U; or
- *(Min–Fill Objective)* the number of inserted fill geometries is minimized while the density of any window remains in the given range (L, U).

The Min–Var objective, introduced in [12], captures the "manufacturing side" of fill synthesis, which seeks the most uniform density distribution possible. The Min–Fill objective, recently proposed in [25], models the "design side" in that it seeks to minimize the coupling capacitance and the uncertainty caused by filling. Algorithms for filling flat designs can be classified into two categories: linear-programming (LP)-based approaches [12], [25], and Monte Carlo-based methods [4], [5].

B. Hierarchical Filling

Hierarchy arises in both custom and semicustom design flows. In custom design, hierarchy is used mostly for streamlining the management and the decomposition of the design problem. In semicustom design, hierarchy is associated more with reuse of standard cells, whose layouts include device layers and local interconnect, or IP blocks. The key observation is that hierarchical designs become difficult to verify when flattened. Hence, hierarchical filling can enable simpler and faster verification of the filled layout, since verification can still follow the structure of the original hierarchy. Hierarchical filling can also decrease data volume for standard-cell designs. (In general, data volume is a big issue for area fill since a filling solution can consist of many millions of tiny geometries.) Thus, hierarchical fill generation is an emerging requirement for future commercial EDA tools [20].

Our present work investigates approaches and tradeoffs inherent in filling master cells rather than just individual instances. We consider hierarchical filling as a postprocessing step performed (on device layers) after placement. When router access to local interconnect (salicide) and M1 layer is strongly restricted,[2] then hierarchical filling may be performed after routing as well. Hierarchical filling entails obvious complex constraints.

[2]For example, Cadence and Avant! gridded routers are often restricted to well-defined pin availabilities at points of the routing grid.

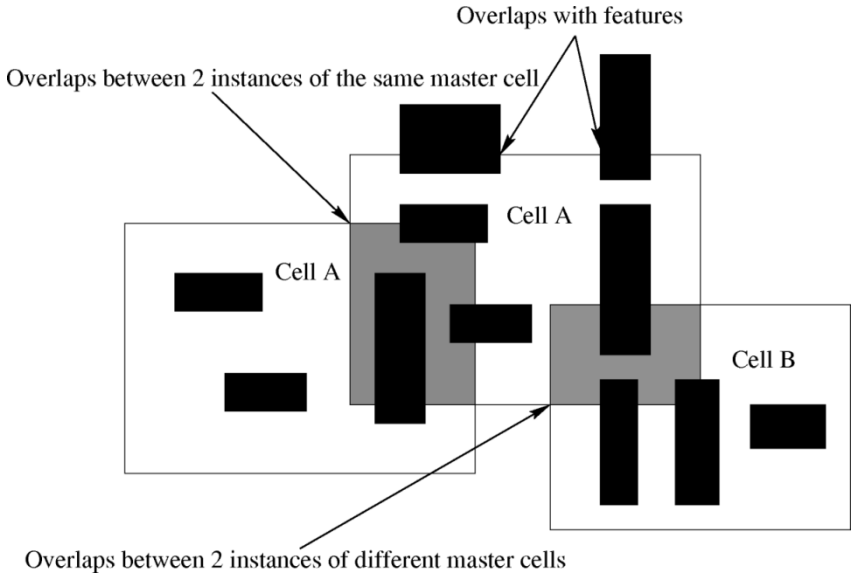

Fig. 1. The types of interactions or interferences with master cells.

- When area fill is inserted into a master cell, it must satisfy density constraints in all contexts for instantiations of the master.
- There are many interactions or interferences at master cell boundaries and at distinct levels of the hierarchy (see Fig. 1).
- Solution quality in terms of either the Min–Var or Min–Fill objective will be worse for hierarchical solutions than flat solutions, because the former are more constrained.
- The number of constraints for LP-based hierarchical filling explodes combinatorially for the known LP-formulations, rendering unusable the linear programming techniques which have been successful for flat filling [12], [25].

The filling problem for hierarchical (standard-cell) layouts is similar to its counterpart for flat layouts, except that the hierarchical structure of master cells must be preserved, i.e., the same filling geometry is simultaneously added to *all* instances of the same master cell. Here, we assume that we can fill the slack (i.e., free) area of each master cell independently and uniformly.

The Hierarchical Filling Problem: Solve the Filling Problem for a given standard-cell layout so that:

- filling geometries are added only to master cells;
- each cell of the filled layout is a filled version of the corresponding original master cell;
- the increase in (hierarchical) layout data volume does not exceed a given threshold.

C. Multiple-Layer Filling

In the layout with multiple layers, each layer except the bottom one cannot assume a perfectly flat starting surface. Thus, independently filling each layer optimally may not achieve an acceptable planarization for the top layers as layers are stacked during the manufacturing process.

The Multiple-Layer Filling Problem: Solve the Filling Problem for a given multiple-layer layout so that either:

- (Min–Var Objective) the sum of variations in window density on each layer is minimized, or the variance of variations in window density on each layer is minimized; or
- (Min–Fill Objective) the number of inserted fill geometries is minimized while the density of any window remains in the given range (L_k, U_k) for each layer k.

III. A Unified Description of Layout Density Models for CMP

Several models for oxide planarization via CMP are reviewed in [18]. In particular, the accurate and well-accepted model of [23] is neither computationally expensive nor difficult to calibrate. In this model, the interlevel dielectric thickness z at location (x, y) is calculated as

$$z = \begin{cases} z_0 - \left(\dfrac{K_i t}{\rho(x,y)}\right), & t < (\rho_0 z_1)/K_i \\ z_0 - z_1 - K_i t + \rho_0(x,y) z_1, & t > (\rho_0 z_1)/K_i \end{cases} \quad (1)$$

where K_i is the blanket polish rate, z_0 is the height of oxide deposition, z_1 is the height of existing features, t is the polish time, and ρ_0 is the initial pattern density. The crucial element of the model is the determination of the effective initial pattern density $\rho(x, y)$. In this section, we give a unified approach to two different definitions of pattern density studied in [12] and [25], respectively. This unification will allow us to exploit the same methods for layout density control for both pattern density definitions.

The pattern density in (x, y) is a local property and therefore depends on spatial pattern density within some close range of the point (x, y). This local property may be captured by introducing (for a certain w) a $w \times w$-window W centered at (x, y), and assuming that $\rho(x, y)$ depends only on the pattern density distribution in W.

To make the filling problem more tractable, a standard industry practice is to consider only a finite set of layout windows. Bounding the effective density in a fixed set of $w \times w$ windows

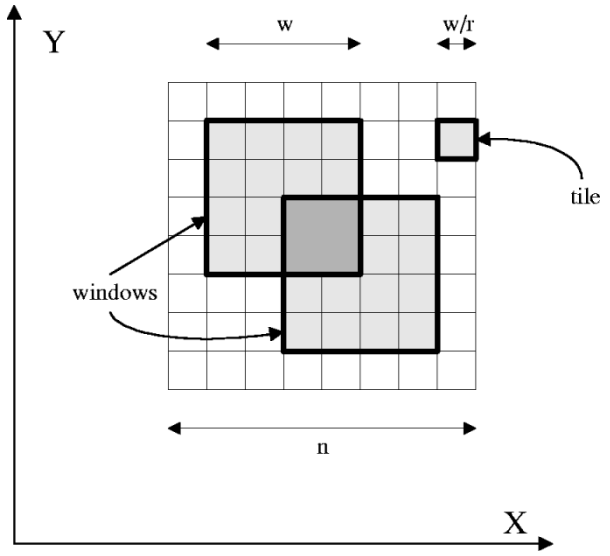

Fig. 2. The layout is partitioned using r^2 ($r = 4$ in this example) distinct dissections (each with window size $w \times w$) into $(nr/w) \times (nr/w)$ tiles. Each dark-bordered $w \times w$ window consists of r^2 tiles.

can incur substantial error, since other windows could still violate the density bounds.[3] A common industry practice is to enforce density bounds in r^2 overlapping *fixed dissections*, where r determines the "phase shift" w/r by which the dissections are offset from each other. In other words, to help control layout density in arbitrary windows, density bounds are enforced only for windows of the *fixed r dissection* (see Fig. 2), which partitions the $n \times n$-layout into *tiles* T_{ij}, then covers the layout by $w \times w$-windows W_{ij}, $i, j = 1, \ldots, (nr/w) - 1$, such that each window W_{ij} consists of r^2 tiles T_{kl}, $k = i, \ldots, i+r-1$, $l = j, \ldots, j+r-1$. Note that windows are "wrapped around" the layout, e.g., a window that overlaps with the upper edge of the layout also contains tiles on the bottom of the layout. This is not only convenient, but also reflects the fact that layout density at the edge of one die may affect the manufacturing of the die's neighbors on the wafer.

We seek to understand how the effective density depends on the spatial pattern density distribution in a window. The simplest model for $\rho(x, y)$ is the local area feature density, i.e., the window density is simply equal to the sum

$$\rho(W_{ij}) = \sum_{k=i}^{i+r-1} \sum_{l=j}^{j+r-1} \text{area}(T_{kl}) \tag{2}$$

where $\text{area}(T_{kl})$ denotes the original layout area of the tile T_{kl}. This model is due to [12], which solved the filling problem using linear programming.

A more accurate model considers the deformation of the polishing pad during the CMP process [7], where the effective local density $\rho(x, y)$ is calculated as the sum of *weighted* spatial pattern densities within the window, relative to an elliptical weighting function

$$f(x, y) = c_0 \exp\left[c_1 (x^2 + y^2)^{c_2}\right] \tag{3}$$

[3]The analysis in [12] bounds the error that results from considering only a finite number of windows, versus considering all possible windows.

with experimentally determined constants c_0, c_1, and c_2 [25]. The *discretized* effective local pattern density ρ for a window W_{ij} in the fixed-dissection regime (henceforth referred to as *effective window density*) is

$$\rho(W_{ij}) = \sum_{k=i}^{i+r-1} \sum_{l=j}^{j+r-1} \text{area}(T_{kl}) \cdot f\left(k - (i + r/2), l - (j + r/2)\right) \tag{4}$$

where the arguments of the elliptical weighting function f are the x and y distances of the tile T_{kl} from the center of the window W_{ij}.

IV. Fill Synthesis for Flat Layouts

A. Linear Programming Approaches

The LP approach seeks the optimum fill area p_{ij} to be inserted into each tile T_{ij}. The fill area p_{ij} cannot exceed $\text{slack}(T_{ij})$, which is the area available for filling inside the tile T_{ij} computed during density analysis. The first LP formulation for the Min–Var objective is [12]

Maximize: M

Subject to:

$$p_{ij} \geq 0 \qquad i, j = 0, \ldots, \frac{nr}{w} - 1 \tag{5}$$

$$p_{ij} \leq \text{slack}(T_{ij}) \qquad i, j = 0, \ldots, \frac{nr}{w} - 1 \tag{6}$$

$$\sum_{s=i}^{i+r-1} \sum_{t=j}^{j+r-1} p_{st} \leq \alpha_{ij} \left(U \cdot w^2 - \text{area}_{ij}\right), \qquad i, j = 0, \ldots, \frac{nr}{w} - r + 1 \tag{7}$$

$$M \leq \sum_{s=i}^{i+r-1} \sum_{t=j}^{j+r-1} \text{area}(T_{st}) + \sum_{s=i}^{i+r-1} \sum_{t=j}^{j+r-1} p_{st}, \qquad i, j = 0, \ldots, \frac{nr}{w} - r + 1 \tag{8}$$

where $\alpha_{ij} = 0$ if $\text{area}_{ij} > U \cdot w^2$ and $= 1$, otherwise.

Constraints (5) imply that we can only add features to, but cannot delete features from, any tile. The slack constraints (6) are computed for each tile: if a tile T_{ij} is originally overfilled, then we set $\text{slack}(T_{ij}) = 0$. The values of p_{ij} from the LP solution indicate the fill amount to be inserted into each tile T_{ij}. The constraints (7) ensure that no window can have density more than U after filling, unless it was initially overfilled. Inequalities (8) imply that the auxiliary variable M is a lower bound on all window areas which include the original feature areas $\text{area}(T_{st})$ and the fill areas p_{st}. The linear program seeks to maximize M, thus achieving the Min–Var objective.

A followup work [25] proposed the Min–Fill objective, along with a *Ranged Variation* LP

Minimize: $\sum_{i,j} p_{ij}$

Subject to:

Monte-Carlo Filling Algorithm

Input: $n \times n$ layout,
fixed r-dissection into tiles T_{ij}, $i, j = 0, \ldots, \frac{nr}{w} - 1$,
$slack(T_{ij})$ = slack of tile T_{ij},
$area(W_{ij})$ = area of $w \times w$ window W_{ij},
$unit_fill$ = unit filling area, and
U = upper bound on $w \times w$ window area.

Output: filled layout

1. **For** each tile T initialize
2. $insert_in(T) = 0$
3. $priority(T) = f(U, slack(T), MaxWin(T))$
4. **While** the sum of tile priorities is positive **Do**
5. Select a random tile T according to priorities
6. $insert_in(T) = insert_in(T) + 1$; $slack(T) = slack(T) - unit_fill$
7. **If** $slack(T) < unit_fill$ **Then** $priority(T) = 0$
8. **Else** $priority(T) = priority(T) - unit_fill$
9. **For** each window W containing T **Do**
10. $area(W) = area(W) + unit_fill$
11. **For** each tile $T' \in W$ **Do**
12. Update $priority(T')$ according to $area(W)$
13. **For** each tile T **Do**
14. Randomly perturb sequence of grid positions: $random(i) = 1, \ldots, slack(T)/unit_fill$
15. **For** $i = 1, \ldots, insert_in(T)$ **Do**
16. Insert a unit-fill geometry into the $random(i)^{th}$ grid position
17. **Output** the filled layout

Fig. 3. The Monte Carlo-based filling algorithm.

$$p_{ij} \geq 0 \qquad i, j = 0, \ldots, \frac{nr}{w} - 1 \tag{9}$$

$$p_{ij} \leq \text{slack}(T_{ij}) \qquad i, j = 0, \ldots, \frac{nr}{w} - 1 \tag{10}$$

$$L \leq \rho_0(i, j) \leq U \qquad i, j = 1, \ldots, \frac{nr}{w} - 1. \tag{11}$$

Here, $\rho_0(i, j)$ is the effective density of tile T_{ij}; L and U are the minimum and maximum tile effective densities, respectively.

We also note a variant LP for the Min–Var objective: given a target window density M (instead of an upper bound on window density), we minimize the variability budget ε

Minimize: ε

Subject to:

$$0 \leq p(T_{ij}) \leq \text{slack}(T_{ij}) \tag{12}$$

$$M - \varepsilon/2 \leq \rho(T_{ij}) \leq M + \varepsilon/2 \qquad i, j = 1, \ldots, \frac{nr}{w} - 1. \tag{13}$$

B. New Monte Carlo and Greedy Approaches

1) Min–Var Objective: The number of variables and the number of constraints in the linear program described above are both $O((nr/w)^2)$. Although the LP solution is optimal, it has several drawbacks. First, solving a very large linear program is too time consuming (expected runtimes are $O(v^3)$, where v is the number of variables in the LP). Second, an optimal solution for an r dissection is not necessarily an optimal solution for e.g., a $2r$ dissection and may also result in a high *floating* window density variation (i.e., density variation over all windows, not only over the fixed-r dissection). Third, rounding is another source of errors in LP formulations; when the tile size is sufficiently small, the problem becomes an instance of integer programming, and rounding errors become crucial.

Monte Carlo Approaches: Here we consider new approaches to the Filling Problem based on the Monte Carlo paradigm. Our goal is to develop a method with significantly better scaling properties than the LP formulation, without incurring a serious loss of solution quality. Our approaches can transparently handle both the spatial pattern density model as well as the effective pattern density model.

The Min–Var Monte Carlo algorithm (see Fig. 3) randomly chooses a tile and increments its content (i.e., spatial/effective density) by a prescribed fill amount. The probability of choosing a particular tile T_{ij} is referred as the *priority* of that tile. The iteration ends when either the sum of priorities of tiles is equal to zero, or when no tile's slack area is left.

Priorities: The Monte Carlo methods considered in this paper randomly choose a tile and increment its contents (i.e., area density) by a prescribed fill amount. The probability of choosing a particular tile T_{ij} defines the *priority* of that tile. The priority of a tile may depend on the density of the windows containing that tile, or else be independent of the window density. Note that the priority of a tile is zero if it belongs to a window which has already achieved the density upper bound U and is "locked" (see below).

We consider three different methods of computing tile fill priorities. The first method does not take into account the density of the windows containing the tile. We call this the *slack priority*, because it sets the priority to be equal to the slack of the tile. Intuitively, this means that we select a tile with probability proportional to its empty area, i.e., the choice of any available legal position of a fill geometry is uniform and independent. In order to take into account the density of windows we consider two more alternatives:

- *maximal* priority of the tile T_{ij} is proportional to $U - \text{MaxWin}(T_{ij})$;

- *minimal* priority of the tile T_{ij} is proportional to $U - \text{MinWin}(T_{ij})$;

where $\text{MaxWin}(T_{ij})$ and $\text{MinWin}(T_{ij})$ are the maximum and minimum densities over all windows containing T_{ij}, respectively.

The intuition behind the maximal priority is to first insert fill into tiles for which the upper density U is less likely to constrain the filling. In other words, we want to insert as much fill as possible before all tiles either exhaust their slack, or belong to a window with density U. On the other hand, the minimal priority scheme ensures a preference toward tiles which belong to the most underfilled windows. Thus, each such insertion of a filling geometry increases the current minimum window density with higher probability. The Monte Carlo algorithm with this minimal priority scheme can be viewed as a randomized greedy algorithm for solving the linear program [see (5)–(8)].

We may further increase the *relative* probabilities of selecting tiles with relatively higher (minimal or maximal) priorities. This is easily accomplished, e.g., by raising priorities to the power of 2 or 4 before normalizing them. Raising priorities to a higher power brings the Monte Carlo algorithm even closer to the greedy algorithm (which fills tiles in a deterministic order).

Updating the Priorities: Regardless of which priority scheme is used, it is essential to update the priority of tiles which belong to *locked* windows (i.e., windows with density U). Thus, when newly added fill causes a window to reach its maximum allowable density, all tiles in that window should be removed from the prioritization scheme, since they cannot be assigned any more fill. We propose two heuristic schedules for updating tile priorities after each fill geometry insertion. In the context of Fig. 3, these are:

H1) update the priorities of all affected tiles, i.e., execute all lines in the algorithm shown in Fig. 3;

H2) update the priorities only of tiles which belong to locked windows, i.e., in the algorithm of Fig. 3, omit Line 8 and execute the loop at Lines 11–12 only if window W achieves the maximum density U.

The above discussion implies that the underlying data structures must support two distinct operations, namely, priority-based tile selection and the efficient updating of priorities. One simple way of implementing tile selection is to: 1) arrange tiles in a one-dimensional array T_i, $i = 1 \ldots, k$; 2) create a list of sums of priorities $S_0 = 0$, $S_1, \ldots, S_k$, such that $S_{i+1} = S_i + \text{priority}(T_i)$; and 3) choose a random number in the range $(0, S_k)$ which will belong to some subinterval (S_{i-1}, S_i) corresponding to selection of the tile T_i. Such tile selection is very fast, but unfortunately priority updating requires $O(k)$ time on average. We recommend the *quadrisection approach* which recursively partitions the design into four quadrants and maintains the sum of priorities of all tiles in each quadrant. The runtime of our data structure is $O(\log k)$ per insertion.[4] Since heuristic schedule H2) updates priorities only once (i.e., when the window containing a tile is locked), the average insertion time will be much smaller for H2) than for H1) (see Table II).

[4]First, we select a random number R between 0 and the sum of all priorities. If R is greater than the priority of the first quadrant q_1, then we set $R = R - q_1$ and so on, until $R < q_i$. We then repeat this process recursively for all sub-quadrants of q_i. Finally, after at most $O(\log k)$ recursive steps, we will find the tile in which to insert fill. Priority updating can be done within the same time complexity, using a bottom-up approach.

Min-Fill Monte-Carlo Algorithm
Input: $n \times n$ filled layout, fixed r-dissection, $w \times w$ window, lower bound on window density L
Output: filled layout with minimized amount of inserted fill area
1. **While** there exist an unlocked tile **do** 2. Choose an unlocked tile T_{ij} randomly, according to its priority 3. Delete a filling geometry from T_{ij} 4. Update priorities of tiles 5. **Output** the resulting layout

Fig. 4. The Monte Carlo algorithm for the Fill-Deletion problem deletes fill geometries from randomly chosen unlocked tiles (i.e., tiles which still have filling geometries, but which belong to windows, having density greater than L).

Filling Schedule: A third family of implementation design choices depends on how many filling geometries may be inserted into a tile per iteration. We compare two alternatives: 1) insert into a tile T_{ij} a single fill geometry per iteration or 2) insert the maximum possible number of fill geometries which is $\min\{U - \text{MaxWin}(T_{ij}), \text{slack}(T_{ij})\}$.

Greedy Approaches: A variant of the Monte Carlo approach is the deterministic *Greedy algorithm.* At each step the Min–Var Greedy algorithm adds the maximum possible amount of fill into a tile with the highest priority, i.e., at each step a tile with the highest priority is locked. The performance of the Min–Var Greedy algorithm is illustrated in Table IV. Greedy run times are slightly higher than Monte Carlo run times, due to the necessity of finding highest priority rather than random tiles.

2) Min–Fill Objective: In the presence of two objectives, a natural strategy is first to find a solution that optimizes one of the objectives (Min–Var) and then modify that solution with respect to the other objective (Min–Fill), hopefully without degrading the solution quality relative to the first objective. Thus, the first objective (density variation) can hopefully be traded off toward a significant improvement in the second objective (the amount of inserted fill). This strategy can be implemented with an LP-based approach as follows.

1) Solve the Min–Var LP formulation with the given upper bound U on window density.
2) Decrease the obtained minimum window density M by a given amount $L = M \cdot (1 - \varepsilon)$.
3) Solve the Min–Fill LP formulation within the interval (L, U).

To implement the same strategy with either the Monte Carlo or this greedy approach, we assume that the density of each window is already within the given interval (L, U) and then solve the following.

Fill-Deletion Problem (With the Min–Fill Objective): Delete as much previously inserted fill as possible, while maintaining a minimum window density of no less than L.

To solve the Fill-Deletion problem using the Monte Carlo approach, we iteratively delete a fill geometry from a tile randomly chosen according to a certain priority. It is natural to choose this priority symmetrically to the priority in the Min–Var Monte Carlo algorithm, i.e., proportional to $\text{MinWin}(T_{ij}) - L$. Again symmetrically, no filling geometry can be deleted from the tile

Iterated Monte-Carlo and Greedy Filling Algorithms
Input: $n \times n$ layout, fixed r-dissection, $w \times w$ window, upper bound on window density U
Output: filled layout
1. **Repeat** forever 2. Run Min-Var Monte-Carlo (Greedy) Algorithm with the upper window density U 3. **If** resulting minimum window density equals the previous M **Then** exit repeat 4. Update the densities of tiles and windows and the minimum window density M 5. Run Min-Fill Monte-Carlo (Greedy) Algorithm with the lower window density M

Fig. 5. In the Iterated Monte Carlo and Greedy Filling approach, each iteration consists of two applications (with the Min–Var and Min–Fill objectives) of the Monte Carlo and Greedy algorithms.

T_{ij} (i.e., T_{ij} is *locked*) if and only if it either has zero priority, or else all fill previously inserted into T_{ij} has been deleted.

Thus, the Min–Fill Monte Carlo algorithm deletes fill geometries from unlocked tiles which are randomly chosen according to the above priority scheme (see Fig. 4). Similarly, the Min–Fill Greedy algorithm iteratively deletes a filling geometry from an unlocked tile with the currently highest priority.

Iterated Monte Carlo and Greedy Methods:

Min–Var Objective: As mentioned above, both the Monte Carlo and Greedy Algorithms are suboptimal for the Min–Var Objective, and although they are both fast in practice, the resulting minimum window density may be significantly lower than the optimum. We now propose *iterated* methods based on alternating the Min–Var and Min–Fill objectives (see Fig. 5), resulting in a monotonic narrowing of the gap between the upper window density bound U and the minimum window density L. Such iterated methods are still very fast and retain all the advantages of their noniterated Monte Carlo and Greedy counterparts, yet offer improved accuracy (see Table IV).

Min–Fill Objective: To solve the Filling Problem with the Min–Fill Objective, the Iterated Monte Carlo and Greedy Filling algorithms (see Fig. 5) may be modified as follows.

1) Interrupt the filling process as soon as the lower bound L on window density is reached, i.e., when $M = L$, instead of improving the minimum window density (while possible) for the Min–Var objective.
2) Continue iterating, but without changing the lower density bound $M = L$. Although this does not guarantee that the total filling area will not increase, an improved solution can typically be obtained if we keep track of the best solution seen over all iterations.

V. Fill Synthesis for Hierarchical Layouts

Most modern designs are hierarchical, with layout representations that are substantially more succinct than flat layouts, and that can be analyzed and processed more efficiently. The filling problem for hierarchical (standard-cell) layouts is similar to its flat layout counterpart, except that the hierarchical structure of master cells must be preserved, i.e., the same filling geometry is simultaneously added to *all* instances of the same master cell. The slack area of each cell can be filled independently and uniformly, as is the case when the size of fill geometries is sufficiently small.

A. Why Not Linear Programming

The filling constraints due to hierarchical characteristics make the LP approach for hierarchical filling problem infeasible. Instead of using $O((nr/w)^2)$ variables and constraints corresponding to each tile and window in the LP formulation for the flat fill problem, we must define the variables and constraints for each window, all instances of each master cell, all feasible fill positions in each master cell, and each window. This greatly increases the number of variables and constraints (e.g., the number of grid cells is much larger than number of tiles). The LP formulation is furthermore complicated by the transformations of master cell instances and the overlaps between the instances. Based on these considerations, the Monte Carlo method constitutes a much more feasible approach for the hierarchical filling problem than linear programming.

B. The Monte Carlo Method

Our proposed hierarchical filling algorithm (see Fig. 6) starts by computing the slack for all master cells. (Cell overlaps are possible and must be addressed carefully, as detailed below.) We then create buffer zones around master cells to avoid overfilling the regions near master cell boundaries. Master cells are then filled in a Monte Carlo fashion, according to a priority scheme where master cells that are more severely underfilled receive higher priority for filling at each iteration. This process continues until all master cells are filled past the lower bound density threshold, or until the slack in all underfilled master cells is exhausted.

C. Slack Computation for Hierarchical Layouts

For each master cell, area fill may be inserted only into the slack area of a master cell, not into its subcells. Computing the slack of a master cell proceeds by first determining the number of grid positions inside the bounding box of the master cell, while excluding all positions that overlap with either a "bloated" feature (i.e., a forbidden buffer zone around each feature) or a "bloated" subcell. However, slack area computation is complicated by the fact that instances of master cells may overlap. Such overlaps can occur between the master cell instance and the features, or between two or more master cell instances (see Fig. 7). In general, overlaps may have a very complicated structure. We distinguish the following cases:

1) the overlap between a master cell instance and a feature;

Monte-Carlo Hierarchical Filling Algorithm

Input: hierarchical layout, fixed r-dissection, buffer distance, $w \times w$ window, upper bound U on window density

Output: new hierarchical layout with filled master cells

```
1. For each Master Cell M_i in the layout Do
2.    Partition the Master Cell M_i according to the given grid size
3.    For all grids in the Master Cell Do
4.       Mark the status of grid "occupied" if it is covered by the original features or the sub Master Cell
5.    For all instances I_j of the Master Cell M_i Do
6.       If the instance I_j is overlapped with features or instances of other Master Cells Then
7.          Update the status of grids which are covered
8.  Calculate the priority of the Master Cells
9.  While the sum of priority > 0 Do
10.    Use the Monte-Carlo method to select one Master Cell M_i
11.    Randomly select a slack grid position in the master cell
12.    For each corresponding position of the grid in all instances of the Master Cell M_i Do
13.       If the insertion causes any window density to exceed the upper bound U on window density Then
14.          Discard the insertion and lock slack grid position
15.          Go over all other grid positions in master cell which are covered by the exceeded window and lock them
16.       Else Increase the fill area of the Master Cell
17.          Add the new fill geometry into the Master Cell
18.       Update the relevant windows' densities
```

Fig. 6. The Monte Carlo Hierarchical filling algorithm.

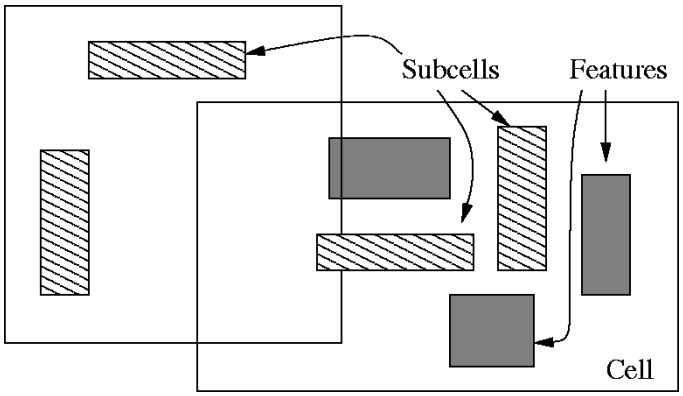

Fig. 7. Computing master cell intersections: the dark features and patterned subcells may either completely or partially overlap with a given master cell.

2) the overlap between two instances of different master cells;
3) the overlap between more than two instances of different master cells;
4) the overlap between two or more instances of the same master cell.

For each region of master cell overlap we must determine which master cell "owns" that intersection region. In other words, it is necessary to assign the space available for filling to the slack of a *single* master cell. We resolve this "ownership" problem by fixing a containment order over all master cells, starting from the global master cell (containing the entire layout), all the way down to individual features. This hierarchy can be represented as an acyclic directed graph H, with the set of nodes consisting of all master cells and individual features, and where there is an arc from a cell A to another cell or feature B, if and only if B participates in the definition of A.

The topological order of the graph H is a linear ordering of its nodes in such a way that all arcs point in the same direction (say, left-to-right). Such a topological order may be obtained by a breadth-first-search traversal of H, starting from the global master cell, and represents a *containment-based ordering* of the hierarchy where no master cell appearing later in the order may use in its definition any master cells appearing earlier in the order. For every intersection of master cell instances, we check which of the master cells appears later in the topological order and assign the intersection area to this master cell. This correctly resolves the overlap cases 1)–3) above. Unfortunately, case 4) cannot be resolved in this manner because hierarchy cannot distinguish different instances of the same master cell. Thus, we exclude such type-4) overlapping regions from the slack of master cells, thereby leaving such regions unavailable for fill.

D. Hybrid Hierarchical/Flat Filling Approaches

Pure hierarchical filling may tend to result in some sparse or unfilled regions (e.g., due to overlaps between different instances of master cells and features, or due to the interactions among the "bloat" regions around master cells), which could result in an unacceptably high layout density variation. A natural and simple solution is to apply a postprocessing "cleanup" phase, i.e., apply a standard flat fill algorithm to the output of the hierarchical phase. However, a purely flat fill approach, even when applied as a secondary postprocessing phase, may greatly increase the resulting data volume and runtime, negating the benefits of using a hierarchical approach in the first place.

We propose a new algorithm for mitigating this drawback, by combining hierarchical filling techniques with a flat filling approach, in a way that smoothly trades off the respective efficiency and accuracy of these two approaches. In our proposed method, varying a user-controlled parameter yields a smooth tradeoff among solution quality, data volume, and runtime, as confirmed by our computational experience. Our three-phase hybrid hierarchical-flat filling approach is summarized as follows:

1) a purely hierarchical fill phase;

k-Way Master Cell Splitting Algorithm

Input: hierarchical layout, and a splitting parameter k

Output: new hierarchical layout with new copies of master cells

1. **For** $i = 1$ to n **Do**
2. Create k new copies of C_i, namely $C_{i,1}, C_{i,2}, \ldots C_{i,k}$
3. **For** any master cell C' containing in the master cell C_i **Do**
4. **For all** $1 \leq j \leq k$ **Do**
5. Put an arc from the master cell $C_{i,j}$ to C'
6. **For** any master cell C which contains master cell C_i **Do**
7. Replace C_i inside C with copy $C_{i,j}$ for random j, $1 \leq j \leq k$
8. In the hierarchy H, replace the arc (C, C_i) with $(C, C_{i,j})$
9. **Output** resulting new hierarchical layout

Fig. 8. Improving the hierarchical filling approach by splitting master cells k-ways. Each master cell is replaced with k distinct masters, each of which may be filled independently and differently.

2) a split-hierarchical phase, where certain master cells that were deemed to be underfilled in phase 1, would be replicated so that distinct copies of the same master cell may be filled differently than other copies of the same master cell;
3) a flat fill "cleanup" phase (say, Monte Carlo based), which will fill any remaining sparse or unfilled regions that were not processed satisfactorily during the first two phases.

The overall goal with this strategy is to quickly fill as much of the layout as possible in phases 1 and 2 while keeping the fill output data volume relatively low, and then further improve and tune the resulting filled layout using a flat filling approach in phase 3 on the (presumably small number of) remaining sparse or unfilled areas.

In particular, phase 2 consists of repeatedly *splitting* master cells located in regions which were determined to be underfilled during phase 1, as follows. Given a top-down containment-based topological ordering of the n master cells, i.e., C_1, $C_2, C_3, \ldots, C_{n-2}, C_{n-1}, C_n$, where a master cell C_i can only contain master cell C_j if and only if $i < j$, a master cell C_i may be split into two master cells $C_{i,1}$ and $C_{i,2}$ and any C_j containing master cell C_i is then modified to point to either the copy $C_{i,1}$ or $C_{i,2}$ (say, randomly chosen). More generally, rather than performing only two-way splits, we can perform k-way splits (see Fig. 8).

Varying the parameter k (which controls the split factor) from 1 (pure hierarchical) to an arbitrarily large number (pure flat) yields a smooth tradeoff between solution quality, data volume, and runtime. As k is increased, the solution quality asymptotically approaches that of flat fill. If the result of hierarchical filling does not satisfy the technological constraints, we then recommend foregoing the original hierarchy in favor of a more uniform filling. This can be implemented by storing in the original cell library different *filled versions* of each master. Such a scheme will not necessarily slow down verification, since having fixed permanent structure, they can be "preverified," and thus dramatically improve the uniformity of hierarchical filling without a large runtime increase.

VI. Fill Synthesis for Multiple-Layer Layout

In the model of [28], topographic variation of each layer attenuates through subsequent CMP steps, each of which is modeled as a low-pass filter based on (3) and (4), according to the following:

$$\rho_{0\hat{(}k)} = \begin{cases} \left[\hat{d}_k + \left(\dfrac{z_{k-1}}{z_k}\right)\rho_{0(\hat{k}-1)}\right] \times \hat{f}, & k > 1 \\ \hat{d}_1 \times \hat{f}, & k = 1 \end{cases} \tag{14}$$

where "^" is the fast Fourier transform (FFT) operator, $\rho_{0\hat{(}k)}$ is the effective local density, z_k is the step height (i.e., the height of layer k from the first layer), d_k is the local density (all for layer k), and f is the weighting function. In the discussion below, we will not explicitly address the multiple-layer model. However, our linear programming and Monte Carlo algorithms have straightforward extensions for simultaneously handling multiple layers.

By mathematical induction on the layer number k and the linearity of Fourier transforms, (14) can be written as [25]

$$\rho_{0\hat{(}k)} = \sum_{l=1}^{k} \left[(z_l/z_k)\hat{f}^{k-l+1} \times \hat{d}_l\right] \tag{15}$$

Furthermore, in order to achieve effective density at location (i, j) on layer k, each term in the summation induces a multiple circular convolution in the physical domain

$$\begin{aligned} &\left[IFFT\left(\hat{f}^{(\alpha)} \times \hat{d}_l\right)\right](i, j) \\ &= [\overbrace{(f \otimes f \cdots f)}^{\alpha} \otimes d_l](i, j) \\ &= \sum_{i_1}\sum_{j_1}\Bigg[f(i_1 - i, j_1 - j) \\ &\qquad \times \cdots \left(\sum_{i_\alpha}\sum_{j_\alpha}[f(i_\alpha - i_{\alpha-1}, j_\alpha - j_{\alpha-1}) \right. \\ &\qquad \left. \times (x_{i_\alpha j_\alpha l} + x^0_{i_\alpha j_\alpha l})]\right)\Bigg]. \end{aligned} \tag{16}$$

Since a multiple convolution written as a series of summations is linear in term of $p_{i,j}$, all LP formulations can be easily extended to multiple layers.

A. Linear Programming Approaches for Multiple-Layer Fill

Wong *et al.* [25] extended the linear programming formulation to address multiple layers, with the objective of minimizing the sum of density variations over all layers

$$\text{Minimize: } \sum_{k=1}^{K}(H_k - L_k)$$

Subject to:

$$\begin{aligned} 0 \leq L_k \leq \rho_0(i, j, k) \leq H_k \leq \sum_{l=1}^{k}\left(\frac{z_l}{z_k}\right) \\ i, j = 0, \ldots, \frac{nr}{w} - 1, \qquad k = 1, \ldots, K \end{aligned} \tag{17}$$

Multiple-Layer Monte-Carlo Filling Algorithm

Input: layout with multiple layers, and fill feature size

Output: layout with multiple filled layers with respect to the minimized sum of density variations of all layers, or other objectives

```
1. For L= bottomLayer To topLayer Do
2.   For each tile[i][j] on layer L Do
3.     Compute its slack area slackAreaOfTile[L][i][j] and cumulative effective density mlEffDenOfTile[L][i][ j]
4.     mlSlackAreaOfTileStack[i][j] += slackAreaOfTile[L][i][j]
5.     mlEffDenOfTileStack[i][j] += mlEffDenOfTile[L][i][j]
6. Compute the priority of tile stacks according to mlEffDenOfTileStack[i][j]
7. While (sum of priorities > 0) Do
8.   Randomly select a tile stack TS(i,j) according to its priority
9.   For L = bottomLayer to topLayer Do
10.    If slackAreaOfTile[L][i][j] > 0 Then
11.      For every neighboring tile[L][m][n] located in (L+1) x (L+1) square Do
12.        If the insertion on tile[L][i][j] causes the neighboring tile meet the upperbound Then
13.          Exit loop
14.      Insert the fill feature into tile[L][i][j]
15.      Update the slackAreaOfTile[L][i][j] and mlSlackAreaOfTile[i][j]
16.      Update the priorities
17.      Exit loop
18.    Else Lock the tile on layer L
```

Fig. 9. The multiple-layer Monte Carlo filling algorithm.

Iterated Monte-Carlo and Greedy Filling Algorithms

Input: $n \times n$ multiple-layer layout with l layers, fixed r-dissection, $w \times w$ window, density upper bound U_l on each layer

Output: Filled multiple-layer layout

```
1. Repeat forever
2.   Run Min-Var Monte-Carlo (Greedy) Algorithm with the upper window densities U_l
3.   If resulting sum of density variations equals the previous solution Then Exit repeat
4.   Else
5.     While there exist an unlocked tile stack Do
6.       Choose an unlocked tile stack TS_ij randomly, according to its priority
7.       Choose a layer
8.       If the deletion does not deteriorate the solution Then
9.         Delete a fill feature from the layer
10.      Else
11.        Lock the tile stack
12.        Update the priorities of the tile stacks
```

Fig. 10. The iterated Monte Carlo and Greedy multiple-layer filling approach.

$$0 \le x_{ijk} \le \text{slack}(T_{ijk}) \qquad i, j = 0, \ldots, \frac{nr}{w} - 1, \qquad k = 1, \ldots, K. \quad (18)$$

Considering only the sum of layer variations in the objective function cannot guarantee that the filling on each layer will satisfy the Min–Var objective, abad polishing of an intermediate layer due to nonplanarization can potentially cause problems on subsequent (upper) layers. We therefore formulate a linear program for multiple-layer fill with the objective of minimizing the variance of density variations over all layers

Minimize: M

Subject to:

$$0 \le L_k \le \rho_0(i, j, k) \le H_k \le \sum_{l=1}^{k} \left(\frac{z_l}{z_k} \right) \qquad i, j = 0, \ldots, \frac{nr}{w} - 1, \qquad k = 1, \ldots, K \quad (19)$$

$$(H_k - L_k) \le M \qquad k = 1, \ldots, K \quad (20)$$

$$0 \le x_{ijk} \le \text{slack}(T_{ijk}) \qquad i, j = 0, \ldots, \frac{nr}{w} - 1, \qquad k = 1, \ldots, K. \quad (21)$$

B. Monte Carlo Approaches for Multiple-Layer Fill

LP-based methods for the fill problem have two main drawbacks: 1) solving large problems is very time consuming and 2) rounding errors adversely affect the solution quality. In this section, we address the multiple-layer fill problem using new Monte Carlo-based approaches.

For the multiple-layer fill problem where the objective is to minimize the sum of density variations over all layers, we define a *tile stack* as a column of tiles all having the same position on each layer [2]. We also define the density of a tile stack as the sum of densities of all the tiles in that tile stack. The Monte Carlo algorithm (see Fig. 9) assumes the original maximum density to be the upper bound for each layer. It then randomly chooses a

tile stack according to its priority value, selects a layer in that stack, and increments the tile's density as well as the tile stack's density by a prescribed fill amount, assuming that this insertion is permitted with respect to the overall objective.

The probability of choosing a particular tile stack TS_{ij} is referred to as the *priority* of that tile stack. Note that the priority of a tile stack TS_{ij} is zero if and only if either TS_{ij} has already achieved the density upper bound U, or the slack of TS_{ij} is less than the prescribed fill area. Tiles with zero priority are said to be *locked*. Based on our computational experience with single-layer fill synthesis, the priority of a tile stack TS_{ij} is chosen to be proportional to $U\text{-}EffDen(TS_{ij})$, where $EffDen(TS_{ij})$ is the effective density of the tile stack TS_{ij}.

During the algorithm's execution, after choosing which tile stack to fill next, we also need to decide into which layer the fill features should be inserted. We consider three different ways to choose the insertion layer. The first method entails choosing the bottom layer first, then trying an upper layer if the current layer is not feasible. The second method is to select the top layer first, then try a lower layer only if the current layer is not feasible. For these two approaches, once no layer is suitable for fill, the tile stack will be "locked" and will not be subsequently selected for any more filling. The third approach is to randomly choose one layer for fill, and then to try the upper layer or the lower layer with equal probability. Here, a tile stack is "locked" when it contains no remaining slack area. Our experimental results indicate that the second method described above outperforms the other two approaches.

A variant of the Monte Carlo approach is the deterministic *Greedy* algorithm. Unlike the Monte Carlo approach, at each step the Greedy algorithm adds a prescribed amount of fill into a tile with the highest priority. The experiments show that the run times of this Greedy approach are slightly higher than Monte Carlo's, due to the necessity of finding a tile with the highest priority, rather than a random tile.

We can also implement the Iterated Monte Carlo and Greedy methods for the multiple-layer filling problem; our approach again alternates between the Min–Var and Min–Fill objectives, resulting in a monotonic narrowing of the density variation (see Fig. 10). Such iterated methods are still very fast and retain all the advantages of the noniterated Monte Carlo and Greedy counterparts, but offer improved accuracy.

VII. Implementation Details

Our implementation of hierarchical filling for layout density control is further enhanced with the following important practical features:

• *Grid Slack Computation.* In previous academic and industry approaches, the area slack in each tile, i.e., the area available for filling, was assumed to be proportional to the total tile area minus the area of original features after the bloating of features by a certain buffer distance. However, such a calculation is quite optimistic because fill geometries have lower/upper bounds on their dimensions. That is also the reason the prescribed LP fill solution may not correspond to a legal filling. An alternative grid slack computation entails using an underlying grid and the actual fill pattern to compute the maximum number of legal positions for fill geometries in each tile. This method of slack calculation is more precise and realistic, since it guarantees that the calculated fill amount can actually be legally inserted into the corresponding tile.

• *Doughnut Area Computation.* In shallow-trench isolation processes, so-called reverse active-area mask steps lead to a density criterion whereby only the width-d "outer ring" of a large feature contributes to the effective density. Our tool is enhanced to optionally apply such a "doughnut" area computation.

• *Wraparound Window Density Analysis and Synthesis.* In the CMP process, the polishing pad typically polishes multiple neighboring dies simultaneously. Since all dies on a wafer are usually identical, we may assume that the rightmost tile of the layout is adjacent to the leftmost tile in the same row, and that the topmost tile is adjacent to lowest tile in the same column. In order to take this into account while performing density analysis as well as fill synthesis, the windows are thus "wrapped around" the layout so that a window overlapping with the upper (left) edge of the layout also contains tiles from the bottom (right) portion of the layout.

• *Different Pattern Types.* In order to reduce worst case coupling capacitance to fill, we may impose a constraint dictating that the same amount of fill area should be intersected by any vertical or horizontal line. To this end, a basket-weaving pattern has been suggested in [12]. Our implementation allows insertion of filling geometries either on a rectangular grid or in a basket-weaving manner. Our implementation can also easily support more exotic fill pattern types.

• *Compressed Fill Insertion.* In practice, filling can increase the size of the output GDSII file by a large factor (sometimes by more than an order of magnitude) due to the small size of the filling geometries and the possibly large amount of empty area in the original layout. We have, therefore, implemented a "compressed fill" approach which greatly reduces the size of the GDSII file (via the AREF construct). The basic idea here consists of utilizing several fill patterns of increasing size, e.g., one filling geometry, four filling geometries arranged in a 2-by-2 square, 16 filling geometries arranged in a 4-by-4 square, etc. The insertion phase tries to first fit in the largest feasible fill pattern, then gradually reduces the size of the fill pattern if it is no longer possible to insert any larger patterns. Naturally, for certain layouts with large empty areas, the file size reduction realized by this technique may even be exponential. Note that this is in itself a third form of hierarchy in our filling solution.

TABLE I
Parameters of Four Industry Test Cases

	Test Cases			
Testcase	**L1**	**L2**	**L1x4**	**L2x4**
layout size n	125,000	112,000	250,000	224,000
# rectangles k	49,506	76,423	198,024	305,692

VIII. Computational Experience

A. Experiments on Flat Layouts

Our experiments were performed using part of a metal layer extracted from an industry standard-cell layout.[5] (see Table I for details[6]) Benchmark L1 is the M2 layer from an 8131-cell

[5]Our experimental testbed integrates GDSII Stream input, conversion to CIF format, and internally developed geometric processing engines, coded in C++ under Solaris.

[6]In the given coordinate system, 40 units is equivalent to 1 micron.

TABLE II

MONTE CARLO METHODS WITH VARYING TILE SELECTION PRIORITIES AND UPDATING SCHEDULES. **NOTATION**: $T/W/r$: LAYOUT/WINDOW SIZE/r-DISSECTION; Max *IN LAYOUT INFO*: THE MAXIMUM WINDOW DENSITY IN THE ORIGINAL LAYOUT; Min *IN LAYOUT INFO*: THE MINIMUM WINDOW DENSITY IN THE ORIGINAL LAYOUT; Max_Pri = MAXIMAL PRIORITY; Min_Pri = MINIMAL PRIORITY; SLK_Pri = SLACK PRIORITY. THE COLUMNS Min AND *CPU* CORRESPOND TO THE MINIMUM WINDOW DENSITY OF THE RESULTING FILLED LAYOUT AND THE RUNTIME IN CPU SECONDS, RESPECTIVELY. THE MAXIMUM WINDOW DENSITY IN THE FILLED LAYOUT IS THE SAME AS IN THE ORIGINAL LAYOUT

			Heuristic I				Heuristic II					
Layout Info			Min_Pri		Max_Pri		Min_Pri		Max_Pri		SLK_Pri	
T/W/r	Max	Min	Min	CPU	Min	CPU	Min	CPU	Min	CPU	Min	CPU
L1/31/2	0.20201	0.10548	0.19354	3.30	0.19336	2.21	0.19345	1.01	0.19327	1.00	0.19254	0.97
L1/31/3	0.20712	0.09683	0.19186	9.96	0.19102	5.96	0.19148	1.33	0.19093	1.31	0.19571	1.32
L1/31/4	0.21248	0.09369	0.19870	26.29	0.19811	15.18	0.19778	1.67	0.19660	1.69	0.19505	1.67
L1/31/5	0.21449	0.09097	0.19950	57.08	0.19871	32.35	0.19874	2.08	0.19847	2.07	0.19678	2.08
L1x4/31/2	0.21075	0.08739	0.15132	13.38	0.15132	10.08	0.15124	4.72	0.15044	4.66	0.14948	4.64
L1x4/31/3	0.21511	0.07808	0.14765	38.00	0.14765	25.10	0.14765	5.78	0.14765	5.72	0.14762	5.66
L1x4/31/4	0.21489	0.10775	0.19192	90.59	0.19101	54.50	0.19027	6.59	0.18977	6.55	0.19002	6.49
L1x4/31/5	0.21462	0.10103	0.18454	187.16	0.18445	109.58	0.18336	7.67	0.18307	7.65	0.18164	7.57
L2/28/2	0.18076	0.05065	0.11353	4.70	0.11327	3.98	0.11301	1.67	0.11411	1.68	0.11305	1.67
L2/28/3	0.22651	0.05125	0.14774	20.98	0.14538	15.98	0.14527	2.87	0.14612	2.86	0.14944	2.87
L2/28/4	0.21827	0.08072	0.17866	49.30	0.17796	35.05	0.17810	3.30	0.17814	3.29	0.17912	3.28
L2/28/5	0.23764	0.07203	0.17121	100.84	0.16703	78.74	0.16535	3.92	0.16582	3.91	0.16830	3.99
L2x4/28/2	0.22327	0.05011	0.17217	44.54	0.16472	34.63	0.16712	13.88	0.16450	13.90	0.16129	13.59
L2x4/28/3	0.20957	0.05087	0.12437	90.25	0.12514	69.53	0.12286	13.12	0.12234	13.19	0.12364	12.99
L2x4/28/4	0.22412	0.05010	0.17105	242.11	0.16867	176.95	0.16887	17.41	0.16908	17.33	0.16407	17.18
L2x4/28/5	0.23771	0.05005	0.16841	516.86	0.16482	373.06	0.16538	21.36	0.16285	21.32	0.16037	21.00

TABLE III

OPTIMAL LP FILLING COMPARED WITH THE MONTE CARLO APPROACH USING DIFFERENT FILLING SCHEDULES. **NOTATION**: *Filling_1*: INSERTING A SINGLE FILLING GEOMETRY INTO A TILE PER ITERATION; *Filling_2*: INSERTING THE MAXIMUM POSSIBLE FILLING GEOMETRIES INTO A TILE PER ITERATION. THE COLUMNS Min AND *CPU* CORRESPOND TO MINIMUM WINDOW DENSITY OF THE FILLED LAYOUT AND THE RUNTIME IN CPU SECONDS, RESPECTIVELY. THE MAXIMUM WINDOW DENSITY IN THE FILLED LAYOUT IS THE SAME AS IN THE ORIGINAL LAYOUT. WE DID NOT FILL ONE OF THE ENTRIES IN THE TABLE DUE TO THE PROHIBITIVELY LARGE RUNNING TIME OF THE LP METHOD FOR HIGHER VALUES OF r

			LP Method		Heuristic I				Heuristic II			
Layout Info					Filling_1		Filling_2		Filling_1		Filling_2	
T/W/r	Max	Min	Min	CPU	Min	CPU	Min	CPU	Min	CPU	Min	CPU
L1/31/2	0.20201	0.10548	0.20119	0.11	0.19354	3.30	0.18036	0.11	0.19345	1.01	0.17834	0.17
L1/31/3	0.20712	0.09683	0.20026	0.23	0.19186	9.96	0.18763	0.12	0.19148	1.33	0.18218	0.14
L1/31/4	0.21248	0.09369	0.20084	0.57	0.19870	26.29	0.19176	0.35	0.19778	1.67	0.19164	0.22
L1/31/5	0.21449	0.09097	0.20328	1.75	0.19950	57.08	0.19212	0.56	0.19874	2.08	0.19634	0.49
L1/8/2	0.26966	0.02080	0.15968	1.52	0.15868	6.32	0.13249	0.20	0.15868	2.29	0.14865	0.36
L1/8/3	0.27043	0.03151	0.17174	11.54	0.17162	14.78	0.16882	1.14	0.17162	2.54	0.16635	1.12
L1/8/4	0.27375	0.03362	0.18261	39.66	0.18282	39.97	0.17834	3.76	0.18282	3.53	0.17763	4.54
L1/8/5	0.27213	0.02766	0.14901	60.80	0.14827	67.45	0.13564	9.21	0.14827	3.90	0.13678	1.42
L1/4/2	0.28250	0.00544	0.16734	24.47	0.16771	7.31	0.11763	0.92	0.16771	2.79	0.11143	3.97
L1/4/3	0.27807	0.00911	0.13792	67.61	0.13229	14.31	0.13102	2.99	0.13229	3.17	0.11784	2.72
L1/4/4	0.28250	0.00950	0.16914	395.95	0.16452	44.07	0.14987	9.57	0.16452	5.28	0.15212	38.36
L1/4/5	0.28237	0.00390	0.12928	335.0	0.12385	88.91	0.11373	22.70	0.12385	8.58	0.11234	45.82
L1x4/31/2	0.21075	0.08739	0.15845	35.9	0.15132	13.38	0.09343	0.23	0.15124	4.72	0.0902	0.87
L1x4/31/3	0.21511	0.07808	0.15082	378.9	0.14765	38.00	0.09188	2.34	0.14765	5.7	0.10465	1.52
L1x4/31/4	0.21489	0.10775	0.19812	1864.3	0.19192	90.59	0.11473	2.37	0.19027	6.5	0.11274	3.59
L1x4/31/5	0.21462	0.10103	N/A	N/A	0.18454	187.16	0.11241	3.22	0.18336	7.67	0.10945	4.84

design, and Benchmark L1 $\times$ 4 is the same layout replicated four times in a 2 $\times$ 2 array to create a larger test case. Benchmark L2 is the M3 layer from a 20 577-cell layout. L2 $\times$ 4 is this same layout replicated four times in a 2 $\times$ 2 array.

Table II compares the CPU runtimes and the original and resulting minimum window densities for the minimal, maximal, and slack priorities. All data provided in this paper are the average values of ten runs, and all experiments assume that U is equal to the maximum window density of the original layout. All run times are in CPU seconds on a 300-MHz Sun Ultra-5_10 with 640 MB of RAM. These results indicate a tradeoff between runtime and accuracy for different priorities; the fastest slack priority has the lowest accuracy and the minimal priority, while the slowest slack priority has the highest accuracy. The best choice seems to be the maximal priority, which is almost as accurate as the minimal priority, but considerably faster.

TABLE IV
ITERATED GREEDY (IGREED) AND ITERATED MONTE CARLO (IMC) ALGORITHMS ARE MORE ACCURATE THAN THE NONITERATED VERSIONS (GREED AND MC), AND ARE FASTER THAN A LINEAR PROGRAM-BASED APPROACH (LP)

	Orig. Density		LP		Greed		MC		IGreed		IMC	
Test case	Max	Min	Min	CPU	Min	CPU	Min	CPU	Min	CPU	Min	CPU
Spatial Density Model												
L1/32/8	0.21447	0.10414	0.19864	41.5	0.18779	18.2	0.19221	17.3	0.19871	26.9	0.19871	24.8
L1/32/16	0.21783	0.10088	0.19768	1077.5	0.19044	21.9	0.19410	19.6	0.19779	98.1	0.19740	93.5
L1/16/8	0.26452	0.07803	0.17519	161.1	0.17556	21.8	0.17556	18.9	0.17556	36.7	0.17556	30.2
L1/16/16	0.26452	0.08551	N/A	N/A	0.18868	44.2	0.18868	23.4	0.18868	202.3	0.18868	168.9
L2/32/8	0.22648	0.07039	0.14467	43.0	0.14257	25.5	0.13565	24.4	0.14469	41.3	0.14463	68.6
L2/32/16	0.22648	0.07650	0.15093	2716.0	0.14621	33.8	0.14459	29.4	0.14971	538.5	0.14940	317.2
L2/16/8	0.33022	0.04552	0.17926	1912.4	0.16709	42.1	0.17748	30.5	0.17980	170.1	0.17980	169.4
L1x4/32/8	0.21693	0.09657	0.18643	255.7	0.18183	82.6	0.18282	72.3	0.18648	131.9	0.18648	111.9
L1x4/32/16	0.21793	0.10263	N/A	N/A	0.19574	124.3	0.19547	80.2	0.19933	632.8	0.19933	565.1
L2x4/32/8	0.22226	0.05776	0.14647	532.6	0.14480	150.7	0.13824	117.7	0.14649	289.5	0.14655	469.7
Effective Density Model												
L1/32/8	0.41625	0.16255	0.31970	32.4	0.31859	22.8	0.31994	22.3	0.31994	26.5	0.31994	23.9
L1/32/16	0.46662	0.10626	0.28249	105.5	0.28353	27.4	0.28353	24.0	0.28353	33.2	0.28353	27.8
L1/16/8	0.46662	0.10626	0.28249	105.2	0.28353	27.1	0.28353	23.1	0.28353	32.8	0.28353	26.0
L1/16/16	0.48313	0.05693	N/A	N/A	0.24748	49.7	0.24748	27.1	0.24748	74.2	0.24748	33.4
L2/32/8	0.53585	0.07249	0.34777	66.8	0.34538	39.7	0.31153	38.3	0.34629	49.5	0.33858	68.9
L2/32/16	0.84446	0.03514	0.35956	520.5	0.36007	57.4	0.34049	41.4	0.36007	67.9	0.35276	107.4
L2/16/8	0.84446	0.03514	0.35956	526.7	0.36007	57.3	0.34206	40.1	0.36007	68.7	0.35120	90.1
L1x4/32/8	0.43270	0.14665	0.28487	171.5	0.28505	107.2	0.28505	90.7	0.28505	126.5	0.28505	100.9
L1x4/32/16	0.46740	0.10494	0.28732	1238.8	0.28835	177.0	0.28835	106.3	0.28835	262.4	0.28835	125.5
L1x4/16/8	0.46740	0.10494	0.28732	1387.8	0.28835	188.8	0.28835	106.3	0.28835	266.6	0.28835	121.5
L1x4/16/16	0.48313	0.05160	N/A	N/A	0.27197	586.0	0.27197	119.5	0.27197	975.0	0.27197	150.1
L2x4/32/8	0.52179	0.04467	0.34176	637.4	0.32008	241.5	0.30799	165.6	0.33620	342.0	0.33524	435.9

Table III compares the optimal results obtained by solving the linear program[7] [see (5)–(8)] from [13] with two fill schedule heuristics. Our results show that the accuracy of the Monte Carlo method is very high. In all of our test cases the resulting variation is no more than 5% larger than the optimal solution obtained by the LP method. On the other hand, the Monte Carlo method is much faster than the LP method. When the window size is small and/or the number of fixed dissections is large, the LP method becomes impractical, while our new method is still fast.

The exhaustive comparison of different tile priorities and updating schedules shows that the minimal-priority updating is the best choice for the Monte Carlo method (see Table III). On the other hand, the faster filling schedule, which fills a chosen tile with the maximum possible number of filling geometries, loses in terms of performance, e.g., in some cases the window density variation does not even change (see Table III).

The runtime advantage of the Monte Carlo methods may be leveraged to obtain more uniform filling. We apply the faster Monte Carlo method to larger numbers of fixed dissections, resulting in filled layouts which are more uniform than those obtainable with the LP method. For instance, the LP method applied to the layout with parameters (L1/4/4) yields a filled layout with a density variation of 15% measured for $r = 16$ fixed dissections. On the other hand, the Monte Carlo method with the slack priority, minimal updating, and single-geometry filling schedule applied to the same layout but for $r = 16$ fixed dissections, yields a filled layout with a density variation less than 14%. Moreover, the LP method requires almost two minutes of CPU time, while the Monte Carlo method takes only 10 s to run.

[7]In this paper, *lp_solve 3.0* is used as the linear programming solver.

TABLE V
PARAMETERS OF THE TEST CASES

Test Cases			
Testcase	**Case1**	**Case2**	**Case3**
layout size	260000	288000	504000
# rectangles k	216	432	540

Table IV compares the minimum window density and the associated run times for the minimum variation linear program, Greedy algorithm, Monte Carlo algorithm, Iterated Greedy algorithm, and Iterated Monte Carlo algorithm. The table consists of two parts, corresponding to the spatial and effective density measures, respectively. The left column of Table IV reports for each test case the window size (in thousands of units) as well as the number r of fixed dissections.

The smaller r-value corresponds to the maximal value for which the LP approach can still produce the optimal minimum window density within a reasonable run time, and the larger r-value is selected sufficiently high to demonstrate the accuracy of the suggested heuristics. The next two table columns report the maximum and minimum window densities of the original layout before filling.

Table IV indicates that the iterated methods are more accurate than previous noniterated approaches, that they are more efficient than LP-based methods, and that they yield more even filling or larger number of tiles (corresponding to larger r). Finally, note that the iterated Monte Carlo and Greedy algorithms can output better solutions than LP-based approaches, since the LP's rounding errors become more significant for larger r.

TABLE VI
HIERARCHICAL, FLAT, AND HYBRID FILLING APPROACHES. **NOTATION**: *data*: DATA VOLUME, I.E., THE NUMBER OF FILL GEOMETRY REFERENCES IN THE RESULTING GDSII OUTPUT FILE; *# fill*: NUMBER OF FILL FEATURES IN THE RESULTING LAYOUT; MinDen: MINIMUM WINDOW DENSITY ACROSS THE LAYOUT; *Hier*: HIERARCHICAL FILLING APPROACH; EXTSL$H + F$: HIERARCHICAL + FLAT FILLING APPROACH; $H + S$: HIERARCHICAL + 2-WAY MASTER CELL SPLITTING FILLING APPROACH; $H + S + F$: HIERARCHICAL + 2-WAY MASTER CELL SPLITTING + FLAT FILLING APPROACH; *Flat*: FLAT FILLING APPROACH

Density Model	Spatial Density Model			Effective Density Model		
	data	#fill	MinDen	data	#fill	MinDen
	Testcase 1					
Original Layout			0.070			0.291
Hier	645	5136	0.11	1054	2608	0.369
H+F	1562	6053	0.335	2758	4312	0.655
H+S	2321	7601	0.17	1552	4166	0.525
H+S+F	2834	8114	0.339	2908	5522	0.676
Flat	5219	5219	0.403	5732	5732	0.735
	Testcase 2					
Original Layout			0.167			0.145
Hier	2081	16060	0.272	2142	16972	0.248
H+F	2451	16430	0.393	5630	17460	0.320
H+S	4368	17494	0.410	4531	18126	0.365
H+S+F	4374	17500	0.421	7234	20829	0.383
Flat	13974	13974	0.527	23415	23415	0.443
	Testcase 3					
Original Layout			0.000			0.091
Hier	4995	22566	0.071	4449	20320	0.157
H+F	7472	25043	0.532	9461	25332	0.371
H+S	9690	23622	0.102	8575	22990	0.159
H+S+F	12212	26144	0.540	13285	25700	0.394
Flat	17695	17695	0.547	31204	31204	0.483

B. Experiments on Hierarchical Layouts

Table V lists the attributes of our three test cases, i.e., the layout dimension N and the number of rectangles k.

Table VI compares the minimum window density, data volume (i.e., the number of fill geometry references in the resulting GDSII output file), and the number of area fill features (i.e., the number of fill geometries in the resulting layout after flattening) for five heuristics: 1) hierarchical; 2) flat; 3) 2-way splitting; 4) hybrid of hierarchical and flat; and 5) hybrid of the hierarchical, splitting and flat approaches. For each test case, we ran all the five filling heuristics under both the spatial density model and the effective density model, with the window density upper bound equal to the original maximum window density.

Table VI indicates that the Flat Monte Carlo approach yields the best quality results (i.e., highest minimum density), but also produces the largest output data volumes. On the other hand, the Hierarchical Monte Carlo approach saves on data volume, but yields low-quality results. The hybrids of the hierarchical and flat fill approaches produce substantially improved results, with only a modest increase in data volume. Finally, we observe that the k-way Master Cell Splitting approach smoothly trades off performance and data volume, i.e., it provides better results than the pure Hierarchical Fill approach, yet produces less data volume than the pure Flat Filling approach.

C. Experiments on Multiple-Layer Flat Layouts

The layouts used in our multiple-layer filling experiments have either two or three layers with the same dimensions. Table VII shows the performances of the two LP formulations proposed in [25] and by us, respectively, for the different multiple-layer fill objectives. The experiments indicate that the LP formulations designed to minimize the sum of density variations across all layers cannot at the same time also minimize the maximum density variation.

Table VIII compares the sum of density variations on all layers and the associated run times for the linear programming method (LP0), Greedy method, Monte Carlo (MC) method, Iterated Greedy (IGreedy) method, and Iterated Monte Carlo (IMC) method. Our results show that the accuracy of the Monte Carlo/Greedy methods is very high. When the window size is small and/or the number of fixed dissections is large, the LP method becomes impractical for the multiple-layer fill problem,[8] while the Monte Carlo/Greedy methods are still fast. On the other hand, the rounding errors inherent in the LP method make its performance worse than the Monte Carlo/Greedy methods on the large test cases.

Table IX shows the performance of the Linear Programming method (LP1), Greedy method, Monte Carlo method, Iterated Greedy method, and Iterated Monte Carlo method with respect to the maximum density variation objective across all layers. The Monte Carlo and Greedy methods yield better solutions than the LP-based approaches on these test cases within shorter run times.

IX. CONCLUSION

We developed a new Monte Carlo approach for layout density control and compared several criteria to decide where to insert fill geometry. The Monte Carlo method is scalable to large designs, yet offers accuracy competitive with previously known linear programming based approaches. We also presented a new unified approach to capturing different models of layout density control for CMP. This enables the application of Greedy and Monte Carlo methods that simultaneously address different filling objectives for spatial and effective density definitions. Our new iterated Greedy and Monte Carlo methods are more accurate and practical than previous LP-based methods. We also discuss and compare extensions of the linear programming and Monte Carlo approaches to multilayer designs.

For hierarchical layouts, we proposed a practical approach to *hierarchical* fill synthesis for layout density control, which trades off runtime, solution quality, and output data volume. Our approach allows distinct copies of a master cell to be filled differently, which improves solution quality in a user-controlled manner. Our system also generates filling geometries in compressed GDSII format, which reduces the resulting fill data volume. Experiments indicate that this new hybrid hierarchical filling approach is scalable, efficient, and highly competitive with previous Monte Carlo and linear programming-based methods.

Ongoing research includes developing alternate pure-hierarchical filling heuristics and developing more robust hierarchy manipulators for in-memory layout representations, in order to enable even more robust tradeoffs between solution quality and data volume. We also seek to make our fill solutions reusable,

[8]For example, the LP0 method did not terminate for the test case L6/8/5 after running for more than 12 h.

TABLE VII
PERFORMANCE OF THE LP FORMULATIONS UNDER THE OBJECTIVES OF MINIMIZING: I) THE SUM OF DENSITY VARIATIONS AND II) THE MAXIMUM DENSITY VARIATION, ON ALL LAYERS. **NOTATION**: $L/W/r$: LAYOUT/WINDOW SIZE/r-DISSECTION; *LP0*: THE LINEAR PROGRAMMING FORMULATIONS FOR MINIMIZING THE SUM OF DENSITY VARIATIONS ON ALL LAYERS; *LP1*: THE LINEAR PROGRAMMING FORMULATIONS FOR MINIMIZING THE MAXIMUM DENSITY VARIATION ACROSS ALL LAYERS; SumVar: THE SUM OF DENSITY VARIATIONS ON ALL LAYERS; maxDenVar: THE MAXIMUM DENSITY VARIATION ACROSS ALL LAYERS; *CPU*: THE RUN TIME; *Area*: THE NUMBER OF INSERTED FILL FEATURES

Testcase	LP0				LP1			
L/W/r	SumVar	maxDenVar	CPU	Area	SumVar	maxDenVar	CPU	Area
L4/16/4	0.2690	0.1696	42.0	20921	0.2875	0.1666	37.6	19609
L4/8/4	0.6626	0.4696	44.2	14769	0.6626	0.4696	43.2	14330
L5/16/4	0.3436	0.2420	101.0	38152	0.3843	0.1932	69.8	38241
L5/8/4	1.0585	0.5531	279.7	34942	1.0621	0.5393	655.7	33376
L6/16/4	0.5986	0.4080	91.3	65578	0.6333	0.3737	71.0	62113
L6/8/4	1.6116	1.1155	12617.0	67178	1.6584	1.0903	6649.0	65576

TABLE VIII
PERFORMANCE OF LP0, GREEDY, MC, IGREEDY, AND IMC FOR THE SUM OF DENSITY VARIATIONS ACROSS ALL LAYERS. **NOTATION**: $L/W/r$: LAYOUT/WINDOW SIZE/r-DISSECTION; SumVar: THE SUM OF DENSITY VARIATIONS ACROSS ALL LAYERS; *CPU*: THE RUN TIME

Testcase	LP0		Greedy		MC		IGreedy		IMC	
L/W/r	SumVar	CPU	SumVar	CPU	SumVar	CPU	SumVar	CPU	SumVar	CPU
L4/16/8	0.6626	33.1	0.6420	36.3	0.6285	36.6	0.6285	37.7	0.6285	33.9
L4/16/5	0.5435	30.7	0.5541	32.2	0.5535	33.0	0.5535	31.1	0.5535	30.5
L4/8/8	0.9031	140.1	0.7794	48.1	0.7766	36.2	0.7762	74.7	0.7762	34.5
L4/8/5	0.8351	33.4	0.7882	35.4	0.7804	32.7	0.7804	39.1	0.7804	30.7
L5/8/8	2.2118	8093.0	2.0526	102.8	2.0913	65.4	2.0526	111.7	2.0716	67.6
L5/8/5	1.3494	8879.0	1.3450	65.0	1.3943	54.2	1.3252	79.6	1.3476	59.3

TABLE IX
PERFORMANCE OF LP1, GREEDY, MC, IGREEDY, AND IMC WITH RESPECT TO MAXIMUM DENSITY VARIATION ACROSS ALL LAYERS. **NOTATION**: $L/W/r$: LAYOUT/WINDOW SIZE/r-DISSECTION; MaxDen: THE MAXIMUM DENSITY VARIATION ACROSS ALL LAYERS; *CPU*: THE RUN TIME

Testcase	LP1		Greedy		MC		IGreedy		IMC	
L/W/r	MaxDen	CPU	MaxDen	CPU	MaxDen	CPU	MaxDen	CPU	MaxDen	CPU
L4/16/8	0.4696	34.8	0.4459	36.3	0.4454	36.6	0.4454	37.7	0.4454	33.9
L4/16/5	0.3638	36.5	0.3638	30.2	0.3635	33.0	0.3635	31.1	0.3635	32.5
L4/8/8	0.6255	120.8	0.5437	48.1	0.5410	36.2	0.5406	74.7	0.5406	34.5
L4/8/5	0.5897	33.2	0.5576	35.4	0.5497	32.7	0.5497	39.1	0.5497	30.7
L5/8/8	1.2174	761.3	1.1081	102.8	1.1089	65.4	1.1081	111.7	1.1081	67.6
L5/8/5	0.6886	524.0	0.6857	65.0	0.7050	54.2	0.6698	79.6	0.6746	59.3

so that fill solutions can be stored in a library along with the master cells, thus would not have to be recomputed from scratch in cases where a cell is used in a context that has different density constraints. However, the reusability methodology currently can only be applied to the master cells which neither overlap with other master cells, nor require routing over their area. One way of achieving such "unrollable" solutions is to produce and store a fill solution in a "monotone" manner, so that successively longer prefixes of a given fill solution would still constitute valid fill solutions in lower density contexts.

REFERENCES

[1] SC Solutions, Inc.. [Online]. Available: http://www.best.com/~solvers/cmp.html.

[2] Y. Chen, A. B. Kahng, G. Robins, and A. Zelikovsky, "Monte Carlo methods for chemical–mechanical planarization on multi-layer and dual-material models," in *Proc. SPIE Conf. Design and Process Integration for Microelectronic Manufacturing*, Santa Clara, CA, Mar. 2002.

[3] ——, "Hierarchical dummy fill for process uniformity," in *Proc. ASP-Design Automation Conf.*, Jan. 2001, pp. 139–144.

[4] ——, "Practical iterated fill synthesis for CMP uniformity," in *Proc. Design Automation Conf.*, Los Angeles, CA, June 2000, pp. 671–674.

[5] ——, "New Monte Carlo algorithms for layout density control," *Proc. ASP-Design Automation Conf.*, pp. 523–528, 2000.

[6] CMP Technology, Inc... [Online]. Available: http://www.cmptechnology.com/.

[7] R. R. Divecha, B. E. Stine, D. O. Ouma, J. U. Yoon, and D. S. Boning *et al.*, "Effect of fine-line density and pitch on interconnect ILD thickness variation in oxide CMP process," in *Proc. CMP-MIC*, 1998.

[8] J. G. Garofalo, J. Q. Zhao, J. Blatchford, and E. Nease, "Applications of enhanced optical proximity correction models," *Proc. SPIE Optical Microlithography XI*, vol. SPIE 3334, Feb. 1998.

[9] W. Grobman, private communication, Aug.–Sept. 2001.

[10] W. Grobman, M. Thompson, R. Wang, C. Yuan, R. Tian, and E. Demircan, "Reticle enhancement technology: Implications and challenges for physical design," in *Proc. Design Automation Conf.*, Las Vegas, NV, 2001, pp. 73–78.

[11] W. Grobman *et al.*, "Reticle enhancement technology trends: Resource and manufacturability implications for the implementation of physical designs," in *Proc. Int. Symp. Physical Design*, 2001, pp. 45–51.

[12] A. B. Kahng, G. Robins, A. Singh, H. Wang, and A. Zelikovsky, "Filling algorithms and analyzes for layout density control," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 445–462, Apr. 1999.

[13] A. B. Kahng, G. Robins, A. Singh, and A. Zelikovsky, "New and exact filling algorithms for layout density control," in *Proc. IEEE Int. Conf. VLSI Design*, Jan. 1999, pp. 106–110.

[14] H. Landis, P. Burke, W. Cote, W. Hill, and C. Hoffman *et al.*, "Integration of chemical–mechanical polishing into CMOS integrated circuit manufacturing," *Thin Solid Films*, vol. 220, no. 20, pp. 1–7, 1992.

[15] G. Y. Liu, R. F. Zhang, K. Hsu, and L. Camilletti, "Chip-level CMP modeling and smart dummy for HDP and conformal CVD films," in *Proc.CMP-MIC*, Feb. 1999.
[16] W. Maly, "Moore's law and physical design of ICs," in *Proc. ISPD*, 1998, Special address.
[17] V. Mehrotra, S. Nassif, D. Boning, and J. Chung, "Modeling the effects of manufacturing variation on high-speed microprocessor interconnect performance," in *Proc. Int. Electron Devices Meeting*, San Francisco, CA, Dec. 1998.
[18] G. Nanz and L. E. Camilletti, "Modeling of chemical–mechanical polishing: A review," *IEEE Trans. Semiconduct. Manuf.*, vol. 8, pp. 382–389, Apr. 1995.
[19] Praesagus, Inc.. [Online]. Available: http://www.praesagus.com/.
[20] J. Rey, private communication, 2000.
[21] SIA, *The National Technology Roadmap for Semiconductors*: Semiconductor Industry Assoc., Dec. 1997.
[22] Int. Technology Roadmap for Semiconductors. (1999, Dec.). [Online]. Available: http://www.itrs.net/1999_SIA_Roadmap/Home.htm.
[23] B. Stine, "A closed-form analytical model for ILD thickness variation in CMP processes," in *Proc. CMP-MIC*, 1997.
[24] B. Stine *et al.*, "The physical and electrical effects of metal-fill patterning practices for oxide chemical–mechanical polishing processes," *IEEE Trans. . Electron. Devices*, vol. 45, Mar. 1998.
[25] R. Tian, D. Wong, and R. Boone, "Model-based dummy feature placement for oxide chemical–mechanical polishing manufacturability," in *Proc. Design Automation Conf.*, June 2000, pp. 667–670.
[26] R. Tian, X. Tang, and D. F. Wong, "Dummy feature placement for chemical–mechanical polishing uniformity in a shallow trench isolation process," in *Int. Symp. Physical Design*, Apr. 2001, pp. 118–123.
[27] M. Tomozawa, "Oxide CMP mechanisms," *Solid State Technol.*, vol. 40, no. 7, pp. 169–175, 1997.
[28] T. Yu, S. Cheda, J. Ko, M. Roberton, A. Dengi, and E. Travis, "A two-dimensional low pass filter model for die-level topography variation resulting from chemical mechanical polishing of ILD films," in *Proc. IEDM*, Washington, DC, Dec. 1999.

Yu Chen obtained the M.S. degree in computer science and engineering from Zhejiang University, in 1998, and the M.S. degree in computer science from University of California at Los Angeles (UCLA), in 2000. He is currently pursuing the Ph.D. degree at UCLA.

His research interests include VLSI physical design, performance analysis, combinatorial optimization, and computational commerce.

Andrew B. Kahng received the A.B. degree (in applied mathematics/physics) from Harvard College and the M.S. and Ph.D. degrees (computer science) from the University of California at San Diego (UCSD).

He was with the UCLA Computer Science Department from 1989 to 2000, most recently as Professor and Vice-Chair. Since January 2001, he has been a Professor of CSE and ECE at UCSD. He has published over 180 papers in the VLSI CAD literature, focusing on physical layout and performance analysis. His research interests include VLSI physical layout design and performance analysis, the semiconductor design–manufacturing interface, combinatorial and graph algorithms, and stochastic global optimization.

Dr. Kahng received the National Science Foundation Young Investigator Award and Best Paper awards at DAC, ASPDAC, and ISQED. He was the founding General Chair of the ACM/IEEE International Symposium on Physical Design, co-founder of the ACM Symposium on System-Level Interconnect Prediction, and since 1997 has defined the physical design roadmap for the SIA International Technology Roadmap for Semiconductors. He has been the Chair of the U.S. Design Technology Working Group, and the International Design Technology Working Group, for the 2001 ITRS and 2002 ITRS update. He is also General Chair of EDP-2002 (the Electronic Design Processes Workshop of the IEEE DATC) and on the steering committees of ISPD and SLIP.

Gabriel Robins received the Ph.D. degree in computer science, in 1992, from the University of California, Los Angeles.

He is a Professor in the Department of Computer Science at the University of Virginia. His research interests include VLSI CAD, with emphasis on physical design, as well as computational geometry, combinatorial optimization, and genomics. He coauthored a book on high-performance routing, as well as over 70 refereed papers. He is a member of the U.S. Army Science Board, an alumni of the Defense Science Study Group, and has served on other advisory panels to the U.S. Department of Defense.

Dr. Robins is a recipient of a Packard Foundation Fellowship, a National Science Foundation Young Investigator Award, a University Teaching Fellowship, an All-University Outstanding Teaching Award, a Faculty Mentor Award, and the Walter N. Munster Endowed Chair, as well as an IBM Fellowship and a Distinguished Teaching Award. He authored a Distinguished Paper at the 1990 IEEE International Conference on Computer-Aided Design. He served on panels of the National Academy of Sciences and the National Science Foundation. He was General Chair of the 1996 ACM/SIGDA Physical Design Workshop and a cofounder of the 1997 International Symposium on Physical Design. He also served on the technical program committees of several other leading conferences, and on the Editorial Board of the IEEE Book Series. He is Associate Editor of IEEE TRANSACTIONS ON VLSI, and a Member of ACM, SIGDA and SIGACT.

Alexander Zelikovsky received the Ph.D. degree, in 1989, in computer science from the Institute of Mathematics of the Belorussian Academy of Sciences in Minsk, Belarus.

He worked at the Institute of Mathematics in Kishinev as a Senior Research Scholar from 1989 to 1995. Between 1992 and 1995, he visited Bonn University and the Institut fur Informatik in Saarbrueken (Germany). He was a Research Scientist at the University of Virginia (1995–1997) and a Postdoctoral Scholar at UCLA (1997–1998) as well. Since January 1999, he has been an Assistant Professor in the Computer Science Department at Georgia State University, Athens. He is the author of more than 70 refereed publications. His research interests include VLSI physical design, performance analysis, design for manufacturing, discrete and approximation algorithms, combinatorial optimization, and computational geometry.