Filling and Slotting : Analysis and Algorithms*

Andrew B. Kahng, Gabriel Robins†, Anish Singh†, Huijuan Wang and Alexander Zelikovsky

UCLA Department of Computer Science, Los Angeles, CA 90095-1596
†Department of Computer Science, University of Virginia, Charlottesville, VA 22903-2442
{abk,huijuan,alexz}@cs.ucla.edu, {robins,as6f}@cs.virginia.edu

Abstract

In very deep-submicron VLSI, certain manufacturing steps – notably optical exposure, resist development and etch, chemical vapor deposition and chemical-mechanical polishing (CMP)– have varying effects on device and interconnect features depending on local characteristics of the layout. To make these effects uniform and predictable, the layout itself must be made uniform with respect to certain *density* parameters. Traditionally, only foundries have performed the post-processing needed to achieve this uniformity, via insertion ("filling") or partial deletion ("slotting") of features in the layout. Today, however, physical design and verification tools cannot remain oblivious to such foundry post-processing. Without an accurate estimate of the filling and slotting, RC extraction, delay calculation, and timing and noise analysis flows will all suffer from wild inaccuracies. Therefore, future place-and-route tools must efficiently perform filling and slotting prior to performance analysis within the layout optimization loop. We give the first formulations of the *filling and slotting problems* that arise in layout post-processing or layout optimization for manufacturability. Such formulations seek to add or remove features to a given process layer, so that the local area or perimeter density of features satisfies prescribed upper and lower bounds in all *windows* of a given size. We also present efficient algorithms for density analysis as well as for filling/slotting synthesis. Our work provides a new unification between manufacturing and physical design, and captures a number of general requirements imposed on layout by the manufacturing process.

1 Introduction

As CMOS technology advances to the 180nm generation and beyond, the manufacturing process has an increasingly constraining effect on physical layout design and physical verification. Foundry economics dictate that process window volumes be maximized; this in turn dictates that device and interconnect features be fabricated as predictably and uniformly as possible. On the other hand, the physics of semiconductor processing make large process windows and uniform manufacturing difficult [3] [11] [8] [4]. In particular:

1. optical interference effects in lithography can create "iso-dense" effects, where the exposure intensity for isolated features is different from that for densely packed features;
2. reaction dynamics in resist development and etch, as well as chemical vapor deposition, can exhibit microloading effects where local variations in the density of exposed feature surface area result in line width or gate length variations across the chip; and
3. uniformity of CMP, which is used for planarization of interlayer dielectrics (or glass, with newer shallow-trench isolation) in multi-layer interconnect processes, depends on uniformity of features on the interconnect layer beneath a given dielectric layer to avoid dishing and other irregularities.

In this paper, we are concerned primarily with (2) and (3). The connection to (1) – that process-induced constraints on layout should not hamper separate optimizations related to proximity effects (i.e., OPC) – is understood. To minimize the impact of manufacturing process physics on device yield, foundries impose *density* rules so that the layout becomes more uniform. For example, a local interconnect metal layer might have a requirement that every $10\mu m \times 10\mu m$ window contain at least 35 μm^2, but no more than 70 μm^2, of metal features [1] [10]. Many process layers, including diffusion and even thin-ox, can have associated density rules [1] [14].[1] While empty areas must be *filled*, very wide features (e.g., power buses on top-layer metal) must be *slotted* to avoid lift-off in CMP.

Traditionally, only foundries or specialized TCAD tools companies have performed the post-processing of layout needed to achieve this uniformity, via insertion ("filling") or partial deletion ("slotting") of features in the layout. Today, however, ECAD tools for physical design and verification cannot remain oblivious to such post-processing. Without an accurate estimate of the downstream filling and slotting at the foundry, all the RC extraction, delay calculation, timing, noise and reliability analyses will be inaccurate, leading to a broken design flow. For instance, slotting will change the cross-section of a power bus, which in turn affects peak current density and reliability.

In the Appendix, we present analyses showing the extent to which *metal* filling and slotting can affect the results of capacitance extraction and performance analysis. Other analyses in the Appendix show that filling and slotting can, for the most part (and especially for the filling case), be viewed in the *single-layer* context. The precise locations of fill/slot geometries on a given layer will not significantly affect performance of interconnect on a neighboring layer. Rather, the dominant effects (coupling to dense geometries on neighboring layers, shielding from farther layers, and shielding of non-adjacent same-layer coupling) stem from lower bounds on area density on all layers. This enables us to address the metal filling and slotting problems one layer at a time.[2]

Our Contributions

In this paper, we give the first formulation of the *filling and slotting problems* that arise in layout post-processing or layout optimization for manufacturability. Essentially, we seek to add or remove features to a given process layer, such that the local area or perimeter density of features satisfies prescribed upper and lower bounds in all *windows*

*Research at UCLA was supported by a grant from Cadence Design Systems, Inc. Professor Robins was supported by a Packard Foundation Fellowship and by NSF Young Investigator Award MIP-9457412. Papers by these researchers may be found at http://vlsicad.cs.ucla.edu and http://www.cs.virginia.edu/~robins/ .

[1] For example, in $0.35\mu m$ and below, one major semiconductor house requires diffusion area density between 0.25 and 0.40, and metal area density between 0.40 and 0.70. Another major semiconductor house requires metal area density to be at least 0.35. Note that density rules and post-processing solutions may differ between, e.g., ASIC and high-end microprocessor technologies, due to tradeoffs between device performance and predictability.

[2] There are several notable conditions under which the single-layer assumption fails. Slotting approaches (e.g., for power buses) must avoid slotting contacted areas. Thus, any slotting synthesis approach must either perform re-layout of power distribution (unlikely) or else specifically mark contacted rectangles within power buses as inviolate. Filling approaches must pay attention to adjacent layers if they contain drawn geometries outside geometries on the layer to be filled. For example, poly fill geometry in regions with underlying active diffusion can create spurious transistors, and again regions must be marked as inviolate on the poly layer.

of given size. After formally defining the filling and slotting problems, we present efficient algorithms for density analysis as well as filling/slotting synthesis. The remainder of this section defines notation and gives a general statement of the filling and slotting problem. Then, Section 2 gives new algorithms for analysis of minimum- and maximum-density (i.e., *extremal-density*) windows. All of these methods can optionally return all extremal-density windows, or all violating windows, with the same time complexity needed to return a single extremal-density window. Section 3 establishes performance bounds for two practical approximations for the filling and slotting problem, namely, when only windows of *fixed dissections* of the layout region need to satisfy density bounds. Section 4 develops new algorithms for synthesis of filling/slotting geometries, and we conclude by listing several directions for future research.

Notation and Problem Statement

We will use the following notation and definitions.

- The input is a *layout* consisting of rectangular *geometries*.[3]
- $c \equiv$ smaller of the minimum feature width and minimum feature separation. The value of c is typically 25 to 50 times the manufacturing unit.
- $w \equiv$ fixed *window* size. The window is the moving square area over which density lower and upper bounds apply. A typical window size would be $w = 50 \cdot c$.
- $n \equiv$ size of the layout region side. Typically, we might see n as an integer which is about $50,000 \cdot c$. Note that c does not imply that $\frac{n}{c}$ is "the size of the grid": the only grid that is guaranteed is the manufacturing grid, which is typically 25 to 50 times smaller than c.
- $k \equiv$ the complexity of the original layout, i.e., the total number of rectangles in the input.
- $L_a, U_a \equiv$ *area density* lower and upper bounds expressed as real numbers $0 < L_a \leq U_a < 1$. Each $w \times w$ region of the layout must contain total area of features (considered as a fraction of the quantity $w \times w$) satisfying these bounds.
- $L_p, U_p \equiv$ *perimeter density* lower and upper bounds expressed as real numbers $0 < L_p \leq U_p < 1$. In practice, the maximum perimeter density is attained in memory cores, and the bounds L_p and U_p are set with respect to this maximum density. Consider a $w \times w$ window filled with as many small $c \times c$ squares as possible, where the origins of the small squares are offset from each other by integer combinations of the vectors $(0, 2c)$ and $(2c, 0)$. We consider the total perimeter of these squares to be the maximum possible feature perimeter. Then, each $w \times w$ region of the layout must contain total perimeter of features (considered as a fraction of the maximum possible feature perimeter) satisfying the given bounds.
- $L_d, U_d \equiv$ *density* lower and upper bounds expressed as real numbers $0 < L_d \leq U_d < 1$. It turns out that most of our results and algorithms easily apply to either the area density or perimeter density regimes. Thus, we will generically indicate the density bounds using L_d and U_d.[4]
- An *extremal-density* window is a window with either maximum density or minimum density over all windows in the layout. If an algorithm applies to either maximum-density or minimum-density analysis, we generically refer to extremal-density analysis.

Given the definitions and parameters above, we define the Filling and Slotting Problem as follows:[5]

> **The Filling and Slotting Problem.** Given a design rule-correct geometry of k disjoint rectilinear rectangles in an $n \times n$ layout region, a minimum feature size c, area and/or perimeter density upper and lower bounds L_a, U_a, L_p and U_p, and a window size $w < n$, add fill and slot geometries into the layout while preserving circuit function and design rule-correctness such that every $w \times w$ window in the layout region satisfies the lower and upper bounds on area and perimeter density.

2 Algorithms for Density Analysis

Before addressing the Filling and Slotting Problem, we first develop algorithms for density analysis (with respect to either area or perimeter) in a given layout. Given a fixed layout and window size, we shall determine a maximum-density and a minimum-density window (i.e., our analysis will return extremal-density window(s)). Our density analysis methods can report *all* violations of density bounds in the layout within the same time complexity needed to report a single extremal-density window (see Section 2.5). We state the density analysis problem as follows:

> **Extremal-Density Window Analysis.** Given a fixed window size w and a set of k disjoint rectangles in an $n \times n$ layout region, find an extremal-density $w \times w$ window in the layout.

This section presents a series of algorithms for the Extremal-Density Window Analysis problem. We first present a density analysis algorithm with time complexity $O(n^2)$ that is strictly a function of the layout size. We then develop a different algorithm with time complexity $O(k^2)$ that is strictly a function of the number of rectangles. Finally, we propose an algorithm with even faster expected runtime. Note that the $O(n^2)$ and $O(k^2)$ time complexities are incomparable in terms of efficiency, since k^2 can sometimes be much smaller than n^2 (e.g., $k = 100$ and $n = 10^4$) and at other times much larger (e.g., $k = 10^5$ and $n = 10^4$). Therefore, our choice of algorithm for density analysis would depend on the exact values of n and k, with overall time complexity of the "hybrid" approach being $O(min(k^2, n^2))$.

2.1 ALG1: $O(n^2)$ Density Analysis

Our first algorithm for density analysis has time complexity $O(n^2)$, and operates as follows.

1. Initialize an $n \cdot n$ boolean array B to all 0's, and then put 1's in array positions corresponding to areas in the layout that are covered by the k rectangles. This takes time $O(n^2)$.
2. Create another $n \cdot n$ array S and initialize each $S[i, j]$ to be equal to the number of 1's appearing in the southwest quadrant of array B with respect to coordinate $[i, j]$ (i.e., $S[i, j]$ counts the number of 1's in the subarray $B[1..i, 1..j]$). This can be done by scanning B one row at a time from left to right, maintaining a running sum of the 1's encountered on all the rows, and storing all these partial sums into the array S. All this preprocessing requires a total of $O(n^2)$ time.

[3] Our implementation reads in layouts from GDSII Stream format. Without loss of generality, our discussion below assumes that rectilinear geometries have been fractured into, say, horizontally maximal rectangles. It is also possible to generalize our analyses and algorithms from rectangles to trapezoids. Note that standard industry tools, such as Cadence Dracula, will fracture geometries into horizontal trapezoids [2].

[4] To our understanding, current foundries have not yet imposed *both* area density and perimeter density bounds *simultaneously* on a given layer [14]. However, we expect that such simultaneous constraints will be required in future technologies, and we analyze fill pattern synthesis for such a situation in Section 4 below.

[5] Note that this is a *satisficing* formulation where we seek only a feasible solution, as opposed to an *optimization* formulation where we seek a best solution. We can easily give variant optimization formulations (e.g., insert as little metal as possible, minimize the sum of window density deviations from an ideal density, etc.). However, it appears that the current state of technology does not yet require such formulations.

3. After this preprocessing phase, the density of an arbitrary-size $w \times h$ rectangle with its bottom-left corner located at an arbitrary position (i, j) can be found in constant time, as follows:

$$\begin{aligned} & \text{density}(w \times h \text{ rectangle at } (i, j)) \\ = \; & S[i+w, j+h] - S[i+w, j] - S[i, j+h] + S[i, j] \end{aligned}$$

This formula uses the principle of inclusion-exclusion: the fourth term is added in the formula above since it is implicitly subtracted *twice* by the middle two terms. The technique is analogous to efficient range tally queries in computational geometry [9].

In particular, the density of all $O(n^2)$ windows of fixed size $w \times w$ can be determined in $O(1)$ time per window, i.e., a total of $O(n^2)$ time. All extremal-density windows can be determined using the same technique within the same time complexity. This method given is considerably more general than is required to solve the extremal-density window problem, in that the preprocessing enables the future solution of arbitrary dynamic queries in constant time per query for *any* window size $w \times h$. Thus, w and h are both (variable) parameters in the query input, rather than fixed (as is the case in practice) over all input instances.

2.2 Properties of Extremal-Density Windows

To obtain an algorithm with time complexity that is strictly a function of k (as opposed to a function of n), we first prove a result that is analogous to Hanan's Theorem for the rectilinear Steiner minimal tree problem [6]. The *Hanan grid* over a given layout is formed by creating vertical and horizontal lines that pass through all the sides of all the rectangles (Figure 1).[6]

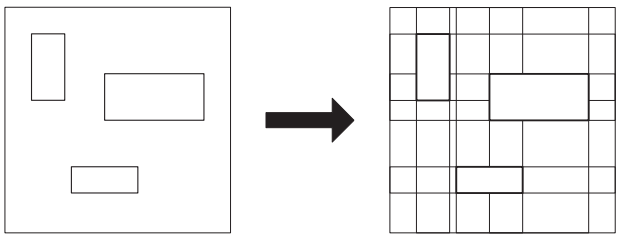


Figure 1: A layout (left) and its corresponding Hanan grid (right).

Theorem 1 *Given a layout of k rectilinearly-oriented rectangles in the $n \times n$ grid and a fixed window size w, there exists a $w \times w$ maximum-density window having at least one of its corners at a vertex of the Hanan grid.* □

Theorem 1 actually establishes a stronger result than coinciding a vertex of the maximum window with a Hanan grid point: it shows that there always exists a maximum-density window that touches rectangles of the layout with at least two of its sides (these sides might touch the *same* layout rectangle). This observation helps us to design an efficient algorithm for density analysis, since it limits the possible locations of a maximum-density window (i.e., abutting either one or two of the layout rectangles). The argument used to prove Theorem 1 can also be used to establish an analogous result for *minimum*-density windows.

[6]Here and elsewhere in what follows we state our result for maximum-density windows, explaining the extension to minimum-density windows only if there is some possibility of confusion. All proofs are omitted due to space constraints, but are available in our technical report [7].

Corollary 2 *Given a layout of k rectilinearly-oriented rectangles in the $n \times n$ grid and a fixed window size w, there exists a $w \times w$ window with extremal area density that abuts layout rectangles with at least two of its sides.* □

Notice that a type of geometric symmetry/duality is present here, in that layout rectangles abut the *interior* of maximum-density windows, and abut the *exterior* of minimum-density windows. Finally, a similar argument establishes analogous results for windows having maximum or minimum *perimeter* density.

Corollary 3 *Given a layout of k rectilinearly-oriented rectangles in the $n \times n$ grid and a fixed window size w, there exists a $w \times w$ window with extremal perimeter density that abuts layout rectangles with at least two of its sides.* □

2.3 ALG2: $O(k^2)$ Density Analysis

Recall that Theorem 1 shows that an extremal-density window must touch rectangles of the layout with at least two of its sides. We observe that there are only $O(k)$ sides of rectangles in the first place, and that $O(k^2)$ density analysis can be achieved essentially by (i) defining a window for each of these $O(k)$ rectangle sides, and (ii) computing in $O(k)$ time the window's intersections with all rectangles as it slides along the rectangle side. This yields an algorithm with overall time complexity of $O(k^2)$.

We preprocess by sorting all left and right edges of the k rectangles by their x coordinates into a single sorted list L (having up to $2k$ elements), within $O(k \log k)$ time. In the main loop (2), for each "pivot" rectangle R, we create a $w \times w$ window W that abuts R on the top and right (i.e., so that their top-right corners coincide - see Figure 2(a)). We then compute the density of W in $O(k)$ time by intersecting W with all k rectangles of the layout (Step (3) of the algorithm).

In the inner loop (4), we slide the window W horizontally to the right (Figure 2(a-c)) until it leaves R, updating the density of W each time its left or right edge intersects an edge in the list L. Note that the perimeter and area density of the window W increase or decrease monotonically between such intersection events.[7] We update the value of area density, or the two values of perimeter density, for W in constant time per intersection event by keeping track of the total "cross section" length of the current intersections between the rectangles and the left and right edges of W. We add new intersections that enter the window W as it advances horizontally, and we subtract from the total the areas of rectangles that exit the window W on the left during the sliding process. Finally, we repeat Steps 3 through 5 for all other $O(1)$ starting orientations of W with respect to the pivot rectangle R (Figure 2(d-f)). The overall time complexity of this algorithm is dominated by the $O(k)$ scans which require $O(k)$ time each. A formal definition of the algorithm is given in Figure 3.

2.4 ALG3: Fast Expected Time Density Analysis

Charging $O(k)$ time for each scan in the ALG2 analysis is pessimistic, since each sliding window is expected to intersect only a small fraction of the total number of rectangles (the window size is typically very small compared with the overall layout area). For each pivot rectangle, it would be advantageous to scan through only the few rectangles that actually intersect its associated sliding window (as opposed to scanning all k rectangles).

We implement this speedup via a new *fixed-dissection preprocessing* step, modifying our algorithm from Figure 3. The layout area is first partitioned into $\frac{n}{w} \times \frac{n}{w}$ squares of size $w \times w$ each. Then, for each such square we create a list of rectangles intersecting it; doing this for all squares requires a single pass through all rectangles. The main loop

[7]The area density is a continuous function and all its minima or maxima occur only at such intersections. The perimeter density has discontinuities when a window edge crosses a vertical feature edge. Therefore, at such intersection events we maintain both possible values of perimeter density (i.e., with and without the vertical feature edge).

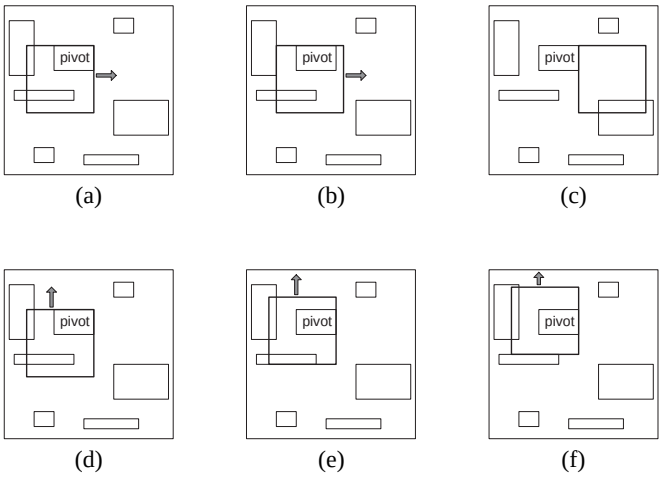


Figure 2: ALG2 starts a window abutting a *pivot rectangle* (a) and slides the window to the right, stopping at each edge that intersects its perimeter (b), until the pivot abuts the opposite side of the window, on the outside (c). Other combinations of the pivot-window orientations are then explored (d-f). This process is repeated for every rectangle, using each as a pivot in turn.

ALG2: $O(k^2)$ **Density Analysis**
Input: $n \times n$ layout with k rectangles **Output:** all extremal-density $w \times w$ windows
(1) Sort all the left and right edges of all k rectangles by x coordinates into a sorted list L (2) For each "pivot" rectangle R do (3) Find the density of a $w \times w$ window W that abuts R on the top and right (4) While W intersects R do (5) Slide W to the right to the next point of intersection with one of the edges on the list L Record changes in density (6) Repeat steps (3-5) for all other starting orientations for W **Output** all extremal-density windows

Figure 3: ALG2: $O(k^2)$ density analysis.

of the algorithm checks the rectangle intersections for a given $w \times w$ query window W by examining four lists of rectangles (corresponding to the four squares that together cover W).

Theorem 4 *Given k non-overlapping rectangles with positions uniformly distributed in the $n \times n$ grid, the algorithm from Figure 3 finds the maximum-density $w \times w$ window in time $O(k \cdot E)$, after applying a fixed-dissection preprocessing phase with runtime $O((\frac{n}{w})^2 + (\frac{n}{w})^2 \cdot E \cdot \log((\frac{n}{w})^2 \cdot E))$, where E is the expected number of rectangles that intersect an arbitrary $w \times w$ window.* □

We call this improved-preprocessing algorithm ALG3. We can show that the expected number of rectangles that intersect a given fixed-size window is indeed quite small.

Theorem 5 *Given k arbitrarily-sized disjoint rectangles located at random positions chosen from a uniform distribution inside the $n \times n$ layout region, the expected number E of rectangles that intersect a given $w \times w$ window is bounded by $E = O(k \cdot (\frac{w}{n})^2)$.* □

By the previous two theorems, substituting $E = O(k \cdot (\frac{w}{n})^2)$ into the overall time complexity of $O((\frac{n}{w})^2 + (\frac{n}{w})^2 \cdot E \cdot \log((\frac{n}{w})^2 \cdot E) + k \cdot E)$ yields:

Corollary 6 *Given k rectangles in the $n \times n$ layout region, the maximum-density width-w window can be found in time $O((\frac{n}{w})^2 + k \log k + k^2 \cdot (\frac{w}{n})^2)$.* □

Note that because a window cannot contain more than $O(w^2)$ rectangles, the expected time complexity of ALG3 is also bounded by $O((\frac{n}{w})^2 + k \log k + k \cdot w^2)$. The same algorithm and expected time bounds will hold for finding minimum-density windows, as well as for extremal-perimeter density analysis.

2.5 Remarks

Our algorithms, as stated, address only the problem of finding a single extremal-density window. However, they all implicitly find and report *all* windows having extremal density. In fact, all of the algorithms above will detect *every* window of the layout whose density violates either of the given density thresholds (either lower or upper).[8] This information can be reported by printing any extremal density encountered at the end of every scan phase involving each pivot element, if its value violates a density bound. Reporting all density violations in this manner does not increase the running time of any of our algorithms.

We must also re-emphasize that all our techniques outlined above extend in a straightforward way to computing extremal-density windows with respect to total perimeter. For example, to adapt the $O(n^2)$ algorithm of Section 2.1 to perform perimeter density analysis, Step (1) of that algorithm should mark the locations in array B that correspond to the perimeters of the k rectangles. Then, Step (2) of the algorithm of Section 2.1 will add up the total perimeter lengths in each point's south-west quadrant. After all this preprocessing, arbitrary-window extremal-perimeter queries can be performed in constant time per query. To adapt sliding-window area density analyses to the detection of extremal-perimeter density windows, we keep track of the total rectangle perimeter inside the sliding window, rather than the total rectangle area. The only caveat is that consistency must be exercised in deciding which grid points are considered to be occupied by particular rectangle perimeters. Finally, all algorithms described above work for any non-square $w \times h$ query window, even if w and h are input parameters (as opposed to being fixed over all input layouts).

3 Fixed-Dissection Density Analyses

In attempting to verify (or satisfy) upper and lower density bounds for $w \times w$ windows, a very practical method is to check (or enforce) these constraints only for $w \times w$ windows of a *fixed dissection* of the layout into $\frac{w}{r} \times \frac{w}{r}$ *tiles*, i.e., the set of windows having top-left corners at points $(i \cdot \frac{w}{r}, j \cdot \frac{w}{r})$, for $i, j = 0, 1, \ldots, r(\frac{n}{w} - 1)$; here r is an integer divisor of w. To our knowledge, this is the type of verification that is most often performed by commercial tools.[9] Unfortunately, a fixed-dissection scheme for small r cannot guarantee *any* nontrivial density bounds over *all* $w \times w$ windows (as opposed to only the fixed tiles in the dissection). For $r = 1$, even if the area density of each tile in the fixed dissection is guaranteed to be at least 75%, a completely empty $w \times w$ tile can exist. Conversely, if the area density of each window in the fixed dissection is guaranteed to be at most 25%, a completely full $w \times w$ window can exist.

[8] Practically speaking, this is the most common use model: a designer would like to know all areas of his layout that violate density bounds, so that these areas can be fixed or an exception granted by the project management. Any design may have numerous special cases that require exceptions, e.g., pads and scribe line areas.

[9] As an example of a fixed-dissection -based commercial analysis tool, consider the Dracula COVERAGE command [2], or capabilities of mask analysis tools in the TCAD marketplace [14]. Dracula COVERAGE, for example, allows checking of area density upper and lower bounds in $w \times w$ windows (e.g., $w = 50\mu m$) that occur at a fixed offset, or *step* (e.g., $\frac{w}{r} = 10\mu m$ and $r = 5$), from each other.

On the other hand, the analysis of fixed dissections can be done much faster than the analysis of all eligible $w \times w$ windows. First we initialize an array of $\frac{n}{w} \times \frac{n}{w}$ counters associated with all of the fixed dissection windows, and then for each rectangle R, we increment the counters of the windows intersecting R by the area of the intersection. In case of $r > 1$, we repeat the procedure above r^2 times in order to check all $(r \cdot \frac{n}{w})^2$ windows.

In the rest of the section, we seek ways in which density bounds for arbitrarily located windows can be enforced by density bounds on fixed dissection windows. Such rules can be viewed as a form of density-related layout design rule. We compare two ways of applying simple local rules to windows having top-left corners at points $(i \cdot \frac{w}{r}, j \cdot \frac{w}{r})$, $i, j = 0, 1, ..., \frac{n}{w}$ for some $r > 1$ such that $\frac{w}{r}$ is an integer. First, we consider what happens when we enforce upper and lower density bounds in each individual $\frac{w}{r} \times \frac{w}{r}$ tile of our fixed dissection (Theorem 7), and then we derive upper/lower bounds in the case when we enforce density bounds for standard $w \times w$ windows (Theorem 8). For example, if we enforce the area density to be at least 25% (i.e. $L_a = 0.25$), then (for $r = 5$) the first rule guarantees 16% area density while the standard method can guarantee only 6%. The bounds from Theorems 7 and 8 can help to choose appropriate combinations of fixed dissections and design rules corresponding to specified area density lower/upper bounds.

Theorem 7 *Suppose all $\frac{w}{r} \times \frac{w}{r}$ fixed dissection tiles with top-left corners at points $(i \cdot \frac{w}{r}, j \cdot \frac{w}{r})$, $i, j = 0, 1, ..., r(\frac{n}{w} - 1)$, have area density at least L_a and at most U_a. Then the exact lower bound on the area density of $w \times w$ windows equals*

$$\frac{(r-1)^2}{r^2} \cdot L_a + \frac{4(r-1)}{r^2} \max\{L_a - 0.5, 0\} + \frac{4}{r^2} \max\{L_a - 0.75, 0\}$$

and the exact upper bound equals

$$\frac{(r-1)^2}{r^2} \cdot U_a - \frac{4(r-1)}{r^2} \max\{U_a - 0.5, 0\} - \frac{4}{r^2} \max\{U_a - 0.25, 0\}.$$

□

Theorem 8 *Suppose all $w \times w$-sized windows with top-left corners at points $(i \cdot \frac{w}{r}, j \cdot \frac{w}{r})$, for $i, j = 0, 1, ..., r(\frac{n}{w} - 1)$, have area density at least L_a and at most U_a. Then any $w \times w$ window has density at least $L_a - \frac{1}{r} + \frac{1}{4r^2}$ and at most $U_a + \frac{1}{r} - \frac{1}{4r^2}$, and these bounds are tight.* □

4 Synthesis of Filling and Slotting Patterns

Given the layout geometry along with the parameters of the Filling and Slotting Problem, we wish to synthesize fill and slot geometries such that all windows satisfy the density bounds. In this section we first construct filling patterns for wiring-type layouts that are usually produced by preferred-direction area routers. Then we consider slotting patterns of minimum area. Finally, we derive conditions when both area and perimeter density bounds can be satisfied, and we suggest appropriate filling patterns for such situations.

4.1 Fill Synthesis for Wiring-Type Layouts

Here we present an efficient metal fill synthesis algorithm that will handle layouts containing mostly wires occupying discrete rows, where wire segments have discrete widths and varying lengths. Gridded preferred-direction area routers typically produce such geometries.

If the separation between adjacent wire rows for this type of layout is nearly the same as the width of the rows (rectangles), the layout density never exceeds 50% or 60% anywhere. Thus, typical density upper bounds trivially hold (i.e., are never violated due to the minimum spacing rules for interconnect). To solve the Filling and Slotting Problem for this kind of layout, we only need to make sure that the density lower bound is satisfied everywhere. An $O(k \log k)$ algorithm can achieve this as follows:

1. Sort the wires/rectangles by rows, and within each row sort them by the coordinates of their leftmost starting points.
2. For each row, from left to right, create metal fill in the space between the rectangles (with small separation from the neighboring rectangles on the left and right).

Figure 4 shows an example of a wiring-type layout along with the metal fill solution produced by the above algorithm. Many current designs contain regions with wiring geometries of this form.

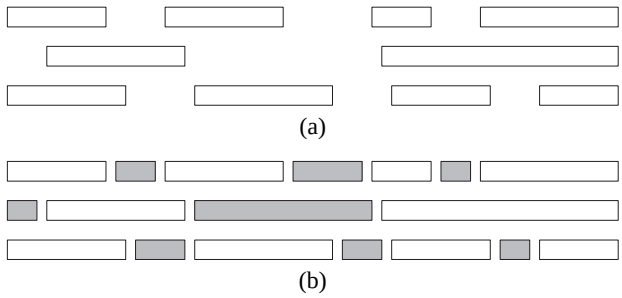


Figure 4: (a) An example of a wiring-type layout, and (b) a corresponding metal fill solution (shaded rectangles denote metal fill).

4.2 Minimization of Slotting

To minimize the slotting of rectangles, we propose the following algorithm that, whenever possible, favors adding metal fill to empty regions rather than slotting existing rectangles. The main idea of this approach is as follows.

1. Inside every rectangle, if there is enough room inside, slot the rectangle lengthwise using parallel slots of width w_1, spaced a distance of d_1 apart. The parameters w_1 and d_1 are chosen so that the density inside the rectangle does not exceed the maximum allowable density (see Figure 5(b)).
2. Outside every rectangle, if there is enough room (with respect to neighboring rectangles), create a maximum-density metal fill band of width w_2 at distance d_2 away from the rectangle, leaving empty space between the rectangle and this band (see Figure 5(c).
3. Fill up the remaining empty areas of the layout (outside all the outer bands) with a canonical slotted metal pattern corresponding to the density lower bound (see Figure 5(d)).

This algorithm clearly satisfies the density upper and lower bounds for appropriate values of d_1, w_1, d_2 and w_2 which depend on w, c and the density upper and lower bounds.[10] These values can be computed in constant time, and the overall algorithm can be implemented to run efficiently.

4.3 Simultaneous Area and Perimeter Bounds

In this subsection, we characterize combinations of area and perimeter densities (D_a, D_p) that can be simultaneously satisfied by the same filling pattern.

As discussed in Section 1, all geometries must satisfy minimum length and minimum separation rules. In particular, no fill feature dimensions, nor any distance between features, can be less than c. In

[10] Recall from Section 1 that slotting requires several design flow changes, particularly since slotted power buses will have reduced current carrying capability. The slotting orientation is aligned with the direction of current flow.

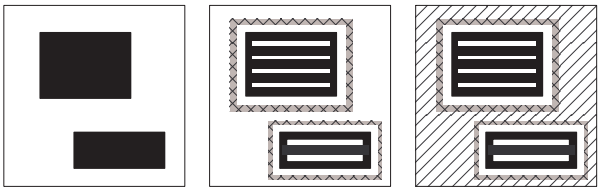


Figure 5: For each rectangle of the given layout (a), we create parallel slots in the direction of the current flow, and a corresponding maximum-density band just outside the rectangle (b). All remaining empty areas between rectangles are filled with a canonical metal pattern having minimum density (c).

practice, the distance between filling or slotting geometries and nearest layout feature is constrained to be greater than $c' > c$. However we can still view regions eligible for filling as *c-polyominoes*, i.e., polyominoes [5] with sides a multiple of c that are in distance c' from the layout features. The fill pattern should also consist of polyominoes in the c-grid, i.e., the minimum separation rule implies that a pair of filled cells which share exactly one corner should have one common filled neighboring cell.

First, we will describe filling patterns for a rectangular region R which have maximum perimeter, and either the minimum or maximum allowable area density. The pattern P_{min} with the minimum area density fills all cells which have top-left corner coordinates $(a+2ci, b+2cj)$, where (a,b) is one of the corners of R (see Figure 6(a)). This pattern has area slightly more than $\frac{1}{4} \cdot area(R)$, because it fills approximately every fourth cell of R. The pattern P_{max} with maximum area density fills R completely, leaving empty only cells with coordinates $(a+c+2ci, b+c+2cj)$ (see Figure 6(b)). The area of this pattern is slightly larger than $\frac{3}{4} \cdot area(R)$ because it leaves empty approximately every fourth cell of R.

Two more patterns are necessary for completing the description of all possible patterns. These are simply the empty pattern P_0 with zero perimeter and area, and the completely-filled pattern P_1 having both perimeter and area equal to those of R. In the graph of Figure 7, the x-axis represents area and the y-axis represents perimeter. The highlighted region with vertices P_0, P_{min}, P_{max} and P_1 represents the combinations of area and perimeter densities for which there exist filling patterns. Notice that a square has the minimum perimeter with a given area. Let S be the area of a maximum square which can be embedded in R. Before the pattern area reaches S, the minimum perimeter grows quadratically; past S, the minimum perimeter grows linearly.

The algorithm for finding a pattern with a given area and perimeter is straightforward: it starts with the minimum area pattern that has the given perimeter, and sequentially adds square cells with side c until the necessary area is achieved.

5 Computational Experience

We now report our computational experience for (i) the fast expected time algorithm of Section 2.4 (ALG3), and (ii) a simple implementation of the approximate overlapping fixed-dissections approach of Section 3 (FD), with $r = 1, 2, 5, 10$.[11] Our benchmarks include CIF-formatted (converted from GDSII Stream) M2 geometries from three

[11] Given a fixed dissection into $(n/w)^2$ windows of size $w \times w$, we iterate over each layout rectangle, and add the rectangle's area contribution to the total of each window that the rectangle intersects. We then check all windows to find the window with maximum area density. We repeat this process r^2 times.

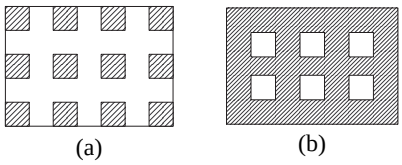


Figure 6: Two patterns with maximum perimeter. (a) the pattern P_{min} with minimum possible area, and (b) the pattern P_{max} with maximum area.

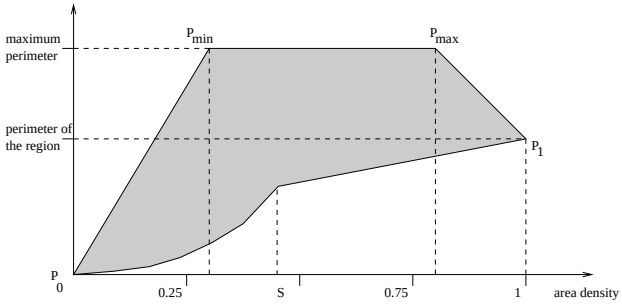


Figure 7: The x-axis represents the area and the y-axis represents the perimeter of the filling pattern. The highlighted region with vertices P_0, P_{min}, P_{max}, and P_1 represents the combinations of area and perimeter for which there exist filling patterns.

standard-cell layouts produced by an industry place-and-route tool.[12] We also use random instances of $k = 2000, 4000, 8000, 16000$ rectangles in square layout regions of side $n = 2000, 4000$, with window sizes $w = 20, 40, 80$.[13]

Table 1 shows runtimes and maximum computed area density for each algorithm. The first column of the Table represents the triple $k/N/w/Type$, where "Type" denotes the two regimes of "wiring-type" (W) and "random-small" (R) (see Footnote 13). The second column gives the runtime (in CPU seconds on a 167MHz Sun Ultra-1) and maximum window density for the algorithm of Section 2.4. The third through sixth columns give (CPU and density values) obtained by the overlapping fixed-dissections approach using $r = 1, 2, 5$, and 10, respectively. We observe that, as expected, the fixed-dissection approach is faster but less accurate, and that its accuracy improves steadily (at the cost of additional CPU time) as r increases.

6 Conclusions and Ongoing Research Directions

In conclusion, we have introduced a critical new problem in the interface between lithography, physical layout design and performance verification. We have given the first formulation of the *filling and slotting problems* that arise in layout post-processing and layout optimization

[12] Benchmark 1 corresponds to a 1756-cell design and has 4470 rectangles; Benchmark 2 corresponds to a 8131-cell design and has 47904 rectangles; and Benchmark 3 corresponds to a 20577-cell design and has 127760 rectangles. For these three benchmarks, we have $n = 34134, 61888, 111905$ and $w = 2000, 2000, 4000$ for window size.

[13] Until k rectangles have been generated, we repeatedly generate a new rectangle having width uniformly random in $[w_{min}, w_{max}]$ and height uniformly random in $[h_{min}, h_{max}]$, such that the rectangle fits inside the layout region and is at least distance c from all previously generated rectangles. There are two regimes: "wiring-type" (W) uses $w_{min} = 1$, $w_{max} = 1000$, and $h_{min} = h_{max} = 1$; "random-small" (R) uses $w_{min} = h_{min} = 1$, $w_{max} = h_{max} = 10$.

for manufacturability. We have also developed a number of effective algorithms for density analysis (both in the general case and in a practical context) as well as for filling/slotting synthesis. Our algorithms have been integrated into a software environment that includes GDSII reader/writer, CIF manipulation, and a geometric database; preliminary data are encouraging, but also point out the need for careful implementation. We are currently seeking more test cases and density rules from industry to further refine our approaches and implementations.[14] We believe that our formulations capture several requirements in future lithography and provide a key unification between lithography and physical design. Our current work addresses such issues as the following:

- developing more efficient, general and provable filling/slotting algorithms (e.g., for simultaneous perimeter- and area-density based criteria);
- finding min/max density/perimeter windows in worst-case time $o(n^2)$ or $o(k^2)$; and
- maintaining knowledge of min/max density/perimeter windows under dynamic rectangle insertion/deletion in time $o(n)$ or $o(k)$.

7 Acknowledgments

We thank Tom Laidig and Kurt Wampler of MicroUnity Systems Engineering, Inc. for many illuminating discussions and patient readings of our drafts. Juan Rey of Cadence has also been generous with his time. We gratefully acknowledge a software donation from Artwork Conversions, Inc.

REFERENCES

[1] R. Bek, C. C. Lin and J. H. Liu, *personal communication*, December 1997.

[2] Cadence Design Systems, Inc. *Dracula Standalone Verification Reference*, November 1997.

[3] L. E. Camilletti, "Implementation of CMP-based Design Rules and Patterning Practices", *1995 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 2-4.

[4] W. B. Glendinning and J. N. Helbert, *Handbook of VLSI Microlithography: Principles, Technology, and Applications*, Noyes Publications, 1991.

[5] S. W. Golomb. *Polyominoes.* Scribner, New York, 1965.

[6] M. Hanan, "On Steiner's Problem with Rectilinear Distance", *SIAM J. Applied Math.* 14 (1966), pp. 255-265.

[7] A. B. Kahng, G. Robins, A. Singh, H. Wang, and A. Zelikovsky, "Filling and slotting: Analysis and algorithms", Technical Report CS-97-30, Department of Computer Science, University of Virginia, December 1997.

[8] H. Landis, P. Burke, W. Cote, W. Hill, C. Hoffman, C. Kaanta, C. Koburger, W. Lange, M. Leach and S. Luce, "Integration of Chemical-Mechanical Polishing into CMOS Integrated Circuit Manufacturing", *Thin Solid Films*, 220(1992), pp. 1-7.

[9] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[10] R. Radojcic, *personal communication*, March 1996.

[11] P. Rai-Choudhury, ed., *Handbook of Microlithography, Micromachining, and Microfabrication, vol. 1: Microlithography*, Bellingham, SPIE Optical Engineering Press, 1997.

[14] Interesting test cases for filling and slotting will not simply be place-and-route test cases: the vast majority of P&R instances are for cell-based implementation of random (control or glue) logic. The majority of the chip – embedded memory cores, high-performance datapaths, global clock and power distribution, analog or mixed-signal blocks, etc. – is what makes the filling and slotting problem challenging, but at the same time such layouts are not typically seen by a place-and-route tool. As a result, test cases for the problem that we address are currently quite difficult to obtain.

Test $k/n/w/Type$	Runtime / Density									
	ALG3		FD(1)		FD(2)		FD(5)		FD(10)	
	CPU	den	CPU	den	CPU	den	CPU	den	CPU	den
1k/2k/20/R	9.5	.360	0.1	.208	0.1	.230	0.5	.315	2.1	.360
1k/2k/40/R	3.5	.113	0.0	.090	0.1	.090	0.4	.106	1.4	.109
1k/2k/80/R	2.7	.045	0.0	.031	0.1	.033	0.4	.038	1.3	.038
1k/4k/20/R	35.4	.340	0.1	.203	0.2	.203	1.0	.203	3.8	.203
1k/4k/40/R	10.1	.129	0.0	.051	0.2	.058	0.5	.090	1.9	.090
1k/4k/80/R	3.5	.037	0.1	.023	0.1	.023	0.4	.023	1.4	.029
4k/2k/20/R	29.7	.403	0.2	.298	0.4	.298	1.7	.388	5.9	.388
4k/2k/40/R	15.4	.169	0.2	.127	0.4	.145	1.4	.160	5.3	.161
4k/2k/80/R	26.9	.078	0.2	.067	0.4	.067	1.3	.069	4.9	.069
4k/4k/20/R	102.3	.362	0.2	.203	0.5	.210	2.1	.225	7.9	.225
4k/4k/40/R	29.7	.133	0.2	.096	0.4	.096	1.5	.097	5.6	.097
4k/4k/80/R	15.2	.052	0.2	.037	0.4	.037	1.4	.041	5.0	.041
16k/2k/20/R	130.8	.518	1.1	.313	1.5	.368	5.9	.403	22.7	.428
16k/2k/40/R	128.6	.266	1.1	.188	1.4	.234	5.7	.249	20.5	.249
16k/2k/80/R	368.2	.172	0.9	.139	1.4	.139	5.3	.144	19.9	.150
16k/4k/20/R	368.1	.380	0.9	.360	1.6	.360	6.5	.360	23.8	.360
16k/4k/40/R	125.9	.183	0.9	.136	1.5	.136	5.6	.136	20.6	.166
16k/4k/80/R	123.7	.096	0.0	.083	1.4	.083	5.4	.085	19.5	.085
1k/2k/20/W	13.7	.350	0.1	.300	0.3	.300	2.0	.350	7.6	.350
1k/2k/40/W	6.4	.324	0.1	.250	0.2	.250	1.0	.250	4.2	.250
1k/2k/80/W	5.7	.242	0.1	.210	0.1	.217	0.6	.217	4.0	.230
1k/4k/20/W	39.9	.250	0.2	.200	0.4	.200	2.5	.200	10.0	.200
1k/4k/40/W	11.9	.175	0.1	.100	0.2	.125	1.2	.150	5.1	.150
1k/4k/80/W	5.0	.138	0.0	.077	0.1	.092	0.7	.092	2.8	.092
4k/2k/20/W	44.0	.500	0.4	.450	0.9	.450	4.8	.450	19.2	.450
4k/2k/40/W	30.7	.474	0.3	.389	0.6	.392	3.0	.396	11.7	.413
4k/2k/80/W	46.1	.410	0.2	.343	0.5	.370	2.2	.373	8.1	.383
4k/4k/20/W	120.1	.400	0.5	.350	1.4	.350	7.8	.350	30.5	.350
4k/4k/40/W	42.4	.325	0.3	.225	0.8	.295	4.3	.322	16.6	.322
4k/4k/80/W	27.9	.259	0.3	.185	0.6	.204	2.7	.211	10.4	.213
16k/2k/20/W	167.9	.500	1.1	.500	2.3	.500	10.5	.500	40.0	.500
16k/2k/40/W	201.7	.500	1.0	.444	1.8	.445	7.8	.469	29.0	.474
16k/2k/80/W	475.8	.463	1.0	.412	1.7	.422	6.9	.435	23.6	.437
16k/4k/20/W	368.1	.500	1.5	.480	3.8	.480	20.4	.500	78.9	.500
16k/4k/40/W	125.7	.484	1.2	.401	2.6	.402	12.5	.404	48.2	.419
16k/4k/80/W	124.3	.410	1.1	.332	2.1	.336	8.8	.349	33.0	.370
Benchmark 1	97	.295	0.0	.266	0.0	.274	0.4	.286	1.4	.289
Benchmark 2	728	.255	0.4	.228	0.9	.228	4.3	.230	16.1	.244
Benchmark 3	3117	.863	1.2	.508	2.7	.559	11.5	.567	39.9	.580

Table 1: CPU times and maximum computed window densities for ALG3 (Section 2.4) and the overlapping fixed-dissections approximate algorithm (FD) with $r = 1, 2, 5, 10$. FD results are normalized to those of ALG3. W = "wiring-type"; R = "random-small".

[12] "SEMATECH Demonstrates New Insulator For Faster Chips", *Business Wire*, Nov. 11, 1997.

[13] Semiconductor Industry Association, *The National Technology Roadmap for Semiconductors*, December 1997.

[14] K. Wampler and T. Laidig, *personal communication*, Sept. 1997.

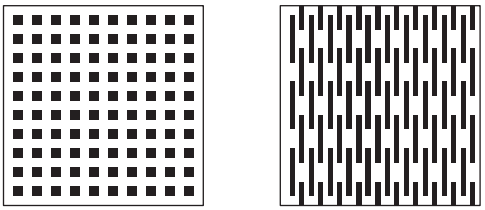
Appendix: Fill Impact on Extraction

Table 2 shows capacitance extraction results obtained with the Raphael 3-D field solver from TMA/Avant!, for an isolated conductor (i) with or without fill insertion in empty regions of adjacent layers, and (ii) with or without same-layer neighbor conductors. The simulation shows that ignoring the possibility of metal fill can result in underestimation of total line capacitance by more than 50%. This can in turn lead to useless RCX, delay calculation, and timing analysis results. We conclude that the presence or absence of fill geometries *must* be modeled during performance-driven layout optimization. Similarly, we can show that slotting must also be modeled for power and reliability analyses. Such modeling must be efficient and "transparent"; since there are many iterations through the layout optimization loop, we must be careful with the time complexity of fill/slot insertion and the increases in data volume.

Tables 3 and 4 give TMA/Avant! Raphael capacitance extraction results for multi-layer interconnect structures involving fill geometries, as follows.

Victim Layer Total Capacitance (10^{-15}F)			
Same layer-i neighbors?	Fill layers $i-1$, $i+1$?	$\varepsilon = 3.9$	$\varepsilon = 2.7$
N	N	2.43(1.00)	1.68(1.00)
N	Y	3.73(1.54)	2.58(1.54)
Y	N	4.47(1.84)	3.09(1.84)
Y	Y	5.29(2.18)	3.66(2.18)

Table 2: Raphael 3-D field solver results for total capacitance extraction of a single victim conductor. The conductor on layer i is 20×1. Line-to-line spacing is 1, line width is 1, line thickness is 1.5, and dielectric height is 1.5. Metal fill features on layers $i-1$ and $i+1$ are 10×1 with side-to-side spacing of 1 and end-to-end spacing of 4. The dielectric permittivity was set to both 3.9 (for SiO_2) and 2.7 (cf. recent announcements by Sematech [12] of new low-permittivity dielectric technologies). Layers $i-2$ and $i+2$ are set to be 40×40 ground planes.



(a) (b)

Figure 8: The two fill patterns considered in Raphael simulations: 1×1 squares separated 1 unit apart (a), and 10×1 rectangles separated 1 unit apart horizontally and 4 units apart vertically (b). The fill pattern (b) was used for the simulations reported in Table 2.

Victim B Total Capacitance (10^{-15}F)			
Fill layer offset	Fill geometry	$\varepsilon = 3.9$	$\varepsilon = 2.7$
N	10×1	3.776(1.00)	2.614(1.00)
N	1×1	3.750(0.99)	2.596(0.99)
Y	10×1	3.777(1.00)	2.615(1.00)
Y	1×1	3.745(0.99)	2.593(0.99)

Table 3: TMA/Avant! Raphael capacitance extraction results: total capacitance for the middle victim conductor B.

- Three 20×1 victim conductors A, B and C (with B in the middle), with spacing 1 between them, are placed on a victim layer i. All conductor thicknesses = 1.5; dielectric height between layers = 1.5. Dielectric permittivity was set at either 3.9 or 2.7.
- A 40×40 bottom ground plane is placed at layer $i-2$.
- Two types of fill geometry patterns were considered for layer $i-1$ (see Figure 8): (a) 1×1 squares with (x,y) origins of form $(2i,2j)$, i and j integers, resulting in an overall pattern area density (for an infinite layout region) of 0.25, and (b) 10×1 (tall and thin) rectangles with (x,y) origins of form $(4i,14j)$ or $(4i-2,14j-7)$, i and j integers, resulting in an overall pattern area density (for an infinite layout region) of 0.357.
- An *offset* is optionally introduced. When the fill geometries are offset, they lie directly under the spaces between the victim conductors. When there is no offset, the fill geometries lie directly under the victim conductors.

Table 3 shows that the total capacitance values for the middle conductor (B) fluctuate by less than 1 percent over all four combinations of fill pattern and offset. The critical factor is that the fill is present in the first place. Similarly, Table 4 shows that the total capacitance values for each of the outside conductors (A and C) also fluctuate by less than one percent. We conclude that the filling and slotting can, subject to constraints involving feature dependencies between layers, be viewed as a "single-layer problem".

Victim A, C Total Capacitance (10^{-15}F)			
Fill layer offset	Fill geometry	$\varepsilon = 3.9$	$\varepsilon = 2.7$
N	10×1	3.009(1.00)	2.083(1.00)
N	1×1	2.984(0.99)	2.066(0.99)
Y	10×1	3.004(1.00)	2.080(1.00)
Y	1×1	2.980(0.99)	2.063(0.99)

Table 4: TMA/Avant! Raphael capacitance extraction results: total capacitance for the outside victim conductor A or C.