Filling Algorithms and Analyses for Layout Density Control*

Andrew B. Kahng, Gabriel Robins†, Anish Singh† 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,alexz}@cs.ucla.edu, {robins,as6f}@cs.virginia.edu

Abstract

In very deep-submicron VLSI, manufacturing steps involving chemical-mechanical polishing (CMP) have varying effects on device and interconnect features, depending on local characteristics of the layout. To reduce manufacturing variation due to CMP and to improve performance predictability and yield, layout must be made uniform with respect to certain *density* criteria, by inserting "fill" geometries into the layout. To date, only foundries and special mask data processing tools perform layout post-processing for density control. In the future, better convergence of performance verification flows will depend on such layout manipulations being embedded within the layout synthesis (place-and-route) flow. In this paper, we give the first realistic formulation of the *filling* problem that arises in layout optimization for manufacturability. Our formulation seeks to add features to a given process layer, such that (i) feature area densities satisfy prescribed upper and lower bounds in all *windows* of given size, and (ii) the maximum variation of such densities over all possible window positions in the layout is minimized. We present efficient algorithms for density analysis, notably a multilevel approach that affords user-tunable accuracy. We also develop exact solutions to the problem of fill synthesis, based on a linear programming approach. These include an LP formulation for the *fixed-dissection* regime (where density bounds are imposed on a predetermined set of in the layout) and an LP formulation that is automatically generated by our multilevel density analysis. We briefly review criteria for fill pattern synthesis, and the paper then concludes with computational results and directions for future research.

Keywords: Layout verification, manufacturability, chemical-mechanical polishing (CMP), density control, metal fill, physical design, yield enhancement

1 Introduction

As CMOS technology advances according to the Semiconductor Industry Association National Technology Roadmap for Semiconductors [24] and moves into the 180nm generation and beyond, foundry amortization becomes a dominant business concern, and manufacturing cost increasingly drives design [17]. To maximize yield, process engineers must achieve *predictability* and *uniformity* of manufactured device and interconnect attributes, e.g., dopant concentrations, channel lengths, interconnect dimensions, contact shapes and parasitics, and interlayer dielectric thicknesses. A total *variability budget* for the design is distributed among such attributes. In very deep submicron technologies, large process windows and uniform manufacturing is difficult [5] [10] [22] [15] [17] [7], and the manufacturing process has an increasingly constraining effect on physical layout design and verification. Many physical design methods have been proposed to address various manufacturing issues such as registration errors, photolithographic random effects, etc.; see such works as [16] [7] for reviews.

*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. This paper has appeared in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 4, April 1999, pp. 445-462.

In this paper, we address the problem of controlling the manufacturing variation that is due to *chemical-mechanical polishing* (CMP) [15] [19] [29]. CMP is the procedure by which wafers are polished using a rotating pad and slurry to achieve the planarized surfaces on which succeeding processing steps can build. The key observations are:

- The polishing environment involves large pad downforce[1] and a significant variability due to pad wear. Hence, control of polish depth (i.e., final thickness of the layer being polished) is extremely difficult.
- The elasticity of the polishing pad compounds the variability problem. Notably, in oxide polishing of interlayer dielectrics (oxide CMP), the pad conforms to local topography and overpolishes empty oxide areas that have no underlying metal features (a phenomenon called *dishing*); on the other hand, areas with dense underlying metal features are underpolished.
- A large fraction of the die's variability budget is used up by the oxide thickness variation [8] [26]. Interlayer dielectric thickness variation of 4000 angstroms is common, and this can severely affect estimates of electrical performance [13] [26] [27].
- The problem of CMP variation is rapidly worsening today, as industry moves to shallow-trench isolation (STI) sub-0.25μ processes, where CMP is used to planarize glass [6] [18] [25]. For such processes, as well as for new inlaid-metal (e.g., damascene copper) processes [4], CMP variation must be even more tightly controlled.

Recent work in the field of statistical metrology shows that fundamentally, CMP variation is controlled if the **local** *feature density* is controlled [20] [28]. Figure 1 illustrates the local dependency of oxide thickness on feature density, which is roughly monotone. By reducing the variation of local feature density over the die, the variation of oxide thickness can be reduced.

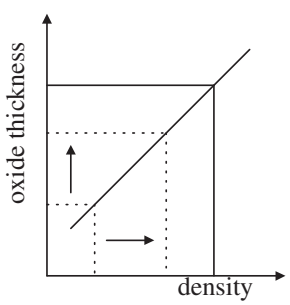


Figure 1: Relationship between oxide thickness and local feature density.

The definition of "local" is determined by the *length scale* at which feature density impacts oxide thickness, and corresponds to the "window size" within which feature density must be controlled. For oxide CMP, this length scale has been estimated to be on the order of 1-3mm, depending on CMP pad material, slurry composition, etc. [9] [20] [28].

To minimize the impact of CMP variation on device yield, foundries have imposed *density* rules for features on active and metal layers, typically starting with mature 0.35μm process generations.

[1]Typical polish downforces in oxide CMP range from 4 to 10 psi, depending on slurry / oxidizer concentration and process considerations. For 200mm substrates, this results in a total wafer downforce of up to 500 pounds [4].

The purpose of the density rules is, of course, to make the layout more uniform. Many process layers, including diffusion and thin-ox, can have associated density rules. As examples, in $0.35\mu m$ and below, one major foundry requires overall feature area density on diffusion layers to be between 0.25 and 0.40, and overall density on any metal layer to be between 0.40 and 0.70; another major foundry requires overall metal layer area density to be at least 0.35. Density rules and layout post-processing approaches may differ in various contexts (e.g., ASIC vs. high-end microprocessor design) due to tradeoffs between device performance and predictability [1] [30].

To satisfy density rules, a post-processing step adds *fill* geometries into the original layout. Traditionally, only foundries or specialized mask data processing tools have performed the post-processing of layout needed to achieve this uniformity. Today, with more customer-owned tool flows, and with the need for early and accurate performance verification, physical verification tools at the back end of the IC design flow are becoming aware of density-driven layout rules.[2]

We observe that the state of the art in density control for CMP leaves much to be desired.

- Many foundry density rules still constrain only the average *overall* feature density on a given layer; the issue of local variation in feature density is ignored.
- Current approaches to analysis of layout density do not actually find the true extremal window densities in the layout. Rather, they find the extremal window densities over a fixed set of window positions using the "fixed r-dissection" approach that we discuss below. This can result in substantial error.
- Current methods for inserting fill geometries into the layout do not actually minimize the maximum variation in layout density between windows of the layout. Rather, simple Boolean layer processing techniques are applied to insert fill patterns into any empty region that is sufficiently large.

These weaknesses can perhaps be attributed to the genealogy of today's software tools for density control. Such tools are typically evolved from physical verification tools and mask processing tools, where the mindset is chiefly concerned with verification rather than data modification, with local rules rather than global rules, and with Boolean rules rather than context-dependent rules. By contrast, our work addresses all of the above weaknesses: (i) our formulation of the filling problem seeks to minimize density variation over all possible windows, (ii) we develop a multilevel density analysis approach that is more accurate and faster than the "fixed-dissection" approach, and (iii) we develop a linear programming approach that considers and optimizes globally the amounts of fill to be added into each region of the layout.

Notation and Problem Formulation

The following notation and definitions are used.

- The input is a *layout* consisting of rectangular *geometries*, with all sides having length a multiple of c (the minimum feature width or spacing).[3] The value of c is typically 25 to 50 times the manufacturing unit.

[2] Note that without an accurate estimate of the filling that will be added later at the foundry, all the RC extraction, delay calculation, timing, noise and reliability analyses done during physical design performance verification can be highly suspect. The Appendix presents analyses showing the extent to which metal filling can affect the results of capacitance extraction and performance analysis. A broken design flow can result if the effects of filling are not properly modeled in earlier design stages.

[3] Without loss of generality, we will assume that rectilinear geometries have been fractured into, say, horizontally maximal rectangles. It is also possible to generalize these analyses and algorithms from rectangles to trapezoids. Standard industry tools, such as Cadence Dracula, will fracture geometries into horizontal trapezoids [3].

- $n \equiv$ side of the layout region. If the layout region is the entire die, n might typically be 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.
- $w \equiv$ fixed *window* size. The window is the moving square area over which the layout density rules apply. A typical window size would be $w = 10,000 \cdot c$.
- $k \equiv$ the complexity of the original layout, i.e., the total number of rectangles in the input.
- $U \equiv$ *area (or perimeter) density upper bound*[4], expressed as a real number $0 < U < 1$. Each $w \times w$ region of the layout must contain total area of features $\leq U \cdot w^2$.
- $L \equiv$ *area (or perimeter) density lower bound*, expressed as a real number $0 < L < U < 1$. Each $w \times w$ region of the layout must contain total area of features $\geq L \cdot w^2$.
- $B \equiv$ *buffer distance.* Fill geometries cannot be introduced within distance B of any layout feature.
- $slack(W) \equiv$ *slack* of a given $w \times w$ window W, i.e. the maximum amount of fill area that can be introduced into W.[5]
- 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 parameters above, we define the Filling Problem as follows:[6]

> **The Filling Problem.** Given a design rule-correct *layout* geometry of k disjoint rectilinear rectangles in an $n \times n$ layout region, minimum feature size c, window size $w < n$, buffer distance B, and area (or perimeter) density lower bound L and upper bound U, add fill geometries to create a *filled layout* that satisfies the following conditions:
>
> (1) circuit function and design rule-correctness are preserved;
>
> (2) no fill geometry is within distance B of any layout feature;
>
> (3) no fill is added into any window that has density $\geq U$ in the original layout;
>
> (4) for any window that has density $< U$ in the original layout, the filled layout density is $\geq L$ and $\leq U$; and
>
> (5) the minimum window density in the filled layout is maximized.

Condition (5) corresponds to what we call the **Min-Variation Formulation**, since it minimizes the difference between minimum and maximum window density in the filled layout. Condition (3) implies that, without loss of generality, no window in the original layout has density $> U$ (otherwise, such a window would have its contents fixed, so that it could not be changed by the filling process).

[4] It turns out that most of the results and algorithms of this paper easily apply to either the area density or perimeter density regimes. Thus, we will generically indicate both the area and perimeter density upper bounds with U, and lower bounds with L. In practice, the maximum density is attained in memory cores, and the bound U is set with respect to this maximum density. To our understanding, no foundry yet imposes *both* area density and perimeter density bounds *simultaneously* on a given layer [30]. However, such simultaneous constraints may be required in the future (e.g., for reverse-active area masks), and we analyze fill pattern synthesis for such a situation in Section 4.

[5] The value of $slack(W)$ will depend on the maximum possible fill pattern density. That is, total empty area outside the buffer distance B from any feature should be scaled by the maximum possible fill density to yield the slack of the window.

[6] Note that this is not merely a *satisficing* formulation where we seek only a *feasible* solution, but rather an *optimization* formulation where we seek a best solution, as dictated by the particular underlying VLSI technology.

Organization

Our paper is organized to reflect three major functions in density control for CMP: (1) window density analysis, (2) determining the optimal fill amount to be inserted in various regions of the layout, and (3) actual insertion of the appropriate fill pattern.

Section 2 describes several ways of finding extremal window density within the layout. We first describe the *fixed-dissection* approach, where a dissection partitions the layout into disjoint windows, and windows of only a finite number of (overlapping) fixed dissections are taken into account. To our understanding, this is the type of analysis afforded by today's physical verification tools. We give a tight analysis of the error inherent in fixed-dissection extremal-density analysis, i.e., it yields only an approximation of the global extremal window density. We then describe and analyze an exact method for finding the extremal global window density. Finally, we develop a *multilevel* density analysis approach with user-tunable accuracy; this method is more accurate, and faster, than the fixed-dissection approach.

Section 3 gives new linear programming based methods for determining the *optimal* fill area to be inserted in the layout. We first address the fixed-dissection regime. We then incorporate results of the multilevel density analysis into a variant LP formulation. Finally, an LP formulation is proposed which minimizes an estimate of global window density variation.

Section 4 describes several approaches to filling pattern synthesis. For example, we note the possibilities of rectangle-based and basket-weave patterns on given pitches, explore the regime where both area and perimeter density bounds must be satisfied, and suggest appropriate filling patterns for such situations. Section 5 describes implementation and computational experience, and Section 6 concludes with directions for future research.

2 Extremal Density Analysis

We first develop algorithms for density analysis (with respect to either area or perimeter). Given a fixed layout and window size, we seek to determine a maximum-density and a minimum-density window (i.e., the algorithms will return extremal-density window(s)).[7] The density analysis problem is stated 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 consider a fixed-dissection approach when windows from several fixed dissections of the layout are taken into account. To our understanding, this approximate method is the type of analysis provided by commercial verification tools. Several algorithms are then proposed for optimal solution (i.e., over all possible windows) of the Extremal-Density Window Analysis problem.

2.1 Fixed-Dissection Density Analyses

In practice, feature density bounds are enforced only within a fixed set of $w \times w$ windows corresponding to a dissection of the layout region into $(\frac{n}{w})^2$ non-overlapping $w \times w$ windows. Since bounding the density in windows of a dissection can incur error (i.e., other windows not in the dissection could violate the density bound), a common practice is to enforce density bounds in r^2 dissections, where r

[7] These density analysis methods can, if desired, report *all* violations of density bounds in the layout within the same time complexity needed to report a single extremal-density window.

determines the "phase shift" w/r by which the dissections are offset from each other. In other words, density bounds are enforced only for windows from the fixed r-dissection defined below.

Definition: A *fixed r-dissection* of the layout is the set of $w \times w$ windows having bottom-left corners at points $(i \cdot \frac{w}{r}, j \cdot \frac{w}{r})$, for $i, j = 0, 1, \ldots, r(\frac{n}{w} - 1)$, where r is an integer divisor of w.

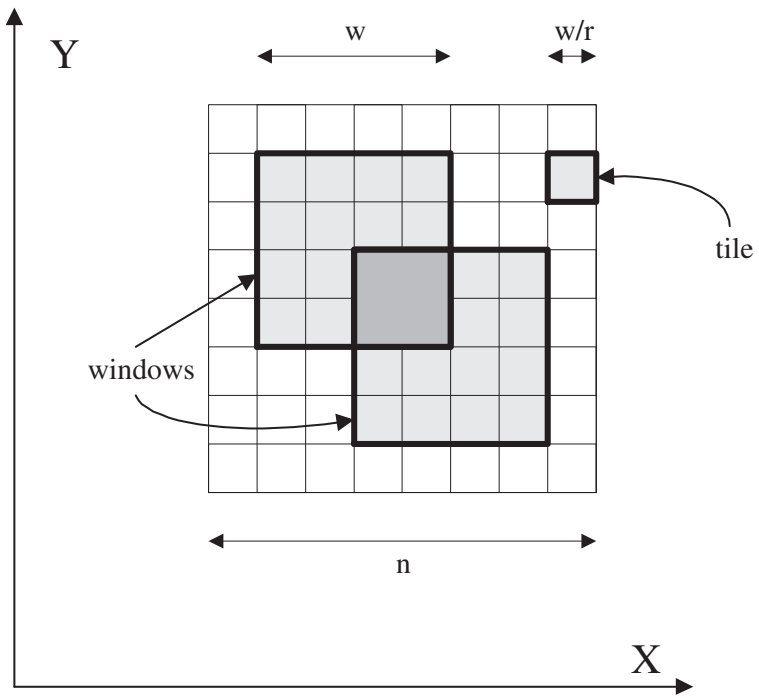


Figure 2: The layout is partitioned by r^2 ($r = 4$) distinct dissections, each dissection having window size $w \times w$, into $\frac{nr}{w} \times \frac{nr}{w}$ *tiles*. Each $w \times w$ window (dark) consists of r^2 tiles. A pair of windows from different dissections may overlap.

A fixed r-dissection divides the layout into $\frac{nr}{w} \times \frac{nr}{w}$ *tiles*, each of size $\frac{w}{r} \times \frac{w}{r}$ (see Figure 2). In other words, each $w \times w$ window in a fixed r-dissection consists of r^2 non-overlapping tiles. For instance, the bottom-left $w \times w$-window corresponds to an $r \times r$ grid of tiles whose origins are at grid node coordinates $(i\frac{w}{r}, j\frac{w}{r})$, $i, j = 0, \ldots, r$. Only, nodes $(i\frac{w}{r}, j\frac{w}{r})$, $i, j = 0, \ldots, r-1$ determine different dissections e.g., the node $(\frac{w}{r}, r\frac{w}{r})$ determines the same dissection as the node $(0, r\frac{w}{r})$ (see Figure 2). In practice, a density upper bound for arbitrarily located windows is sought by enforcing density upper bounds on all windows in a fixed r-dissection. [8]

Unfortunately, it turns out that 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 r-dissection is guaranteed to be at least 75%, a

[8] To the best of our knowledge, commercial tools (Avant! Hercules, Mentor Calibre, Cadence Dracula/Vampire) provide only layout density checking with respect to fixed r-dissections. E.g., the Cadence Dracula COVERAGE command [3] allows checking of feature area density upper and lower bounds in $w \times w$ windows that occur at a fixed offset from each other (e.g., an offset of $100\mu m$ with $w = 500\mu m$ corresponds to $r = 5$).

completely empty $w \times w$ tile can exist. Conversely, if the area density of each window in the fixed r-dissection is guaranteed to be at most 25%, a completely full $w \times w$ window can exist.

On the other hand, we believe that the analysis of fixed r-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 r-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 $r > 1$, the above procedure is repeated r^2 times in order to check all $(r \cdot \frac{n}{w})^2$ windows.

The rest of this subsection seeks ways in which density bounds for arbitrarily located windows can be enforced by density bounds on fixed r-dissection windows. We compare two ways of applying simple local rules to windows having bottom-left corners at points $(i \cdot \frac{w}{r}, j \cdot \frac{w}{r})$, $i, j = 0, 1, \ldots, \frac{n}{w}$ for some $r > 1$ such that $\frac{w}{r}$ is an integer. First, we consider what happens when the upper and lower density bounds are enforced in each individual $\frac{w}{r} \times \frac{w}{r}$ tile of the fixed r-dissection (Theorem 1), and then we derive upper/lower bounds in the case when we enforce density bounds for standard $w \times w$ windows (Theorem 2). For example, if the area density is enforced to be at least 25% (i.e. $L = 0.25$), then (for $r = 5$) the first rule guarantees 16% area density while the standard method can guarantee only 6%. The bounds from Theorems 1 and 2 can help to choose appropriate combinations of fixed r-dissections and design rules corresponding to specified area density lower/upper bounds.

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

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

and the exact upper bound equals

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

Proof: Let the bottom-left corner of a $w \times w$ window W have coordinates (a, b). Then W is covered by $(r+1)^2$ fixed r-dissection tiles of size $\frac{w}{r} \times \frac{w}{r}$ which form a square with diagonal corners $(\lfloor a/\frac{w}{r} \rfloor \frac{w}{r}, \lfloor b/\frac{w}{r} \rfloor \frac{w}{r})$ and $(\lfloor (a+w)/\frac{w}{r} \rfloor \frac{w}{r}, \lfloor (b+w)/\frac{w}{r} \rfloor \frac{w}{r})$. In general, all these $(r+1)^2$ tiles can be classified into three groups: $(r-1)^2$ tiles which are completely covered by the window W; $4(r-1)$ tiles which intersect the boundary of W but do not contain the corners of W; and 4 tiles each containing a corner of W (see Figure 3(a)). Separately compute the contribution of each group to the lower bound on area density of the window W. The first group contributes $(r-1)^2$ tiles with density at least L. If the area density $L > 0.5$, then the second group contributes $4(r-1)$ tiles with density at least $(L - 0.5)$; and if $L > 0.75$, then the third group contributes 4 tiles with density at least $(L - 0.75)$. The total of these contributions yields the claimed lower bound on area density.

We now compute the upper bound on filled area in the window W. Without loss of generality, we assume that each fixed r-dissection tile is U-filled. Clearly, the filled area in W cannot be more than the total filled area in all $(r+1)^2$ tiles; this is at most $\frac{(r+1)^2}{r^2} \cdot U$. We then subtract the filled area not covered by W that is possibly contained in tiles from the second and third groups. If $U > 0.5$, each tile in the second group contains area not covered by W with the density at least $U - 0.5$. If $U > 0.25$, each of the 4 tiles from the third group contains area not covered by W with the area density at least $U - 0.25$. We thus obtain the claimed upper bound on area density.

To prove that the upper and lower bounds are tight we need to present an instance for which these bounds hold. It is easy to check that this happens when the bottom-left corner of W is at the center of a fixed-dissection tile. □

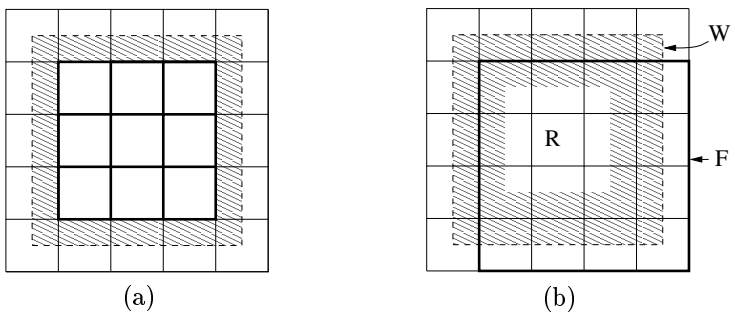


Figure 3: Worst-case analysis of two design rules when density bounds are enforced (a) in all $\frac{w}{r} \times \frac{w}{r}$-sized tiles of a fixed r-dissection, and (b) in all $w \times w$-sized tiles with bottom-left corners at points $(i \cdot \frac{w}{r}, j \cdot \frac{w}{r})$, $i, j = 0, 1, \ldots, \frac{n}{w}$ (this corresponds to r^2 dissections into $w \times w$-sized windows). For the first rule (a), the window W with dashed boundary contains $(r-1)^2$ tiles with thick boundary (the first group) and the highlighted area (the second and third groups) can be completely or partially filled. For the second rule (b), the window W with dashed boundary can contain a square region R (the empty area in the center of W) that overlaps with any fixed r-dissection $w \times w$ window F (square with thick boundary) having largest intersection with W.

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

Proof: Let the bottom-left corner of a $w \times w$ window W have coordinates (a, b). The four fixed r-dissection $w \times w$ windows with the bottom-left corners in the four corners of the $\frac{w}{r} \times \frac{w}{r}$ tile containing (a, b) have the largest overlap with W. At least one of these four windows, say F, and the window W overlap by a square with size at least $(\frac{r-\frac{1}{2}}{r}) \cdot w^2$ (see Figure 3(b)). The upper density constraint implies that the total empty area inside F is at least $(1-U) \cdot w^2$. Therefore, even if the region $F - W$ of the total area $\left(1 - \left(\frac{r-\frac{1}{2}}{r}\right)\right) \cdot w^2$ is empty, the window W must still contain an empty area of size

$$(1-U) \cdot w^2 - \left(1 - \left(\frac{r - \frac{1}{2}}{r}\right)^2\right) \cdot w^2 = \left(-U + \left(1 - \frac{1}{2 \cdot r}\right)^2\right) \cdot w^2$$

Therefore, even if the rest of the window W is completely filled, the total filled area cannot be more than

$$w^2 - \left(-U + \left(1 - \frac{1}{2 \cdot r}\right)^2\right) \cdot w^2 = \left(U + \frac{1}{r} - \frac{1}{4r^2}\right) \cdot w^2$$

The proof of the lower bound is similar.

The worst case occurs when the bottom-left corner of the window W is at the center of a $\frac{w}{r} \times \frac{w}{r}$-tile. Then the empty region R of area $(U + 1 - (1 - \frac{1}{2 \cdot r})^2) \cdot w^2$ can be placed in the center of W, and the rest of W can be filled. This way, all of the four fixed r-dissection windows with the largest overlap with W have the common region R. On the other hand, any fixed r-dissection $w \times w$ window which has smaller intersection with W can have a larger empty part outside of W (see Figure 3(b)). $\square$

2.2 Optimal Extremal-Density Window Analysis

This subsection is devoted to *optimal* extremal density analysis, i.e., covering all possible $w \times w$ windows in the layout region. 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, 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, the 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 $O(min(k^2, n^2))$.

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

A simple algorithm for density analysis has time complexity $O(n^2)$.

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. 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:

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

   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 [21].

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.

2.2.2 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 [12]. 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 4).[9]

Theorem 3 *Given a layout of k rectilinearly-oriented rectangles in the $n \times n$ grid and a fixed window size w, there exists a $w \times w$ maximum-density window having at least one of its corners at a vertex of the Hanan grid.* □

[9] Here and elsewhere in what follows we state the results for maximum-density windows, explaining the extensions to minimum-density windows only if there is a possibility of confusion.

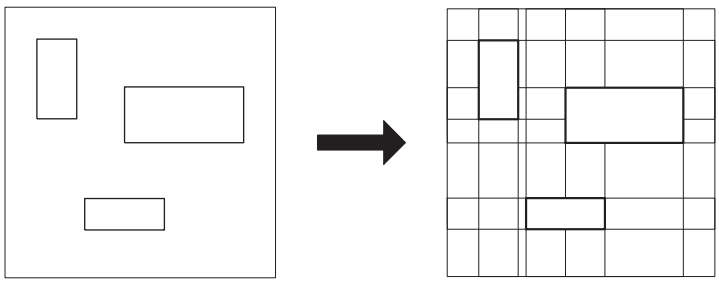


Figure 4: A layout (left) and its corresponding Hanan grid (right).

Proof: If neither the left nor right edge of a maximum-density window W touches the boundary of any of the k rectangles, then we can continuously slide W horizontally either to the left or to the right without decreasing its density, until it touches one of the rectangles on either its left or right side (see Figure 5). Similarly, if neither the top nor the bottom edge of W touches the boundary of any of the rectangles, then W can be slid vertically either up or down until it touches one of the rectangles with either its top or bottom edge, without decreasing W's density.

Note that the same arguments hold even if some of the rectangles intersect W's boundary before the sliding operations commence. Since we assumed that W was a maximum-density window, W must remain a maximum-density window after these two sliding operations have been performed. It follows that there exists a maximum-density window (i.e., W in its new position after the two sliding operations) that abuts one or more rectangles of the layout on two of its adjacent sides. Thus, there exists a maximum-density window with one of its corners coinciding with a Hanan grid point. □

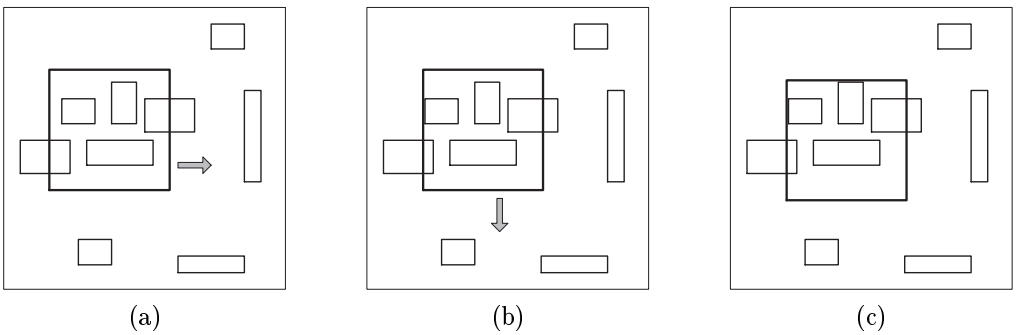


Figure 5: A maximum-density window may be slid horizontally (a) until it touches one of the rectangles; the window may then be slid vertically (b) until it touches one of the rectangles. After the sliding operations have been performed, the window will abut one or more rectangles of the layout on two adjacent sides (c).

Theorem 3 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 feasible locations of a maximum-density window (i.e., as abutting either one or two of the layout rectangles). The argument used to prove Theorem 3 can also be used to establish an analogous result for *minimum*-density windows.

Corollary 4 *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 exists 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 5 *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.* □

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

Recall that Theorem 3 establishes that an extremal-density window must touch rectangles of the layout with at least two of its sides. Since there are only $O(k)$ sides of rectangles, the extremal density analysis can be achieved 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. A careful implementation of this scheme yields an algorithm with overall worst-case time complexity of $O(k^2)$ as follows (see Figure 7).

We preprocess the rectangles 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 (line (2) in Figure 7), 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 6(a)). We then compute the density of W in $O(k)$ time by intersecting W with all k rectangles of the layout (line (3) in Figure 7).

In the inner loop (4), we slide the window W horizontally to the right (Figure 6(a-f)) 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.[10] 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 lines 3 through 5 of algorithm ALG3 (Figure 7) for all other $O(1)$ starting orientations of W with respect to the pivot rectangle R (Figure 6(g-i)). The overall time complexity of this algorithm is dominated by the $O(k)$ scans which require $O(k)$ time each, to a total of $O(k^2)$ time.

2.2.4 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

[10] 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).

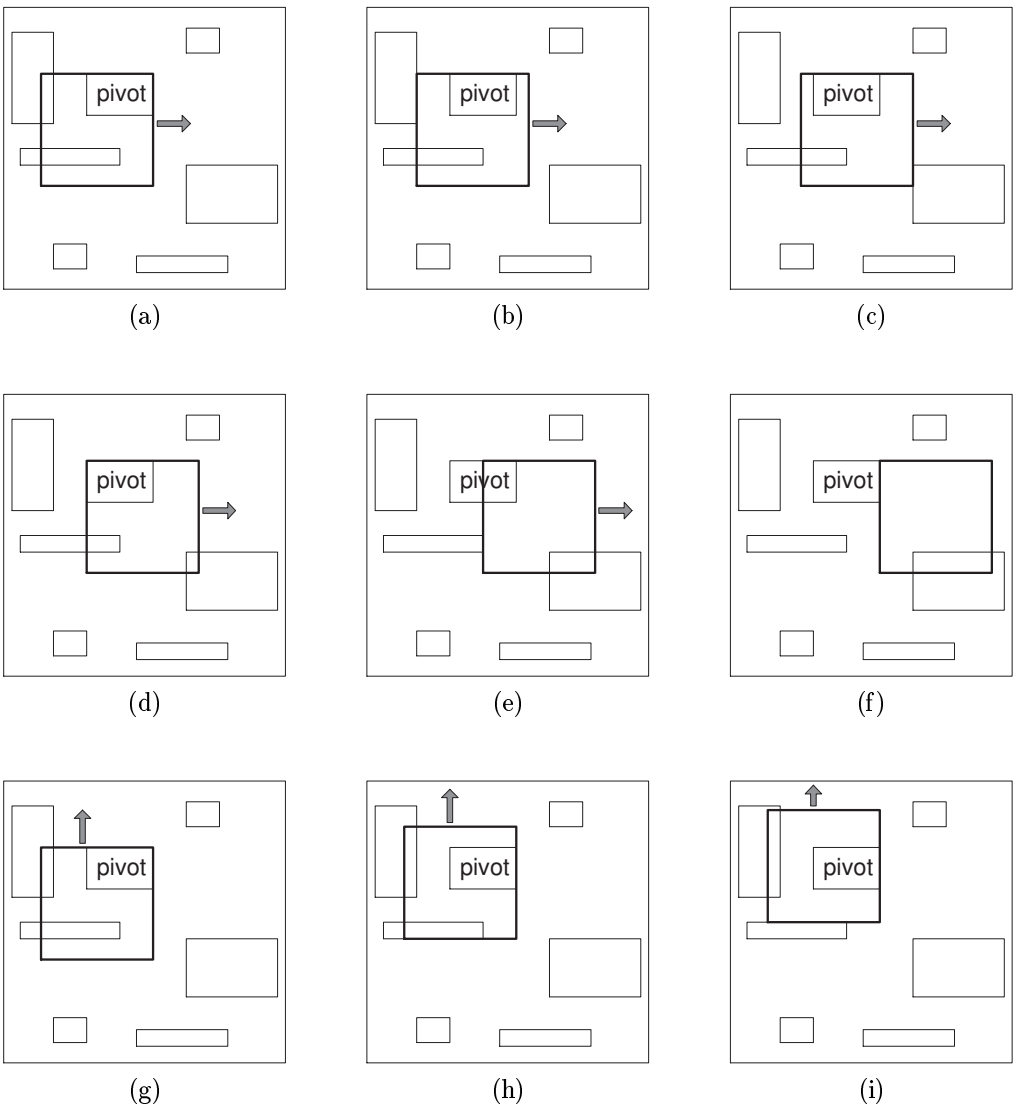


Figure 6: ALG2 starts a window abutting a *pivot rectangle* (a) and slides the window to the right, stopping at each edge that intersects its perimeter (b), until the pivot abuts the opposite side of the window, on the outside (f). Other combinations of the pivot-window orientations are then explored (g-i). This process is repeated for every rectangle, using each as a pivot in turn.

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 the algorithm from Figure 7. 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 of the algorithm checks the rectangle intersections for a

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 lines (3)-(5) for all other starting orientations for W **Output** all extremal-density windows

Figure 7: ALG2: $O(k^2)$ density analysis.

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

Theorem 6 *Given k non-overlapping rectangles with positions uniformly distributed in the $n \times n$ grid, the algorithm in Figure 7 finds the maximum-density $w \times w$ window in time $O(k \cdot E)$, after applying a fixed-dissection preprocessing phase with runtime $O(k \cdot E + (\frac{n}{w})^2 + (\frac{n}{w})^2 \cdot E \cdot \log((\frac{n}{w})^2 \cdot E))$, where E is the expected number of rectangles that intersect an arbitrary $w \times w$ window.*

Proof: Let E be the expected number of rectangles that intersect an arbitrary $w \times w$ window, under a uniform random distribution model. Although we will use E as an indeterminate variable here, the actual value of E (as a function of k, w, and n) will be determined later in Theorem 7 below.

To prove the present theorem, we follow the same overall strategy as in the $O(k^2)$ algorithm described in Section 2.2.3: for each of the k rectangles, we slide a $w \times w$ window W over the pivot rectangle and compute the intersections of the various rectangles with that sliding window. These sliding phases can be performed in time linear in the number of intersecting rectangles, assuming that we can compute this set efficiently. The time for each one of these $O(k)$ scanning phases is therefore dominated by the time to obtain and scan a sorted list of the left and right coordinates of the E rectangles that are expected to intersect each sliding window; as we will see below, this can be accomplished within time $O(E)$ per window, given appropriate preprocessing.

The remaining issue here is how to efficiently find all rectangles that intersect a given fixed-size window as it slides over a pivot rectangle. This is accomplished as follows.

1. Partition the layout area into $\frac{n}{w} \times \frac{n}{w}$ squares of size $w \times w$ each, and create and initialize an $\frac{n}{w} \times \frac{n}{w}$ array corresponding to this tiling.

2. Iterate over all rectangles and mark all the tiles that intersect with each rectangle, thus creating for each tile a list of rectangles that intersect it. Then, sort these lists and put into each array position a pointer to the sorted list containing all rectangles that intersect with the corresponding tile. The sum S of the lengths of all these lists is equal to the number of tiles $(\frac{n}{w})^2$ times the expected number E of rectangles that intersect each tile, so $S = (\frac{n}{w})^2 \cdot E$. The time to create the preprocessed data structure is therefore the sum of the array creation time plus the total time to sort all the lists, which brings the total to $O((\frac{n}{w})^2 + S \log S)$. The total space required by this data structure is $O((\frac{n}{w})^2 + S)$.

3. Given the preprocessing above, we can find all rectangles that intersect a given $w \times (2w)$ query window as follows. First, find all the tiles that intersect the query window: there can be at most 6 of these. Then, merge the corresponding ≤ 6 (presorted) rectangle lists into a single sorted list of rectangles that intersects the query window. The size of each of the sublists is $O(E)$, and there are $O(1)$ of them , so the overall work involved in this step is $O(E)$.

The overall time complexity of the algorithm is therefore the preprocessing time of $O((\frac{n}{w})^2 + (\frac{n}{w})^2 \cdot E \cdot \log((\frac{n}{w})^2 \cdot E))$ plus the time to process each of the $O(k)$ pivots and its associated list of intersected rectangles, i.e., $O(k \cdot E)$, where E is the expected number of rectangles that intersect an arbitrary $w \times w$ window. □

We call this improved-preprocessing algorithm ALG3, and now show that the expected number of rectangles that intersect a given fixed-size window is indeed quite small. We define a "random rectangle" as a rectangle uniquly determined by a pair of opposite corners chosen independently at random from a uniform distribution.

Theorem 7 *Given k random pairwise-disjoint rectangles distributed uniformly in the $n \times n$ layout region, the expected number E of rectangles that intersect a given $w \times w$ window is bounded by $E \leq 3k \cdot \frac{w^2}{n^2} + 3$.*

Proof: Consider the following two types of rectangles that can intersect a given window:

1. Rectangles having at least one of their corners contained inside the window; and
2. Rectangles having none of their corners contained inside the window (yet who still intersect it).

In order to simplify the probabilistic analysis which follows, we allow overlaps to occur among the rectangles. Note that this can only *increase* the expected number of rectangles that intersect a window, because if the non-overlap constraint is enforced, rectangles which intersect a window preclude some other rectangles from intersecting it due to the non-overlap requirement, thereby reducing the expected number of rectangles that may intersect a given window. We analyze separately the expected number of rectangles of each type.

Type 1: rectangles having at least one of their corners contained inside the window. The probability that a type-1 random rectangle[11] will have at least one of its corners inside a fixed $w \times w$ window is equal to 1 minus the probablity that *neither* of the rectangle's two opposite corners are inside the window, i.e., $1 - (\frac{n^2-w^2}{n^2})^2 = 2 \cdot \frac{w^2}{n^2} - \frac{w^4}{n^4} \leq 2 \cdot \frac{w^2}{n^2}$. Thus the expected number of type-1 rectangles that intersect a window is $E_1 \leq 2k \cdot \frac{w^2}{n^2}$.

Next, we account for type-2 rectangles, i.e., those having none of their corners contained inside the window, yet whose area still intersects the window's area. There are three subcases here:

Type 2a: rectangles of type 2 where *one* of the rectangle's edges intersect the perimeter of the window (with the other edge being entirely outside the window's area). Rectangles of this can occur at most twice per window, on opposite edges (by applying the non-overlapping constraint to such rectangles). Thus the expected number of type-2a rectangles that intersect a window is $E_{2a} \leq 2$.

Type 2b: rectangles of type 2 where *two* of the rectangle's edges intersect the perimeter of the window. Rectangles of this type have an occurrence probability less than $(\frac{w}{n})^2$, since both opposite corners of the rectangle in question must independently fall inside the strip of size $w \times n$ containing the window. Note that this is actually an over-estimate, since this probability includes rectangles inside the $w \times n$ strip but strictly outside the window itself; however, this is not a problem since this over-estimate

[11] Recall that a "random rectangle" has two of its opposite corners uniformly and independently distributed in the layout region.

still upper-bounds the actual expectation for case $2b$. Thus the expected number of type-$2b$ rectangles that intersect a window is $E_{2b} \leq k \cdot (\frac{w}{n})^2$.

Type 2c: rectangles of type 2 that completely contain the window. Rectangles of this type can occur at most once per window (since by applying the non-overlapping constraint, such a rectangle will preclude any other rectangles from intersecting the window). Thus the expected number of type-$2c$ rectangles that intersect a window is $E_{2c} \leq 1$.

Thus, the expected number E of rectangles intersecting a window of size $w \times w$ is upper-bounded by the sum of the expectations for case 1, $2a$, $2b$, and $2c$:

$$E \leq E_1 + E_{2a} + E_{2b} + E_{2c} \leq 2k \cdot \frac{w^2}{n^2} + 2 + k \cdot (\frac{w}{n})^2 + 1 = 3k \cdot \frac{w^2}{n^2} + 3 = O(k \cdot \frac{w^2}{n^2})$$

□

In real layouts where rectangles are *disjoint*, even fewer intersections are likely than indicated by the bound above, since some intersections will preclude other intersections by delimiting large areas that no other rectangles may occupy. 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 8 *Given k rectangles in the $n \times n$ layout region, the maximum-density width-w window can be found in time $O((\frac{n}{w})^2 + k \log k + k^2 \cdot (\frac{w}{n})^2)$.* □

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 the extremal-perimeter density criteria.

2.3 Multilevel Density Analysis

The algorithms described in the previous two subsections have two drawbacks: (i) the fast analysis in the fixed-dissection regime may significantly underestimate the maximum density among all $w \times w$-windows in the worst case (Theorem 1), while (ii) the optimal density analysis is too slow when the number of rectangles is large (Corollary 8). We now develop a new *multilevel* approach that attempts to overcome both drawbacks simultaneously. It is based on the following simple fact (see Fig. 8).

Lemma 9 *Given a fixed r-dissection, any arbitrary $w \times w$ window will contain some* shrunk $w(1-1/r) \times w(1-1/r)$ *window of the fixed r-dissection, and will be contained in some* bloated $w(1+1/r) \times w(1+1/r)$ *window of the fixed r-dissection.* □

We suggest the following ideas.

- Lemma 9 suggests that the possible error of the fixed-dissection approximation can be estimated more accurately than in Theorem 1. Our first idea is that if we find the area of not only *standard* windows (i.e., fixed r-dissection windows consisting of $r \times r$ tiles) but also *bloated* windows (i.e., fixed r-dissection windows consisting of $(r+1) \times (r+1)$ tiles), then the maximum area of a *floating* window (i.e., arbitrary $w \times w$-window) can be bounded by the maximum area of a bloated window (see Figure 8).

- Our second idea is to use zooming to make fixed-dissection density analysis for any given $r = r_0$ even faster. The main points of this approach are: (i) starting with one fixed r-dissection ($r = 1$), omit all tiles which do not belong to any bloated window that can possibly contain high-density floating windows, and (ii) recursively subdivide the remaining tiles into 4 subtiles (i.e., multiply r by 2) until the necessary $r = r_0$ is reached.

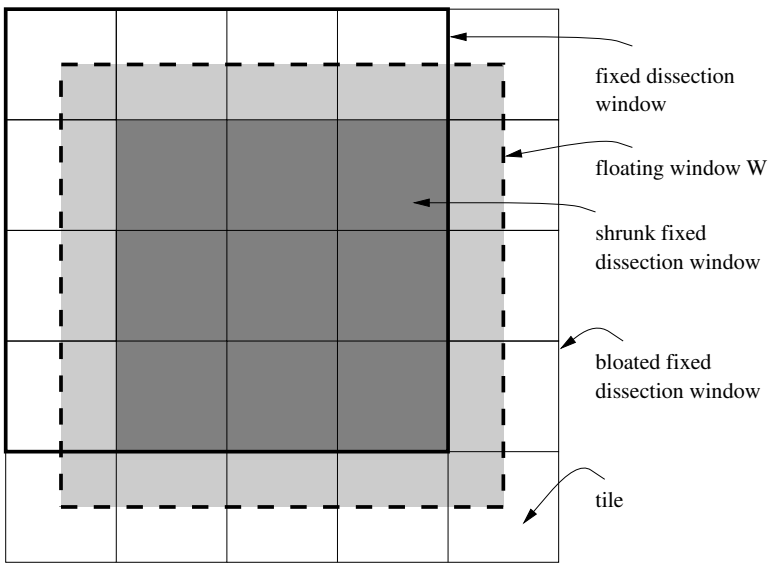


Figure 8: An arbitrary floating $w \times w$-window W always contains a *shrunk* $(r-1) \times (r-1)$-window of a fixed r-dissection, and is always covered by a *bloated* $(r+1) \times (r+1)$-window of the fixed r-dissection. In the figure, a *standard* $r \times r$ fixed-dissection window is shown with thick border. A floating window is shown in light gray. The white window is the *bloated* fixed-dissection window, and the dark gray window is the *shrunk* fixed-dissection window.

- Our third idea is that the recursive subdivision may be continued until the number of rectangles left in tiles is sufficiently small to run the optimal density analysis algorithm ALG3. Alternatively, the subdivision can be terminated at the moment when some *user-defined accuracy*, say 2%, is reached.

The following algorithm is a formal implementation of the above ideas. We use $\epsilon > 0$ to denote the user-defined accuracy that is required in finding the maximum window density. The lists $TILES$ and $WINDOWS$ are byproducts of the analysis, which will be used in Section 3.2 below to find the optimal amounts of fill geometries to add into the corresponding tiles.

Since any floating $w \times w$-window W is contained in some bloated window, the filled area in W ranges between Max (maximum $w \times w$-window filled area found so far) and $BloatMax$ (maximum bloated window filled area found so far). The algorithm terminates when the relative gap between Max and $BloatMax$ is at most $2 \cdot \epsilon$, and then outputs the middle of the range $(Max, BloatMax)$.

The runtime of multilevel density analysis depends on ϵ. At each iteration of the main loop (3) the difference in area between the bloated and standard window is reduced by half. The loop (3) terminates when the original area difference $3w^2$ decreases to 2ϵ after t iterations, i.e.,

$$\frac{3w^2}{2^t} = 2\epsilon$$

Thus, the maximum number of iterations T can be estimated as

$$T = \log_2(1.5w^2 \cdot \epsilon^{-1}) = O(\log(w/\epsilon))$$

This formula implies a worst-case runtime of $O((\frac{n}{w} \log \frac{w}{\epsilon})^2)$. In practice, the layout is unevenly filled and the majority of tiles are dropped in early iterations of the main loop (3). This explains the excellent

Multi-Level Density Analysis Algorithm
Input: $n \times n$ layout and accuracy $\epsilon > 0$ **Output:** maximum area density of $w \times w$ window with accuracy ϵ
(1) make a list $ActiveTiles$ of all $w/r \times w/r$-tiles (2) $Accuracy = \infty$, $r = 1$ (3) While $Accuracy > 1 + 2\epsilon$ do (a) find all rectangles in $w/r \times w/r$-tiles from $ActiveTiles$ (b) find area of each standard window consisting of tiles from $ActiveTiles$ and add such window to the list $WINDOWS$ (c) Max = maximum area of standard window with tiles from $ActiveTiles$ (d) $BloatMax$ = maximum area of bloated window with tiles from $ActiveTiles$ (e) For each tile T from $ActiveTiles$ which do not belong to any bloated window of area more than Max do if $Accuracy > 1 + \epsilon$, then put T in $TILES$ remove T from $ActiveTiles$ (f) replace in $ActiveTiles$ each tile with four of its subtiles (g) $Accuracy = BloatMax/Max$, $r = 2r$ (4) Move all tiles from $ActiveTiles$ to $TILES$ (5) Output maximum window density = $(Max + BloatMax)/(2 \cdot w^2)$

Figure 9: Multi-level density analysis algorithm.

performance of multilevel density analysis for actual VLSI layouts (see Subsection 5.2).[12]

3 Computing the Optimal Fill Amount

To solve the Filling Problem, it is necessary to compute the proper *fill amount* that should be added in each particular tile. In the next subsection, we develop an *optimal* linear program solution for the fixed-dissection regime. Then, two modifications of the LP formulation are described in the following subsections. The first modification is applied to the output of multilevel density analysis; the second modification uses window area bounds from Lemma 9 to minimize an estimate of the maximum deviation among arbitrary (floating) windows.

3.1 Minimizing Density Variation in the Fixed-Dissection Regime

This subsection develops exact solutions to the Filling Problem in the *fixed-dissection* regime. Recall that Theorem 2 indicates that if $r = 10$ and all windows of a fixed r-dissection have feature area density at most 75% (i.e., $U = 0.75$), then the density of *any* $w \times w$ window in the layout is at most 85%. Theorem 2 thus allows us to consider the Filling Problem for only a fixed r-dissection of the layout, i.e., we will analyze density with respect to each $w \times w$-window W that covers exactly r^2 tiles. Desired accuracy of the result is achieved by increasing r.

For any given tile $T = T_{ij}, i, j = 1, \ldots, \frac{nr}{w}$, denote the total feature area inside T as $area(T)$. We define the *slack* of T, $slack(T)$, as the maximum fill amount that can be introduced using a given fill pattern into T without violating the density upper bound U in any window containing T. In other words, the total layout feature area inside T can be increased up to any value between $area(T)$ and $area(T) + slack(T)$, using fill geometries. The slack of T is determined by the total area of metal features inside T *and its neighbor tiles*. The slack of a window W is the sum of the slacks of the tiles

[12]The multilevel analysis can also be applied in finding minimum window density. By Lemma 9, the minimum layout area in *shrunk* windows (i.e., fixed r-dissection windows consisting of $(r-1) \times (r-1)$ tiles) is a lower bound for the layout area in an arbitrary $w \times w$ window. Therefore, the multilevel algorithm can be easy modified to find minimum window density with user-defined accuracy.

that form W (efficient algorithms for slack computation are discussed in Section 3.1.2 below). Using the concept of slack, the Filling Problem for the fixed-dissection regime can be formulated as follows.

The Filling Problem for a fixed r-dissection. Suppose we are given a fixed r-dissection of the layout into tiles of size $\frac{w}{r} \times \frac{w}{r}$, as well as an $area(T)$ and $slack(T)$ for each tile in the dissection. Then, for each tile T_{ij}, the total fill pattern area $p_{ij} = p(T_{ij})$ to be added to T_{ij} must satisfy

$$0 \leq p_{ij} \leq slack(T_{ij})$$

and

$$\sum_{T_{ij} \in W} p_{ij} \leq \max\{U \cdot w^2 - area(W), 0\} \tag{1}$$

for any fixed dissection $w \times w$-window W.

Then, the **Min-Variation Formulation** seeks to maximize the minimum window density:

$$maximize \quad \left(\min_{ij} (area(T_{ij}) + p_{ij}) \right)$$

3.1.1 A Linear Programming Approach

Consider the linear program:

Maximize M

subject to:

$$p_{ij} \geq 0, \quad i, j = 1, \ldots, \frac{nr}{w} - 1 \tag{2}$$

$$p_{ij} \leq pattern \cdot slack(T_{ij}), \quad i, j = 1, \ldots, \frac{nr}{w} - 1 \tag{3}$$

$$\sum_{s=i}^{i+r-1} \sum_{t=j}^{j+r-1} p_{st} \leq \alpha_{ij} \left(U \cdot w^2 - area_{ij} \right), \quad i, j = 1, \ldots, \frac{nr}{w} - r + 1 \tag{4}$$

$$M \leq area_{ij} + \sum_{s=i}^{i+r-1} \sum_{t=j}^{j+r-1} p_{st}, \quad i, j = 1, \ldots, \frac{nr}{w} - r + 1 \tag{5}$$

where

$$area_{ij} = \sum_{s=i}^{i+r-1} \sum_{t=j}^{j+r-1} area(T_{st})$$

is the area of the (i, j)-th window, and $\alpha_{ij} = 0$ if $area_{ij} > U \cdot w^2$ and 1 otherwise. Also, the pattern-dependent coefficient *pattern* denotes the maximum pattern area which can embedded in an empty unit square.

The constraints (2) imply that features can only be added, and cannot be deleted from any tile. The slack constraints (3) are computed for each tile. The pattern-dependent coefficient *pattern* denotes the maximum pattern area which can embedded in an empty unit square. If a tile T_{ij} is originally overfilled, then we set $slack(T_{ij}) = 0$. From the linear programming solution, the values of p_{ij} indicate the fill amount to be inserted in each tile T_{ij}. The constraint (4) says that no window can have density more than U after filling unless it was overfilled initially, i.e., such a window cannot increase its density. The number of variables and the number of constraints in the linear program are both $O((\frac{nr}{w})^2)$. In

practice, even for a large die and a user requirement of high accuracy, we might have $n = 15000$, $w = 3000$, $r = 10$, which yields a linear program of tractable size. Equation (5) implies that the auxiliary variable M is the lower bound on all window densities. The linear programming seeks to maximize M, thus achieving the min-variation objective.

Solving the above LP formulation will give the optimal fill amounts to be added to each tile in the fixed r-dissection, as dictated by the Min Variation objective. However, as shown in Figure 3, the LP solution may distribute the fill unevenly among the tiles of a given window. If this is unsatisfactory, various simple fixes can be applied (e.g., partial pre-filling of all tiles, binary search on an upper bound of fill added into each individual tile, etc.) so that the result is more balanced while still being optimal. (Our current implementation sets an upper bound U_t on the tile density in order to achieve a balanced fill pattern.)

3.1.2 Slack Computation

This subsection discusses how to efficiently compute slack values for the linear programming formulation described in the previous subsection. To compute slack, i.e. to determine the total area of k possibly overlapping rectangles, we adopt the "measure of union of rectangles" sweep-line -based technique described in [21]. We begin by sorting all the left and right edges of the k rectilinear rectangles according to their x coordinates. Next, we sweep horizontally across these $2k$ edges from left to right, while using a segment tree [2] to keep track of the total length of the sweep line intersected by any of the k rectangles (see Figure 10).

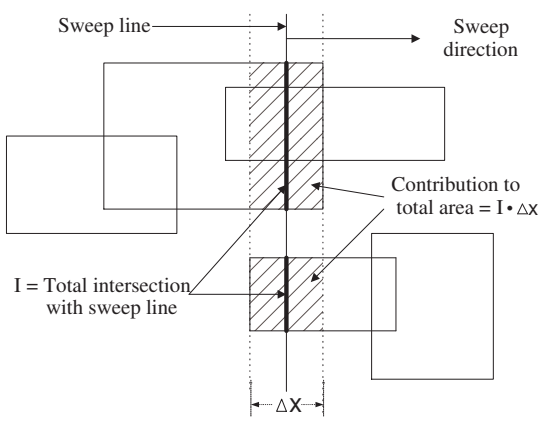


Figure 10: Finding the total area of a union of possibly intersecting rectangles using a sweep line technique.

The time complexity of the sorting step is $O(k \log k)$. Insertions and deletions from the segment tree require $O(\log k)$ time each, and the total time to process all $2k$ segments is therefore $O(k \log k)$. The total time complexity to determine the area of the union of k possibly overlapping rectangles is therefore $O(k \log k)$.

A simple implementation which avoids the usage of segment trees altogether can still have reasonably fast expected time as follows. We still use the sweep line technique as before, but rather than using a segment tree to store the intersected rectangle, we instead use a simple linked list to store those segments, and then apply the one-dimensional "measure of union of intervals" technique of [14]. The

time complexity of this practical implementation is $O(k^2)$ in the worst-case, and the expected time is $O(k \cdot l)$ where l is the average length of this list (i.e., the expected number of rectangles intersected by the sweep line). For random uniform distributions, we would expect $l = O(\sqrt{k})$, thus on average this method will run in time $O(k\sqrt{k})$ in practice.

3.2 Multi-Level Computation of the Fill Amount

If the multilevel density analysis approach has been used, we can use data obtained during that computation to compute fill amounts. Recall that during the multilevel density analysis, we keep track of active tiles (i.e. tiles which can possibly belong to a maximum density window) and check the area of some windows in order to update the maximum window density if necessary. The multilevel computation of fill amounts attempts to decrease the number of tiles and windows, i.e., variables and constraints participating in the LP formulation. Let $r_{max} = 2^{l_{max}}$ be the highest r reached in the multilevel density analysis algorithm; this corresponds to the user-defined accuracy parameter ϵ. Instead of considering all $\frac{w}{r_{max}} \times \frac{w}{r_{max}}$-tiles and all $w \times w$-windows consisting of such tiles, we propose to consider only tiles $\frac{w}{2^l} \times \frac{w}{2^l}$-tiles, $l \leq l_{max}$, and windows consisting of such tiles which were tried during the multilevel density analysis.

The multilevel fill amount computation is implemented as follows. During multilevel density analysis, we save a tile in $TILES$ at the moment when the tile is deactivated (cannot belong to a window of maximum density) or the size of the tile becomes $\frac{w}{r_{max}} \times \frac{w}{r_{max}}$. We also record the area and slack of each such tile. On the other hand, each time when we find the area of a $w \times w$-window W, we put W in the list of windows $WINDOWS$. In the LP formulation for multilevel fill amount computation, for each window W from $WINDOWS$ there are two constraints: (i) the first constraint upper-bounds the filled area (i.e., the area after fill geometries are added to the original layout) of W, and (ii) the second constraint forces an auxiliary variable M to be less than or equal to the filled area in W. Each filled window area is expressed as a sum of filled tile areas. In addition, tile fill amount constraints ensure that each tile fill amount is nonnegative, and at most the corresponding tile slack $\times$*pattern*.

3.3 Minimizing Density Variation of Arbitrary (Floating) $w \times w$-Windows

Finally, we suggest a third LP formulation that may better reflect the quality of the fill amount computation. Again, this is because the linear program for the fixed-dissection regime will be susceptible to density deviations in floating windows. Consider two different LP solutions in fixed-dissection regime with different number of fixed dissections: the first has r^2 dissections and the second has $(2r)^2$ dissections. It is obvious that the more dissections we take in account the better result we should have. On the other hand, more dissections imply more constraints in the LP and, therefore, worse (bigger) deviation achieved (i.e., smaller value of target variable M). A fair comparison of results with different number of fixed dissections entails finding the *floating deviation*, i.e., the difference between the minimum and maximum floating window density. However, since the number of floating windows is too large, we suggest comparing *worst-case* estimates of the floating deviation, which can be derived from Lemma 9.

Moreover, instead of comparing LP solutions according to the above estimate of floating deviation, we suggest using such an estimate as an *objective* in a new LP formulation. Specifically, we constrain the area of each bloated $w(1 + 1/r) \times w(1 + 1/r)$-window by the user-defined density upper bound U, and we maximize the auxiliary variable M which is the lower bound for the area of *any* $w(1 - 1/r) \times w(1 - 1/r)$-window. We refer to this LP formulation as the *floating deviation* LP. The floating deviation LP formulation optimally decreases the estimate of the density range between the maximum- and minimum-density floating windows.

4 Synthesis of Filling Patterns

Given the layout geometry along with the parameters of the Filling Problem, we apply the methods of previous sections to analyze density violations, and determine the necessary amounts of fill to be added in each region of the layout. We now discuss criteria for, and actual synthesis of, the fill geometries added into the layout.

4.1 Uniform Coupling to Long Conductors

Fill patterns should be devised such that all long conductors on adjacent layers have identical coupling capacitance to the inserted fill.[13] There are several practical ways of achieving this, of which one is to "basket-weave" the fill [30]. In other words, the fill pattern should not consist of a regular grid geometries, but instead have some internal offsets that "skew" the pattern. Figure 11 illustrates this concept.

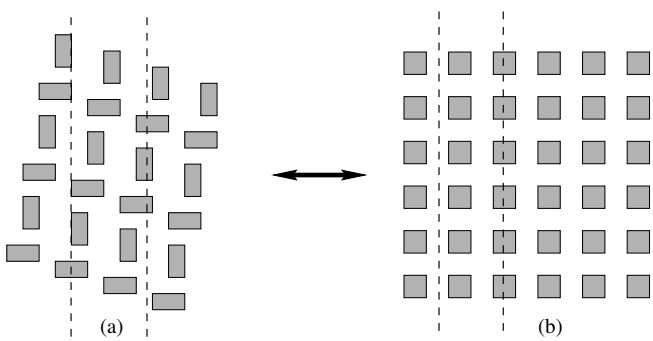


Figure 11: "Basket-weaving" of the fill pattern so that long conductors on adjacent layers will have identical coupling to the fill. With the pattern in (a), each vertical or horizontal crossover line will have the same overlap capacitance to fill. On the other hand, with the fill pattern in (b) two crossovers can have different coupling to fill.

4.2 Grounded vs. Floating Fill

Grounded fill can be required for *predictable* extracted parasitic values. Structured-custom (microprocessor) designs have strong requirements for predictability, due to aggressive timing tolerances. For such designs, it is better to have larger, but exactly known, coupling capacitances to grounded fill geometries, rather than indeterminate capacitances to floating fill. On the other hand, for ASIC designs where timing is not being pushed too hard, designers seek the simplest fill construction that meets feature density requirements. A secondary reason for studying grounded-fill constructions is that modern parasitic extraction tools do not handle floating capacitors well. If fill synthesis should be performed earlier so as to achieve an accurate performance verification flow during the layout phase, it may be necessary to use grounded fill.

[13] Coupling to same-layer conductors is not a concern, because the buffer distance B is usually quite large, on the order of 10 μm or more.

We seek a grounded fill pattern that requires relatively few edges to specify. For example, a metal fill pattern consisting mostly of long parallel stripes is preferable to a checker-board pattern, since the number of lines required to fully specify the latter is considerably smaller than the former. Thus, we propose a grounded metal fill pattern that spans the area to be filled as follows. We start by striping the empty areas in the layout using horizontal lines (see Figure 12b). Then, we span the horizontal stripes using vertical lines (see Figure 12c). The width and pitch of the horizontal stripes, and the number of vertical segments, can be easily determined in terms of the required pattern density. Connections to an existing ground distribution network can be made using standard special-net routers.[14]

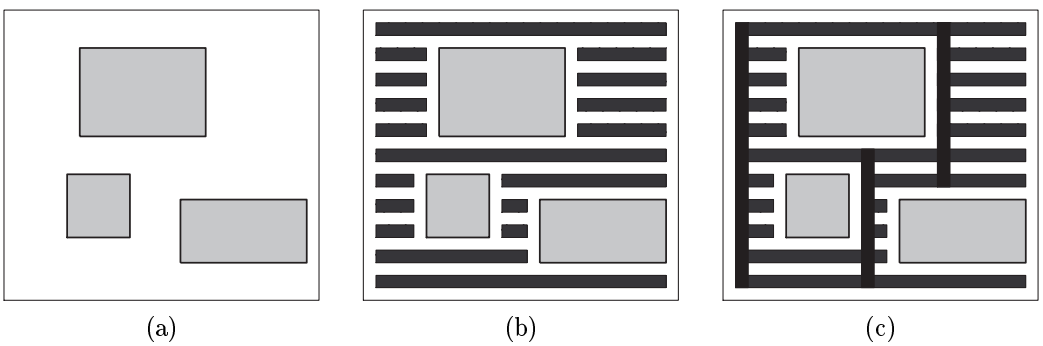


Figure 12: Given a layout (a), we create a grounded fill pattern by first (b) creating horizontal stripes, and then (c) spanning these stripes using a small number of vertical lines.

4.3 Simultaneous Area and Perimeter Constraints

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 practice, the distance between filling 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 [11] 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 13(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 13(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

[14] An interesting possibility arises if separate ground planes of metalization are used in between signal layers (as in printed-circuit board construction), in which case grounded fill patterns can look similar to floating fill patterns (connections to ground are achieved by vias down to the adjacent layer).

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 Figure 14, 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.

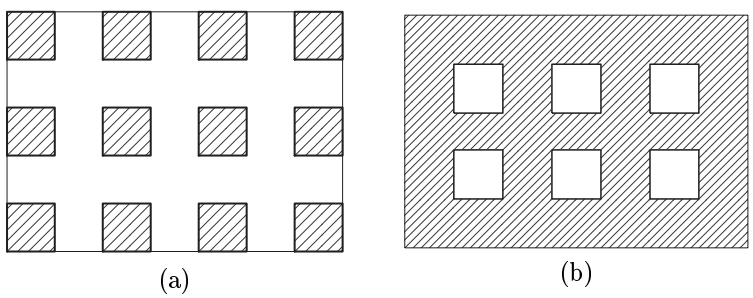


Figure 13: Two patterns with maximum perimeter. (a) The pattern P_{min} with minimum possible area, and (b) the pattern P_{max} with maximum area.

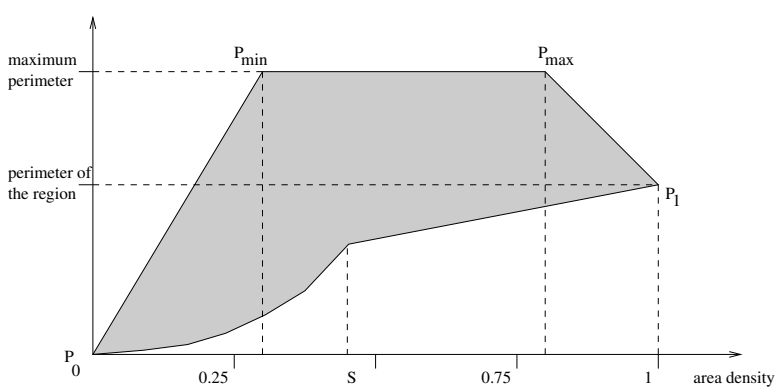


Figure 14: The x-axis represents the area and the y-axis represents the perimeter of the filling pattern. The highlighted region with vertices P_0, P_{min}, P_{max}, and P_1 represents the combinations of area and perimeter for which there exist filling patterns. The pattern with the area S minimum perimeter is the largest square which can be embedded in R. Before the pattern area reaches S, the minimum perimeter grows quadratically; when the area exceeds S, the minimum perimeter grows linearly.

5 Implementation and Computational Results

5.1 Implementation

Our current experimental testbed integrates GDSII Stream input, conversion to CIF format, and internally-developed geometric processing engines. For density analysis, the user specifies the parameters w, r, B, U, L, etc., and receives output indicating extremal window densities, average window density, and lists of violating windows. For density improvement, the user specifies additional parameters such as the maximum possible fill pattern density (used to compute available slacks in each tile). The program outputs whether the density lower bound L can be achieved (and if so, the maximum achievable density lower bound L'), and the amounts of fill p_{ij} that should be introduced into each tile. Finally, for pattern insertion the user specifies further parameters, including the type of fill pattern desired (rectangular grid or basket-weave floating fill), and a parametric specification of the pattern. The program outputs the final layout, including the added fill geometries, in CIF format.

5.2 Computational Experience

Industry Test Cases			
Benchmark	N = layout size	k = # rectangles	w = window size
L1	125,000	49,506	31,250
L2	112,000	76,423	28,000
L3	112,000	133,201	28,000

Table 1: Parameters of three industry test cases.

Our experiments have been run using three metal layers extracted from industry standard-cell layouts. Benchmark L1 is the M2 layer from an 8131-cell design; Benchmark L2 is the M3 layer from a 20577-cell design; and Benchmark L3 is the M2 layer from the same 20577-cell design. The layout dimension, number of rectangles, and window size (w always chosen to equal 1.5mm) for each test case are shown in Table 1.

Table 2 reports the maximum window density found by the multilevel density analysis with accuracy parameter ϵ set to either 2% or 3%. We report the maximum density of a standard window, rather than the midpoint between the maximum density standard and bloated windows, in order to enable comparison with the fixed-dissection analysis results below. CPU time corresponds to seconds on a 140MHz Sun Ultra-1 with 256MB RAM. In practice, we find that the multilevel analysis is preferable to the exact method of ALG3, since the latter has runtimes on the order of tens of CPU minutes for the same test cases (see [13] for ALG3 runtimes on slightly variant layouts of the same standard-cell designs). Table 3 shows the analogous results for the fixed-dissection approach, which we understand to be used in industry. The two tables show that large values of r, and correspondingly large runtimes, will be required for the fixed-dissection approach to find the window densities that the multilevel analysis can find in only a few seconds.

Last, Table 4 depicts the performance of our software for LP generation and solution to obtain optimal fill amounts, along with fill insertion into the output CIF file. The fixed-dissection LP achieves lower bounds M on density that are reasonably close to the best possible values.[15] Larger r values may be used to reduce the potential variation from uniform density. For the multilevel density analysis, LP and fill generation, we observe somewhat lower values of M, and one large runtime for the L2 benchmark with $r = 8$. The floating deviation LP, which allows the user to bound the density variation between

[15] In all reported experiments, we use U = maximum window density found during multilevel density analysis. Experiments with larger values of U, e.g., $U = 0.50$, yield similar results.

Multilevel Density Analysis			
Benchmark	Accuracy	Max Std Density	CPU time
L1	2%	.2184	2.8
L1	3%	.2184	2.8
L2	2%	.1830	6.9
L2	3%	.1829	3.8
L3	2%	.2925	7.1
L3	3%	.2911	6.6

Table 2: Multilevel density analysis results. We report the maximum density of a standard window, rather than the midpoint between the maximum density standard and bloated windows, in order to enable comparison with the fixed-dissection analysis results below.

Fixed-Dissection Density Analysis			
Benchmark	r	Max Density	CPU Time
L1	2	.2021	1.3
L1	4	.2125	2.9
L1	8	.2170	9.2
L2	2	.1610	2.1
L2	4	.1791	4.5
L2	8	.1791	14.5
L3	2	.2883	3.6
L3	4	.2895	8.0
L3	8	.2910	25.1

Table 3: Fixed-dissection density analysis results.

minimum- and maximum-density *floating* windows, shows steady improvement in solution quality with increasing r, just as we expect. In practice, we believe that the floating deviation LP is attractive for its control over arbitrary windows; the multilevel LP is also attractive for its data flow directly from multilevel density analysis.

6 Conclusions and Future Directions

In conclusion, we have addressed an increasingly critical problem in the interface between process, physical layout design and performance verification. We have given the first statement of the *filling problem* (with min-variation objective), which arises in layout post-processing for CMP uniformity. We have developed effective algorithms for density analysis as well as for filling synthesis. Our current experimental testbed integrates GDSII Stream input, conversion to CIF format, and internally-developed geometric processing engines. Runtimes show that the proposed techniques are practically useful.

We are currently seeking more test cases and density rules from industry to further refine the proposed approaches and implementations.[16] Ongoing work addresses such issues as:

- developing even more efficient, general and provably-good filling algorithms (e.g., for simultaneous

[16] Interesting test cases for filling 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 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.

Fixed-Dissection LP for Fill Amount and Fill Generation						
Benchmark	r	LP generation CPU time	LP solution CPU time	M	Fill CPU time	Total CPU time
L1	2	4.3	0.0	.2192	3.3	7.6
L1	4	4.0	0.4	.2192	3.2	7.6
L1	8	10.3	18.3	.2189	3.3	31.9
L2	2	2.8	0.0	.1816	5.2	8.0
L2	4	5.2	1.7	.1704	5.0	11.9
L2	8	15.8	41.5	.1631	5.2	62.5
L3	2	5.2	0.0	.2640	8.3	13.5
L3	4	9.4	0.8	.2606	8.0	18.2
L3	8	27.2	24.4	.2553	8.1	59.7

Multilevel LP for Fill Amount and Fill Generation						
Benchmark	r_{max}	LP generation CPU time	LP solution CPU time	M	Fill CPU time	Total CPU time
L1	4	2.8	0.4	.2192	4.2	7.4
L1	8	2.7	20.3	.2192	4.1	27.1
L2	4	3.1	5.9	.1834	6.8	15.8
L2	8	3.8	376.5	.1834	6.6	386.9
L3	4	9.4	1.0	.1761	9.6	20.0
L3	8	10.0	51.5	.1745	10.3	71.8

Floating Deviation LP for Fill Amount and Fill Generation						
Benchmark	r	LP generation CPU time	LP solution CPU time	M	Fill CPU time	Total CPU time
L1	2	4.3	0.0	.0546	3.2	7.5
L1	4	4.0	0.4	.1218	3.3	7.7
L1	8	10.3	5.8	.1615	3.2	19.3
L2	2	2.8	0.0	.0437	5.2	8.0
L2	4	5.2	0.4	.0827	5.2	10.8
L2	8	15.8	43.8	.1037	5.3	64.9
L3	2	5.2	0.0	.0677	9.1	14.3
L3	4	9.4	0.8	.1451	10.0	20.2
L3	8	27.2	18.4	.1875	8.5	54.1

Table 4: Experimental results showing CPU times for three variant LP approaches to computing optimal fill amounts; CPU times for fill generation are also shown.

perimeter- and area-density based criteria);

- finding improved heuristics or exact algorithms for the min-variation formulation;
- maintaining knowledge of min/max density/perimeter windows under dynamic feature insertion/deletion in time $o(n)$ or $o(k)$;
- calibrating our proposed methods against data and density control requirements from industry partners; and
- extending the present infrastructure to address, in a unified way, requirements for both slotting (metal stress relief) and filling at other length scales (micro-loading, iso-dense) in combination with the current requirements.

7 Acknowledgments

We thank the anonymous reviewers for their valuable comments on the previous draft of this work. We thank Tom Laidig and Kurt Wampler of MicroUnity Systems Engineering, Inc. for many illuminating

discussions and patient readings of early drafts during 1997. Juan Rey of Cadence has also been generous with his time. We thank Larry Camilletti and Duane Boning for enlightening discussions. We appreciate the help of Huijuan Wang with the Raphael simulations, we thank Nayantara Gupta for help with implementing the user interface, and we thank Tongtong Zhang and Chris Helvig for their help with proofreading. 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] J. L. Bentley, *Algorithms for Klee's Rectangle Problems.* unpublished manuscript, 1997.

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

[4] L. E. Camilletti, personal communication, April, 1998.

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

[6] A. Chatterjee, I. Ali, K. Joyner, and D. M. et al, *Integration of Unit Processes in a Shallow Trench Isolation Module for a 0.25 μm Complementary Metal-Oxide Semiconductor Technology*, Journal of Vacuum Science and Technology B, 15 (1997), pp. 1936–1942.

[7] V. K. R. Chiluvuri and I. Koren, *Layout-Synthesis Techniques for Yield Enhancement*, IEEE Trans. Semiconductor Manufacturing, 8 (1995), pp. 178–187.

[8] R. R. Divecha, B. E. Stine, D. O. Ouma, J. U. Yoon, D. S. Boning, J. E. Chung, O. S. Nakagawa, and S. Y. Oh, *Effect of Fine-line Density and Pitch on Interconnect ILD Thickness Variation in Oxide CMP Process*, in Proc. CMP-MIC, Santa Clara, February 1997, p. 29.

[9] R. R. Divecha, B. E. Stine, D. O. Ouma, J. U. Yoon, D. S. Boning, J. E. Chung, O. S. Nakagawa, and S. Y. Oh, *Effect of Fine-line Density and Pitch on Interconnect ILD Thickness Variation in Oxide CMP Process*, in Proc. CMP-MIC, Santa Clara, February 1998.

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

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

[12] M. Hanan, *On Steiner's Problem With Rectilinear Distance*, SIAM J. Applied Math., 14 (1966), pp. 255–265.

[13] A. B. Kahng, G. Robins, A. Singh, H. Wang, and A. Zelikovsky, *Filling and Slotting: Analysis and Algorithms*, in Proc. International Symposium on Physical Design, Monterey, CA, April 1998, pp. 95–102.

[14] V. Klee, *Can the Measure of* $\cup[a_i, b_i]$ *be Computed in Less than* $O(n \log n)$ *Steps?*, American Mathematical Monthly, 84 (1977), pp. 284–285.

[15] 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.

[16] W. Maly, *Computer-aided design for VLSI circuit manufacturability*, Proceedings of IEEE, 78 (1990), pp. 356–392.

[17] W. Maly, *Moore's Law and Physical Design of ICs*, in Proc. International Symposium on Physical Design, Monterey, California, April 1998. special address.

[18] S. Nag and A. Chatterjee, *Shallow Trench Isolation for Sub-0.25-mu m IC Technologies*, Solid State Technology, 40 (1997), pp. 129–130, 132, 134, 136.

[19] G. Nanz and L. E. Camilletti, *Modeling of Chemical-Mechanical Polishing: A Review*, IEEE Trans. on Semiconductor Manufacturing, 8 (1995), pp. 382–389.

[20] S. Prasad, W. Loh, A. Kapoor, E. Chang, B. Stine, D. Boning, and J. Chung, *Statistical metrology for characterizing CMP processes*, Microelectronic Engineering, 33 (1997), pp. 231–240.

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

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

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

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

[25] K. Smekalin, *CMP Dishing Effects in Shallow Trench Isolation*, Solid State Technology, 40 (1997), pp. 187–188, 190, 192, 194.

[26] B. E. Stine, D. S. Boning, J. E. Chung, and L. Camilletti, *The Physical and Electrical Effects of Metal-fill Patterning Practices for Oxide Chemical-Mechanical Polishing Processes*, IEEE Transactions on Electron Devices, 45 (1998), pp. 665–679.

[27] B. E. Stine, V. Mehrotra, D. S. Boning, J. E. Chung, and D. J. Ciplickas, *A Simulation Methodology for Assessing the Impact of Spatial/Pattern Dependent Interconnect Parameter Variation on Circuit Performance*, in IEDM Technical Digest, 1997, pp. 133–136.

[28] B. E. Stine, D. O. Ouma, R. R. Divecha, D. S. Boning, J. E. Chung, D. L. Hetherington, C. R. Harwood, O. S. Nakagawa, and S.-Y. Oh, *Rapid characterization and modeling of pattern-dependent variation in chemical-mechanical polishing*, IEEE Transactions on Semiconductor Manufacturing, 11 (1998), pp. 129–140.

[29] M. Tomozawa, *Oxide CMP Mechanisms*, Solid State Technology, (1997), pp. 169–175.

[30] K. Wampler and T. Laidig, personal communication, September, 1997.

Appendix: Fill Impact on Extraction

Table 5 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.[17] 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 inaccurate 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. 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 insertion and the increases in data volume.

Victim Layer Total Capacitance (10^{-15}F)			
Same layer-i neighbors?	Fill layers $i-1$, $i+1$?	$\epsilon = 3.9$	$\epsilon = 2.7$
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 5: 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 (see Figure 15(b)). The dielectric permittivity was set to both 3.9 (for SiO_2) and 2.7 (cf. recent announcements by Sematech [23] of new low-permittivity dielectric technologies). Layers $i-2$ and $i+2$ are set to be 40×40 ground planes.

Victim B Total Capacitance (10^{-15}F)			
Fill layer offset	Fill geometry	$\epsilon = 3.9$	$\epsilon = 2.7$
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 6: TMA/Avant! Raphael capacitance extraction results: total capacitance for the middle victim conductor B.

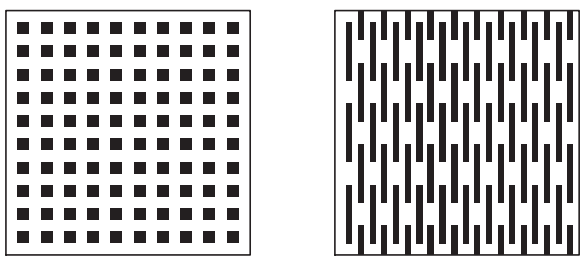
Tables 6 and 7 give TMA/Avant! Raphael capacitance extraction results for multi-layer interconnect structures involving fill geometries, as follows.

- 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 $i-2$.

[17] The use of the Raphael field solver correctly models floating geometries according to the abilities of the tool. However, typical use of the capacitance extraction in Raphael is for grounded features, and the tool may not be optimized for accuracy on floating features.

Victim A, C Total Capacitance (10^{-15}F)			
Fill layer offset	Fill geometry	$\epsilon = 3.9$	$\epsilon = 2.7$
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 7: TMA/Avant! Raphael capacitance extraction results: total capacitance for the outside victim conductor A or C.



(a) (b)

Figure 15: 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 5.

- Two types of fill geometry patterns were considered for layer $i-1$ (see Figure 15): (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 (see Figure 15(a)), 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 (see Figure 15(b)).
- 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 6 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 7 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 can, subject to constraints involving feature dependencies between layers, be viewed as a "single-layer problem".[18]

[18] There are several notable conditions under which the single-layer assumption is not quite correct. For example, poly fill geometry in regions with underlying active diffusion can create spurious transistors, and such regions must be marked as inviolate on the poly layer. This is a simple preprocessing step before slack calculation and fill insertion.