A Simple Fast Exact Density Calculation Algorithm

Hua Xiang
IBM T.J. Watson
Yorktown Heights NY

Chris Chu
Iowa State University
Ames IA

Ruchir Puri
IBM T.J. Watson
Yorktown Heights NY

Abstract

VLSI technology is facing an extreme challenge due to the miniaturization and complexity of leading-edge products. Density control is a must step to ensure the yield and performance for the manufacturing smaller, faster and cheaper chips. A fundamental problem in the density control is how to calculate density correctly and efficiently. In this paper, we propose a simple but efficient two-level hierarchical approach to exactly identify the maximum density window for a given layout. Comparing with the latest work [7], the new algorithm shows big runtime reductions on testcases which have a long runtime with [7].

1. Introduction

Achieving high-yielding designs in the state of the art, VLSI technology is facing an extreme challenge due to the miniaturization and complexity of leading-edge products. In order to produce smaller, faster and cheaper chips, manufacturing issues are catching more and more attention. Density control is no doubt an unavoidable step to reduce the process variations and improve performance predictability and yield. A fundamental problem in the density control is how to calculate density correctly and efficiently.

The density calculation problem is first addressed in [1] as "Extremal - Density Window Analysis" problem.

Extremal Density Window Analysis (EDWA): Given a fixed window size W and an MxN layout with K non-overlap rectangles, find a WxW density window which has the maximum (minimum) density.

[1, 3] proposed exact algorithms to address the EDWA problem. However, the running time is very long measured by hours or days. [2, 3] also proposed approximate algorithms such that the difference between the reported maximum density and the actual maximum density is within the given error bound. However, the algorithms are unable to report the exact solutions. Recently, [7] proposed a fast exact density calculation algorithm which shortens the running time from hours/days to seconds/minutes. [7] uses a recursive approach. At each iteration, the pruning technique is used to narrow down the regions that may contain the maximum density windows.

In this paper, we propose a simple two-level hierarchical approach to exactly identify the maximum density window for a given EDWA problem. (To simplify the presentation, we will focus on identifying the maximum density windows. The minimum density window can be handled in the similar way.) Comparing with [7], the new algorithm can report the exact maximum window density with equivalent or shorter running time. Especially for the long runtime test cases, our new algorithm shows big runtime reductions. For example, as shown in the experimental results, the runtime of the testcase Test5 with a window size $24,000$ is reduced from $23.7s$ to $2.8s$.

The paper is organized as follows. In Section 2, we briefly review the density theorems presented in the literature and propose a new theorem. All these theorems are used to prune regions so that the density calculation algorithm can efficiently focus on the regions that might contain the maximum density windows. Section 3 outlines the two-level hierarchical exact density calculation algorithm. In Section 4, a dynamic programming based approach is presented to efficiently calculate the window density for a given density map. And in Section 5, a hash-table based method is proposed to facilitate the exact density calculation for a given region. Then a discussion of the tradeoff on the sliding steps is presented in Section 6. Experimental results are given in Section 7 and Section 8 concludes the paper.

2. Theorems for Density Bound

In industry, most commercial tools use fix-dissection approach [4, 5, 6]. In this approach, a layout is partitioned into non-overlapping RxR tiles. Usually W is multiple times of R. Then only windows whose boundaries fall on the R-grid are checked for density. As pointed out in [7], this approach only checks a very limited number of windows. And it cannot produce the exact density numbers until R reaches the minimum feature size. However, the fix-dissection approach provides basic information on density distribution which can guide us to identify regions that might contain the maximum density windows.

The density bound theorems reveal the window density relationship between a window on the fix-dissection grid and any window within a certain region. (For convenience, a window on a fix-dissection grid is called sliding window.) These theorems play an important role in the density calculation algorithm such that they could be used to prune regions which do not contain the maximum density windows. In this section, we first review the theorems addressed in the literature [3] and [7]. Then one new theorem is proposed and proved.

Lemma 1 [7] For any window, it can be fully covered by four sliding windows on the fix-dissection grid.

As shown in Figure 1 (a), (b), (c) and (d), the red window can be fully covered by four sliding windows W_{LB}, W_{RB}, W_{LU} and W_{UB}. Suppose their densities are D_{LB}, D_{RB}, D_{LU} and D_{UB}, and let R be the grid size for the fix-dissection grid.

Theorem 1 [7] For any window Win, its density D_{Win} satisfies that $D_{Win} \leq D_{Cmax} + (\frac{R}{W} - (\frac{R}{2W})^2)$, where $D_{Cmax} = \max\{D_{LB}, D_{RB}, D_{LU}, D_{RU}\}$.

Meanwhile, it is easy to conclude that any window can be fully covered by a $(W+R)$x$(W+R)$ window which is shown as $Wout$ in Figure 1 (e).

Theorem 2 [3] For any window Win, its density D_{Win} satisfies that $D_{Win} \leq D_{Wout}$, where D_{Wout} is the total feature area in the window $Wout$ over W^2.

Furthermore, any window covers a $(W-R)$x$(W-R)$ window $Wcenter$ as illustrated in Figure 1 (f).

Theorem3 For any window Win, its density D_{Win} satisfies that $D_{Win} \leq D_{Wcenter} + (W^2 - (W-R)^2)/W^2$, where $D_{Wcenter}$ is the total feature area in the window $Wcenter$ over W^2.

The proof of Theorem 3 is straightforward. Any window inside $Wout$ covers $Wcenter$. The maximum feature area difference between $Wout$ and $Wcenter$ is $(W^2 - (W-R)^2)$. Since $D_{Wout} \geq D_{Win} \geq D_{Wcenter}$, we get $(W^2 - (W-R)^2)/W^2 \geq (D_{Wout} - D_{Wcenter}) \geq D_{Win} - D_{Wcenter}$.

3. Two-level Calculation Algorithm

In this section, we outline our two-level density calculation algorithm. The basic observation is that the fix-dissection approach is very fast. Also the memory in today's computers is quite large. Therefore, this motives us to start the first level with a fine-grid fix-dissection approach. Based on the fix-dissection results, the three theorems are applied to narrow down the interested regions which may potentially contain a maximum density window. In the second level, each selected region is checked to get the maximum window density. The algorithm is outlined as Two_Level_Density_Calculation. $COLS$ and $ROWS$ are the x and y dimensions of the given input, respectively.

In the following algorithm, Lines $1 \sim 3$ are to apply the fix-dissection approach, and get the maximum sliding window density from the fix-dissection grid. In the following section, we present an efficient algorithm (DCWG) to calculate the maximum sliding window density from a fix-dissection grid. Since Win, $Wcenter$ and $Wout$ are on the same grid, the values of D_{Win}, $D_{Wcenter}$ and D_{Wout} can be obtained in a single pass of DCWG. Lines $8 \sim 14$ are to loop on each $Wout$ to check if the three theorems are satisfied or not against $maxdens$. If all the three IF checks are passed, it means that the $Wout$ might contain a maximum density window, and Cal_Wout_Dens is applied to get the maximum window density in the $Wout$. Cal_Wout_Dens works on a $(W+R)$x$(W+R)$ region. By appropriately setting up the grid, DCDM algorithm can be used to get the exact maximum window density for the given $Wout$ region as well. The details of Cal_Wout_Dens are presented in Section 5.

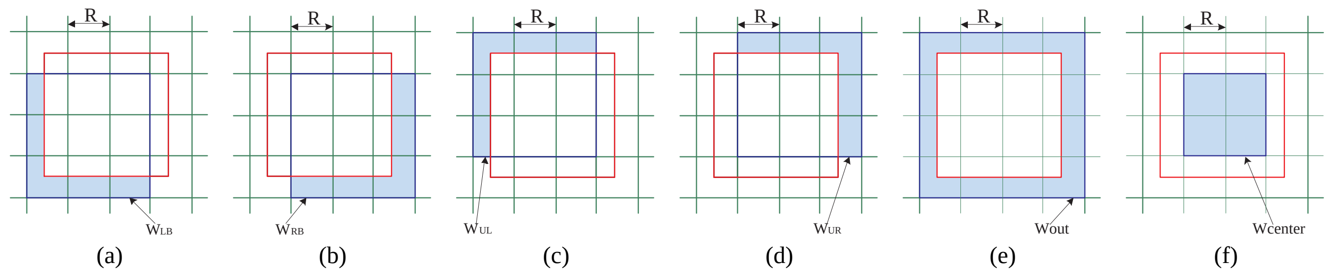

Figure 1: (a) Sliding window W_{LB} (b) Sliding window W_{RB} (c) Sliding window W_{LU} (d) Sliding window W_{RU} (e) A $(W+R)$x$(W+R)$ window $Wout$ (f) A $(W-R)$x$(W-R)$ window $Wcenter$

```
Algorithm Two_Level_Density_Calculation()
1.  Apply the fix-dissection approach with a fine grid size $R$;
2.  Get $D_{Win}$, $D_{Wcenter}$, $D_{Wout}$ for all sliding windows;
3.  maxdens=$\max\{D_{Win}$ for all sliding windows $\}$;
4.
5.  wdiff = $1-(W-R)^2/W^2$;
6.  sdiff = $\frac{1}{R}\cdot(1-\frac{1}{4R})$;
7.
8.  for(i=0; i $< COLS-W-1$; i++)
9.     for(j=0; j $< ROWS-W-1$; j++)
10.        if($D_{Wout}[i][j] >$ maxdens)
11.           if($D_{Wcenter}[i+1][j+1]$ + wdiff $>$ maxdens)
12.              maxlocal=$\max\{D_{Wout}[i][j], D_{Wout}[i][j+1],$
                            $D_{Wout}[i+1][j], D_{Wout}[i+1][j+1]\}$;
13.              if (maxlocal + sdiff $>$ maxdens)
14.                 maxdens = $\max\{maxdens, Cal\_Wout\_Dens($Wout[i][j]$)\}$;
15. return maxdens;
```

4. Density Calculation on W-Grid (DCWG)

In this section, we address the density calculation problem for a given W-grid. A W-grid is defined as follows.

W-Grid Given a window size W, if an MxN grid satisfies the constraint that if one of the window corners is on the grid, then its other three corners must be on the grid except that the window covers regions that are outside the grid, then such a grid is called W-Grid.

Density Calculation on W-Grid (DCWG) Given a W-Grid, Assume that the feature area of each grid tile is given, find the maximum density for windows which are on the given W-Grid.

As we notice that the grid tile sizes of a W-Grid can be different. For convenience, we still call the windows on the W-Grid as sliding windows. Figure 2 shows an example. Figure 2 (a) is a 8x7 W-Grid. For any window, if its left-bottom corner is on the grid, then its right-upper corner must be on the grid as well. On the other hand, Figure 2 (b) is not a W-Grid. The right-upper corner of the window is not on the grid.

Obviously, the fix-dissection density calculation is just a special case of DCWG such that the tile sizes of the W-Grid are the same.

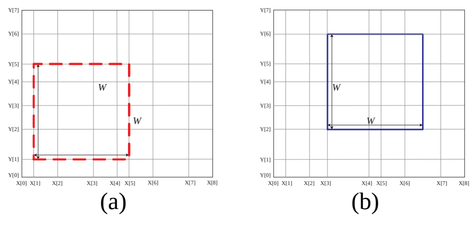

Figure 2: (a) An 8x7 W-Grid. (b) An invalid 8x7 W-Grid.

Once the area map of a W-Grid is given, our target is to find the maximum density of the sliding windows on the W-Grid. In this section, we propose a dynamic programming based approach to solve the DCWG problem.

Let $X[i]$ $(i=0,\ldots,m)$ and $Y[j]$ $(j=0,\ldots,n)$ be the coordinates of the i^{th} vertical grid and j^{th} horizontal grid, separately. According to the W-Grid property, we get that for any i, if $X[i]+W \leq X[m]$, there must exist i_w such that $X[i]+W = X[i_w]$. Similar for $Y[j]$.

Since our target is to get the density for on-grid windows, we calculate the density for windows from bottom to top, and from left to right. To get the density of the window which is on the left-bottom corner of the given grid, we need to sum the feature areas of all the tiles covered by the window. As shown in Figure 3 (a), suppose $A[i][j]$ is the feature area of tile[i][j]. The density of the blue window is $\sum A[k][p]/W^2$ $(k=0,\ldots,2; p=0,\ldots,3)$. Similarly, the density of the red window in Figure 3 (b) is $\sum A[k][q]/W^2$ $(k=0,\ldots,2; q=1,\ldots,4)$. It is easy to see that the two adjacent windows share the tiles $A[k][l]$ $(k=0,\ldots,2; l=1,\ldots,3)$. Therefore, if we know the density of the blue window, we only need to subtract the bottom row in the blue window and add the upper row in the red window. This motivates us to develop a dynamic programming approach to get the window density on a W-Grid. The main idea is summarized in the algorithm DCWG_Window_Density. $COLS$ and $ROWS$ are the x and y dimensions of the given input, respectively.

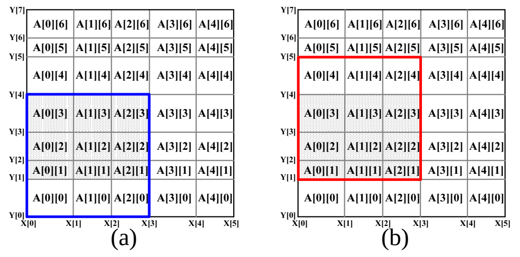

Figure 3: Two neighbor windows share the shadow region.

Figure 4 shows the flow of the algorithm. In Figure 4 (a), rowarea is calculated from bottom to top as Line 1 $\sim$ 2. Line 3 is to get the total feature area of the first window by adding rowarea as illustrated in Figure 4 (b). The second window gets its feature area by adding rowarea[3] and subtracting rowarea[0]. This process continues until all the windows on the first column are processed (Line 5 $\sim$ 9). Then move to the next column. Each rowarea is updated by one sum operation and one subtract operation as Line 12 $\sim$ 14. Once rowarea update is done, the window feature area can be calculated from bottom to top (Line 16 $\sim$ 21). maxarea records the maximum feature area for all sliding windows on W-Grid, and maxarea/W^2 is the maximum window density for the given W-Grid.

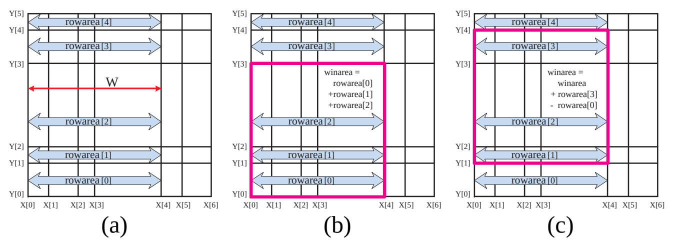
rowarea[4]
rowarea[3]
W
rowarea[2]
rowarea[1]
rowarea[0]
winarea = rowarea[0] +rowarea[1] +rowarea[2]
winarea = winarea + rowarea[3] - rowarea[0]
(a)
(b)
(c)

Figure 4: (a) rowdens is calculated for each row. (b) The first window on the left bottom corner is calculated. (c) Calculate the total feature area of the windows along the first column.

In the DCWG_Window_Density algorithm, the calculations are to get the values of rowarea and winarea. To calculate rowarea, each $A[i][j]$ involves at most one add and one subtract operations. Therefore, rowarea calculation time can be bounded by $O(M \cdot N)$, where M and N are the number of columns and rows, respectively. winarea are derived from rowarea. And each rowarea gets at most one add and one subtract operations in order to calculate the values for winarea. Totally there are at most $M \cdot N$ rowarea. So the calculation time for winarea can be bounded by $O(M \cdot N)$. Finally, there are at most $M \cdot N$ sliding windows and maxarea calculation can be bounded by $M \cdot N$ as well. Based on the above analysis, the runtime of DCGW_Window_Density algorithm can be tightly bounded by $O(M \cdot N)$.

```
Algorithm DCWG_Window_Density($COLS$, $ROWS$, $W$, $A$)
1.     for(j=0; j < ROWS; j++)
2.       rowarea[j] = $\sum_{i=0}^{p-1} A[i][j]$, where $X[p] = X[0] + W$;
3.     winarea = $\sum_{i=0}^{q-1} rowarea[i]$, where $Y[q] = Y[0] + W$;
4.
5.     maxarea = winarea;
6.     for(j=1; q < ROWS; j++, q++)
7.       winarea += rowarea[q];
8.       winarea -= rowarea[j-1];
9.       maxarea = MAX(winarea, maxarea);
10.
11.  for(i=1; p < COLS; i++, p++)
12.    for(j=0; j < ROWS; j++)
13.      rowarea[j] += A[p][j];
14.      rowarea[j] -= A[i-1][j];
15.
16.    winarea = $\sum_{k=0}^{q-1} rowarea[k]$, where $Y[q] = Y[0] + W$;
17.    maxarea = MAX(winarea, maxarea);
18.    for(j=1; q < ROWS; j++, q++)
19.      winarea += rowarea[q];
20.      winarea -= rowarea[j-1];
21.      maxarea = MAX(winarea, maxarea);
22.  return maxarea/$W^2$;
```

Finally, since the fix-dissection approach is a special case of DCGW with all grid tiles having the same size, the fix-dissection approach can get the maximum density of all sliding windows in $O(M \cdot N)$.

5. Exact Calculation on a *Wout* region

In Two_Level_Density_Calculation, the first level is to apply the fix-dissection approach on the whole layout. The maximum density of the sliding windows can be efficiently derived with DCWG algorithm. Then a certain amount of *Wout* regions are pruned based on the three theorems. For the rest of the *Wout* regions, Cal_Wout_Dens is called to find the exact maximum window density for each *Wout* region.

In [7], a theorem is presented that helps to identify an exact maximum density window.

Theorem 4 [7] Given a region with k rectangles, there exists a maximum density window that has two adjacent window edges overlap two rectangle edges. Furthermore, the overlapped window edges and rectangle edges are in the same directions. (The edge directions of windows/rectangles are defined as the clockwise direction.)

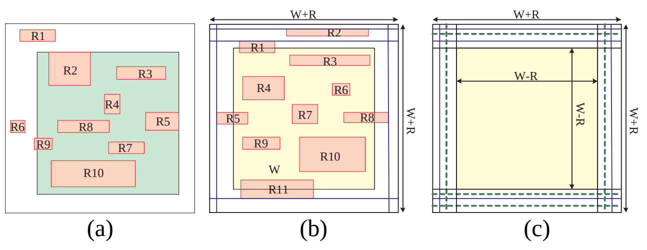

Figure 5: (a) A maximum density window has its upper edge and right edge overlapping with the upper edge of $R2$ and the right edge of $R5$, respectively. (b) A *Wout* window with 11 rectangles. (c) The constructed grid for the given *Wout* window.

For a given region, the maximum density window may not be unique. But according to Theorem 4, we can always find a maximum density window whose two adjacent edges are on two rectangles. For example, in Fig 5 (a), the window is a maximum density window. The upper window edge overlaps with the upper edge of $R2$ and the right edge of $R5$ is on the right window edge. Therefore, we can narrow down the searching space to only consider the windows that satisfies the constraints in Theorem.

In Cal_Wout_Dens, we first construct a grid on the given *Wout* region based on the above theorem such that at least one maximum density window is on the grid. We further show that the constructed grid is a W-Grid. Therefore, the DCWG algorithm can be applied on the grid to get the maximum window density efficiently.

According to Theorem 4, we only need to focus on windows whose edges have overlap with rectangles. Furthermore, since the input is a *Wout*, which is a $(W+R)$x$(W+R)$ Region, any WxW windows inside the given region share the center $(W-R)$x$(W-R)$ part, i.e., $Wcenter$. Therefore, we only need to consider the rectangle edges outside $Wcenter$. Figure 5 shows an example. Figure 5 (b) is a *Wout* region with 11 rectangles. The blue solid lines are generated for the rectangles outside $Wcenter$. Furthermore, a dashed green line is created for each solid blue line such that the distance between the two lines is W. In this way, it guarantees that the four edges of a window can be on the constructed grid. Figure 5 (c) shows the constructed grid. It is easy to conclude that the grid in Figure 5 (c) is a W-Grid.

To complete the grid construction, an important step is to find the x-coordinate (y-coordinate) of each vertical (horizontal) grid line. In this section, we propose a hash table based method to efficiently achieve this goal. To simplify the presentation, we only show how to calculate the y-coordinate of each horizontal grid line.

Since the input is a *Wout* region, its size is $(W+R)$x$(W+R)$. Furthermore, we know that the center region $Wcenter$ is shared by all windows inside $Wout$. Therefore, only y-coordinates outside $Wcenter$ need to be considered. In this case, two hash tables with $R+1$ entries are enough to hold all y-coordinates. The first hash table corresponds to the y-coordinates within $[Wout.y_l, Wout.y_l + R]$, and the second one records the y-coordinates $[Wout.y_r - R, Wout.y_r]$. The hash table includes two items: coord and index. coord is used to record the coordinates of rectangle edges, while index is used to speed building the density map. The coord field of the two hash tables is initially assigned a value that is lower than $Wout.y_l$.

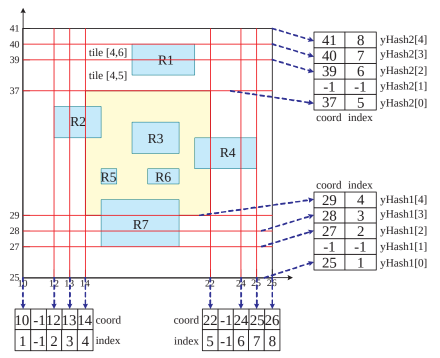

Figure 6: Two hash tables are constructed in order to identify the x/y-coordinates of the vertical/horizontal grid lines, respectively.

Figure 6 gives an example. The window size W is 12, and R is 4. *Wout* is bounded by $[(10,25)(26,41)]$, i.e., $Wout.x_l = 10$, $Wout.y_l = 25$, $Wout.x_h = 26$ and $Wout.y_h = 41$. The bottom edge of the rectangle $R7$ falls in the range $[Wout.y_l, Wout.y_l + R]$. A horizontal edge should be created. Therefore, $yHash1[27-25].coord$ is marked as 27. At the same time, another horizontal line with the y-coordinate 39 should be created as shown by a dashed green line. Accordingly, $yHash2[39-(41-R)].coord$ is marked as 39. Similarly, the upper edge of the rectangle $R1$ also corresponds to two horizontal grid lines, and we get $yHash2[3].coord = 40$ and $yHash1[3].coord = 29$. By traversing all the related rectangles once, the coord field of the two hash tables is done. Then going through the two hash tables from bottom to top to determine the index field. For example, the index of $yHash1[3]$ is 3 since it is the third horizontal line from the bottom.

Once the grid is constructed as shown in Figure 6, the next step is to calculate the feature area for each grid tile. Obviously, the grid tile $Wcenter$ could get its feature area from the previous fix-dissection calculation. For the rest of the tiles, we need to traverse all the related rectangles and find their overlap with the grid tiles. With the help of the index field, this step can be easily accomplished. For example, for the rectangle $R1$, the y-coordinate of its upper edge is 40. Since yHash2[40-(41-R)].index = 7, it means that the rectangle is below the 7th horizontal line. Meanwhile, by checking the left and right edges of $R1$, we know that the rectangle falls between the 4th and 5th vertical lines. Therefore, we only need to calculate the overlap areas between $R1$ and tile[4,6] and tile[4,5].

Since the constructed grid for the given *Wout* is a W-Grid, the DCWG algorithm can be applied to get the maximum window density. The Cal_Wout_Dens algorithm can be summarized as follows.

```
Algorithm Cal_Wout_Dens(Wout)
1. Construct the W-Grid with the Hash table structure;
2. Calculate the feature area for each grid tile;
3. Apply the DCWG algorithm;
4. return the maximum window density;
```

In Cal_Wout_Dens, by traversing all the related rectangles once, the W-Grid can be constructed. Then another traverse of all the related rectangles can finish step two. The runtime of Step three is bounded by $O(M \cdot N)$ where M and N are the number of rows and columns, respectively. Therefore, the runtime of Cal_Wout_Dens is $O(M \cdot N + K)$ where K is the number of the related rectangles.

6. Tradeoff

In Two_Level_Density_Calculation, the first level is to apply the fix-dissection approach on the whole layout with a fine sliding step, and the second level is to calculation the exact maximum window density on each identified *Wout* region from the first stage.

As we notice that, for some test cases, after the pruning step, there are still many *Wout* regions. One reason is that some high density regions cover a relative large space. Since the sliding step is small, the high density regions may generate many *Wout* as shown in Figure 7. In Figure 7 (a), the white rectangle is an identified *Wout* region in the first stage. Shifting the rectangle left/right a little bit, we can still get high density regions as shown by the purple rectangle and the green rectangle. In this case, the two neighboring *Wouts* share a large portion of the area and it is not efficient to calculate those *Wouts* one by one.

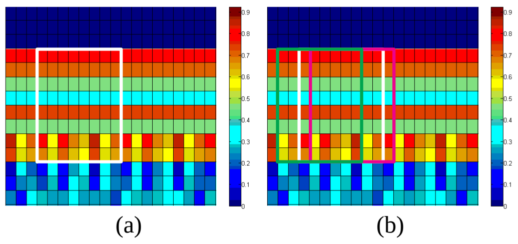

Figure 7: (a) The white rectangle outlines an identified *Wout* region from the first stage. (b) The neighboring *Wout* regions such as the regions outlined by the purple rectangle and the green rectangle also have a high density.

Based on the above consideration, we introduce a *Wout* threshold to control the number of the identified *Wout* regions. After applying the first round of fix-dissection, if the number of identified *Wout* regions is larger than the threshold, then redo the fix-dissection step with a large sliding step, and the rest of the steps are applied on the *Wout* regions from the second fix-dissection. Of course, in this case, the *Wout* regions are larger and it may take longer time to calculate the exact maximum window density. However, since the number of *Wout* regions is greatly reduced, the short runtime can be guaranteed. The Two_Level_Density_Calculation algorithm in Section 3 is modified as follows. The first step is extended into 1.1 $\sim$ 1.5.

```
Algorithm Two_Level_Density_Calculation()
1.1  Apply the fix-dissection approach with a fine grid size $R$;
1.2  Apply the three theorems to prune regions;
1.3  If the number of the identified $Wout > Threshold$
1.4      Redo the fix-dissection with a large grid size $R'$;
1.5      $R = R'$;
2.       ...
```

7. Experimental Results

We implemented our algorithm in C on a linux machine (3.2GHz) with 3.5GB memory. We use the same test set as [7]. Two window sizes $24,000nm$ and $32,000nm$ are used for testing, separately.

The algorithm starts the fix-dissection approach with a fine sliding step $W/100$. By applying the three pruning theorems, a lot of *Wout* regions are discarded. Table 1 shows the effectiveness of the pruning technique. In this paper, a new pruning theorem (Theorem 3) is proposed. "Theorem 1-2" refers to the number of *Wout* regions after applying only Theorem 1 and 2. While "Theorem 1-3" shows the number of *Wout* after applying the three theorems. It is clear that Theorem 3 also helps a lot in pruning regions. Especially, for Test4, the number of *Wout* is reduced from $175,151$ to $25,831$ with Theorem 3.

Table 1: Number of *Wout* after applying pruning theorems

Testcase	$W = 24,000$		
	Total *Wout*	Theorem 1-2	Theorem 1-3
Test1	5,294,601	9,147 (0.17%)	6,270 (0.12%)
Test2	5,294,601	13,107 (0.25%)	11,567 (0.22%)
Test3	4,137,156	1,351 (0.03%)	1,351 (0.03%)
Test4	25,335,557	175,151 (0.69%)	25,831 (0.10%)
Test5	4,137,156	180 (0.00%)	180 (0.00%)
Test6	16,273,156	94 (0.00%)	42 (0.00%)
Test7	16,273,156	3,953 (0.02%)	3,741 (0.02%)
Test8	16,273,156	1,364 (0.01%)	1364 (0.01%)
Test9	24,671,089	1,396 (0.00%)	562 (0.00%)
Test10	16,273,156	527 (0.00%)	527 (0.00%)

Table 2 shows the runtime comparison between the algorithm in [7] and the proposed two-level algorithm. The first stage sliding step is set as $W/100$, and the *Wout* threshold is 5000. If the number of identified *Wout* regions is larger than 5000, then the sliding step is set as $W/8$. Comparing to [7], for testcases which have a short runtime with [7], our algorithm also achieves equivalent runtime. For testcases which have a long runtime with [7], the proposed two-level algorithm shows advantages on the runtime. For example, for Test5, the runtime is reduced by more than $20s$. For these 10 test cases, on average, our proposed algorithm shorts the runtime by $4.5s$ with the window size $24,000$ and $6.7s$ with the window size $32,000$.

Table 2: Runtime Comparison

Testcase	$W = 24,000$			$W = 32,000$		
	[7](s)	Ours(s)	Diff(s)	[7](s)	Ours(s)	Diff(s)
Test1	0.553	1.849	+1.296	0.601	1.115	0.514
Test2	1.518	3.371	+1.853	1.832	1.627	-0.205
Test3	12.302	8.042	-4.260	15.727	19.361	+3.634
Test4	2.275	5.592	+3.317	1.586	3.608	+2.022
Test5	23.725	2.841	-20.884	42.788	16.851	-25.937
Test6	12.420	8.515	-3.905	13.349	7.199	-6.150
Test7	35.509	25.835	-9.674	61.886	42.269	-19.617
Test8	29.159	18.267	-10.892	27.554	13.427	-14.127
Test9	7.737	11.550	+3.813	7.757	9.863	2.106
Test10	20.225	13.887	-6.338	24.745	14.961	-9.784
Ave	-	-	-4.567	-	-	-6.754

8. Conclusion

In this paper, we propose a simple but efficient two-level hierarchical approach to exactly identify the maximum density window for a given EDWA problem. Comparing with the latest work [7], the new algorithm can report the exact maximum window density with equivalent or even shorter running time. Especially for the long runtime test cases, our new algorithm shows big runtime reductions.

9. References

[1] A. B. Kahng, G. Robins, A. Singh, H. Wang and A. Zelikovsky, Filling and Slotting: Analysis and Algorithm, ISPD, 1998.

[2] A. B. Kahng, G. Robins, A. Singh and A. Zelikovsky, New Multilevel and Hierarchical Algorithms for Layout Density Control, Proc. Asia and South Pacific Design Automation Conf., pp. 221-224, Jan. 1999.

[3] A. B. Kahng, G. Robins, A. Singh and A. Zelikovsky, Filling Algorithms and Analyses for Layout Density Control, IEEE Transactions on Computer-Aided Design 18(4), pp. 445-462, 1999.

[4] Y. Chen, A. B. Kahng, G. Robins, and A. Zelikovsky, Monte-Carlo Algorithms for Layout Density Control, Proc. Asia and South Pacific Design Automation Conf., pp. 523-528, Jan. 2000.

[5] R. Tian, D. F. Wong, R. Boone, Model-based Dummy Feature Placement for Oxide Chemical-Mechanical Polishing Manufacturability, Proc. Design Automation Conf, pp 667-670, 2000.

[6] X. Wang, C. C. Chiang, J. Kawa and Q. Su, A Min-Variance Iterative Method for Fast Smart Dummy Feature Density Assignment in Chemical-Mechanical Polishing, ISQED, pp. 258-263, 2005.

[7] H. Xiang, K. Chao, R. Puri and M. D. F. Wong, Is your layout density verification exact? – a fast exact algorithm for density calculation. Proc. ACM/IEEE Intl. Symp. on Physical Design, pp 19-26, March 2007.