An Efficient Character Generation Algorithm for High-Throughput E-Beam Lithography

Shih-Ting Lin[1], Hong-Yan Su[1], Oscar Chen[2], and Yih-Lang Li[1]
Department of Computer Science, National Chiao Tung University[1]
AnaGlobe Technology, Inc.[2]

ABSTRACT

E-beam lithography has been one of promising next generation lithography for 7*nm* and below technology nodes. Among various electron-beam lithography features, character projection (CP) attracts users because complex patterns can be printed in one e-beam shot. However, we still face severe challenges of generating characters on interconnection layers due to its pattern diversity. In this paper, we proposes a multi-intersection-level (MIL) layout that can efficiently capture the relationships between nearby objects including the spacing between them. The inflated layer reduces the problem instance size for identifying the frequently used patterns while the intersection layers help in clipping windows to obtain ideal character set. Experimental results show that the proposed methodology can efficiently yield the frequently used character set with up to 93.3% and 81.23% covering rate in via layer and metal layer. Besides, for a panel layout, a set of frequently used characters to reach 100% covering rate is successfully identified.

1. INTRODUCTION

As one of the promising next generation lithography, E-beam lithography (EBL) has the advantage on the fabrication of sub-nanometer technology nodes because it can easily focuses on nanometer diameter. Another E-beam application is for the manufacturing of retina-resolution display panel, which has higher pixel density than traditional ones. However, the drawback of low throughput makes EBL hard to be used on mass production, and several techniques such as variable shaped beam (VSB) or character projection (CP) are employed to improve throughput of E-beam direct write (EBDW). Among these techniques, CP is widely adopted because complex shapes can be printed in one E-beam shot with the provided templates, called *character*. Hence, the number of required E-beam shots can be reduced in orders of magnitude and improve throughput drastically.

To identify frequently used patterns (FUPs) in a layout as characters can increase the benefit of CP-EBL and minimize the number of required E-beam shots. For example, Takeshi Fujino *et al* convert complex standard cells to be sets of basic standard cells, each of which becomes a character [1]. And then the majority components of a standard cell based design can be fabricated with CP-EBL. However, only few works discuss how to investigate FUPs on interconnect layers of a layout because of the wide diversity of pattern shapes on these routing layers, and most of the works focus on packing characters in a stencil [2]-[4]. Fig. 1 illustrates an example of challenges caused by pattern diversity. Fig. 1(a) shows a layout for identifying corresponding characters for CP-EBL. One intuitive method to investigate FUPs is placing the character window of desired pattern size on anywhere of the layout, and we can collect all possible patterns according to every located window. After examining all possible patterns and their repetition numbers, we can conclude a set of desired patterns to improve the throughput of EBDW. Examples in Fig. 1(b) and 1(c) show that, as we change the

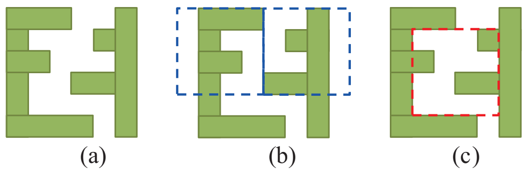
Fig. 1. Pattern locations is the critical challenge of investigating frequently used patterns. (a) Original layout; (b) one possible pattern can find two occurrences; (c) another patterns with the same size has only one occurrence.

location of character window, the identified patterns and their repetition numbers also change as well. Since each polygon in a layout only needs to be written once, the EBDW cannot be conducted at the same location twice. The intersection of the polygons in any two patterns used by EBDW should be empty. Apparently, it is infeasible to enumerate all possible patterns in a layout; furthermore, most of the patterns may not be a repetitive one such as Fig. 1(c).

Because of the challenge of pattern diversity during investigating FUPs in a layout, previous works have different limitations on their proposed methodologies. The assumption is made that every wire in a layout has the same direction in [5][6]. This limitation is adopted in the advanced technology for one-dimensional (1D) IC layout, but the panel layouts are not limited to 1D layouts. Via patterns in [5] cannot contain more than three vias theoretically, and via patterns are one-dimensional arrays in [6]. These limitations not only diminish the diversity of pattern shapes but also restrict the freedom of generating characters for two-dimensional layouts. Masahiro Shoji, *et al* focus on the mask patterns after OPC, and they fracture each polygon into several small rectangles [7]. Adjacent rectangles are then re-grouped to form a character. Efficiency issue caused by complex rectangle combinations may hinder the application on interconnection layers. Rimon Ikeno *et al* focus on generating characters for non-Manhattan polygons such that the numbers of trapezoids using VSB can be reduced while still improving line-edge quality [8].

In this work, we propose a multi-intersection-level (MIL) layout that can efficiently capture the relationships between nearby objects including the spacing between them. Each polygon is inflated with a fixed value first. After that, all inflated polygons are consolidated together by performing a geometrical OR operation on all inflated polygons. Meanwhile, intersection operations are performed on all inflated polygons such that the intersection result of any *i* polygons is stored in the *i*-th level of MIL layout. With this scheme, exact pattern matching can be accelerated on the MIL layout due to the reduced problem instance size, and the FUPs also can be obtained. Meanwhile, the MIL layout keeps the regions where most inflated polygons overlap. If the FUPs is larger than the user-defined character window, the regions where most inflated polygons overlap are the ideal locations for the centers of clipping window. The frequently used characters (FUCs) can then be calibrated, and the repetitive locations for each FUC can be identified through conducting an exact matching on the entire layout. Experimental results indicate that the generated characters can achieve 100% covering rate

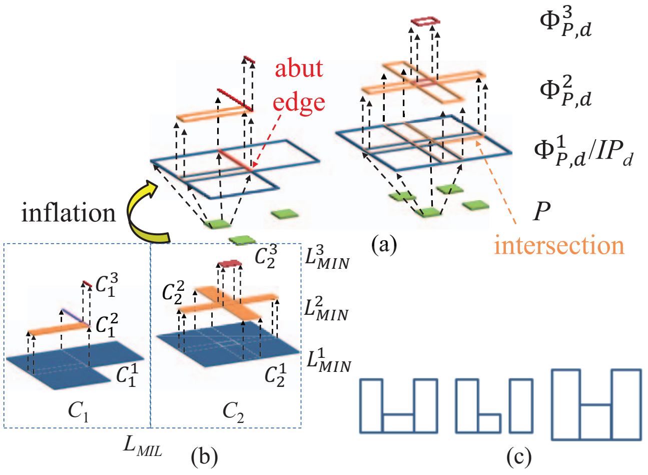

Fig. 2. MIL layout construction example. (a) Polygon inflation and rectangle intersection; (b) resultant L_{MIL}; (c) polygon inflation cannot yield one-to-one mapping.

for a panel layout, and up to 99.7% of the layout can be printed with three generated characters. As for the interconnection layers, we can have 93.3% and 81.23% covering rates on via and metal layers, respectively.

The remainder of this paper is organized as follows. Section 2 describes the problem formulation. Section 3 describes the concept and associated properties of the proposed MIL layout. Then, we detail the overall pattern generation algorithm with the proposed MIL layout in section 4. In section 5, we present the experimental results, and finally, we draw conclusions in section 6.

2. PROBLEM FORMULATION

In previous works, character generation does not aim at identifying a set of patterns, with which the entire layout can be printed. They only try to identify a set of non-overlapping FUPs. In this work, we follow the problem used in previous works.

Character generation problem: Given a layout, a maximum allowable window size W, and the stencil area for placing characters, the objective of character generation problem for CP-EBL is to find a set of non-overlapping FUPs fitting W to decrease the number of required E-beam shots. The remainder of the layout that is not contained by FUPs is printed through VSB.

3. MIL-LAYOUT CONSTRUCTION

All previous exact matching algorithms work on original polygons, which lacks the spacing information between vicinal polygons. The fundamental concept of MIL layout is to inflate all polygons with a fixed value such that vicinal polygons can be merged as a larger polygon with the spaces among them as hidden information. However polygon inflation cannot yield a one-to-one mapping between the original pattern and the inflated one. For instance, the inflation with a value to the left pattern (one U-shaped polygon) and the middle pattern (one L-shaped polygon and one rectangle) in Fig. 2(c) yields the same outcome (the right polygon in Fig. 2(c)). To avoid this confusion, each polygon is split into several non-overlapping rectangles, and the inflation of a polygon becomes an inflation to its all rectangles. The first-level of a MIL layout is the union of all inflated rectangles of all polygons. From the second level to the end, the resultant layout at level i is the intersection of any i inflated rectangles. With the match on the union layout at level 1 together with matches on the intersection layouts at the other levels in a MIL layout, one-to-one mapping between the original pattern and the inflated one can be confirmed.

Algorithm 1. MIL_Layout_Construction

Input: A set of disjoint polygons P and inflation size d.
Output: Corresponding MIL layout L_{MIL}.

```
1. $IP_d \leftarrow \forall p_j \in P$ inflate with $d$.
2. $i \leftarrow 1$ and $\Phi^1_{P,d} = IP_d$.
3. do
4.     $L^i_{MIN} \leftarrow \cup\Phi^i_{P,d}$ & $\Phi^{i+1}_{P,d} \leftarrow \cap\Phi^i_{P,d}$.
5.     $i \leftarrow i + 1$.
6. while ($\Phi^i_{P,d}$ is not empty & $L^i_{MIN}$ != $L^{i-1}_{MIN}$ )
```

Fig. 3. Algorithm for MIL layout construction

Given a polygon p, let $R_p = \{r_{p1}, r_{p2}, \ldots , r_{pk}\}$ be the rectangle list obtained through splitting p into k non-overlap rectangles. Then, polygon inflation in this work is defined as follows.

Definition 1. Given a polygon p and a desired inflation size d, the inflation of p is $IR_{p,d} = \{ ir_{p1}, ir_{p2}, \ldots , ir_{pk} \}$, where ir_{pi} is the inflation of r_{pi} with d, $\forall\ r_{pi} \in R_p$.

Definition 2. Let P be a pattern and $P = \{p_1, p_2, \ldots , p_n\}$ be a set of disjoint polygons in a layout. Then $IP_d = \{ IR_{p1,d}, IR_{p2,d}, \ldots , IR_{pn,d}\}$ is the polygons after inflation with a desired inflation size d.

Definition 3. A *component* can be a rectangle, a line, or a point.

Definition 4. $\Phi^i_{P,d} = \cap\Phi^{i-1}_{P,d}$, $\forall\ i > 1$, where $\cap\Phi^{i-1}_{P,d}$ is the outcome of applying an intersection operation on all pairs of components in $\Phi^{i-1}_{P,d}$ and $\Phi^1_{P,d} = IP_d$. Then, the MIL layout $L_{MIL} = \{L^i_{MIN} = \cup\Phi^i_{P,d}$, $1 \le i \le n\}$, where $\cup\Phi^i_{P,d}$ merges all rectangles in $\Phi^i_{P,d}$ into polygons and n is the last level to have non-empty $\Phi^n_{P,d}$.

In this work, the intersection of two rectangles abutting along an edge of their boundary yields a line while that of two rectangles abutting at one of their corners yields a point. The intersection of a line and a rectangle may yield a line. For instance, the pattern in Fig. 2(a) has seven rectangles, and two of their inflated rectangles abut along one edge of their boundary. Fig. 2(b) shows the corresponding L_{MIL} of Fig. 2(a). Next, the equivalence of two MIL layouts is defined as following.

Definition 5. Given L_{MIL} and L'_{MIL}, $L_{MIL} = L'_{MIL}$ if $L^i_{MIN} = L'^{\,i}_{MIN}$, $\forall$ $i > 0$ and $|L_{MIL}| = |L'_{MIL}|$.

Fig. 3 shows the algorithm of the MIL layout construction. In the first step, we inflate all the polygons in P with size d and set $\Phi^1_{P,d}$ to be IP_d (lines 1 ~ 2). Then, we iteratively apply intersection and union operations on $\Phi^i_{P,d}$ to obtain $\Phi^{i+1}_{P,d}$ and L^i_{MIN}, respectively, until $\Phi^{i+1}_{P,d}$ becomes empty or $L^i_{MIN} = L^{i-1}_{MIN}$ (lines 3 ~ 7).

Proposition 1. *Given P, d, and associated L_{MIL}, L^i_{MIN} defines the regions where at least i rectangles in $\Phi^1_{P,d}$ overlap.*

Definition 6. Let $L^1_{MIN} = \{p^1_1, \ldots, p^1_m\}$, where p^1_i is the i-th polygon in L^1_{MIN}. As we project the boundary of each p^1_i onto the other levels (from level 2 to level $|L_{MIL}|$), the components in each level are also clustered into m groups. All components in L_{MIL} are thus clustered into m groups, i.e., $L_{MIL} = \{C_1, C_2, \ldots , C_m\}$, based on the boundary projection from L^1_{MIN}, where C_i, named *MIL component*, is the collection of polygon p^1_i and all the components in the other levels that are within the regions defined by the boundary projection of p_i^1 onto the other levels.

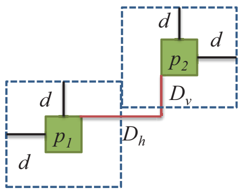

Fig. 4. Polygon spacing and inflate size.

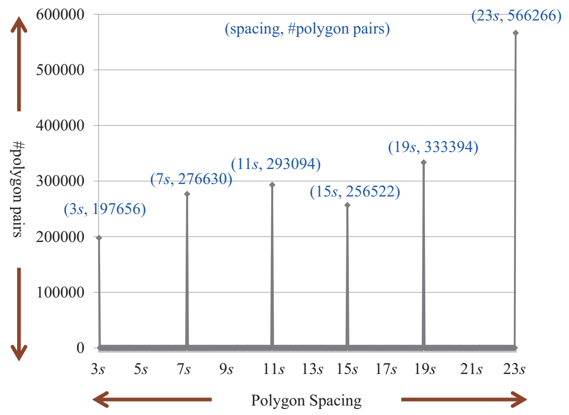

Fig. 5. Polygon spacing distribution.

Definition 7. C_i^j represents the components of C_i in L_{MIN}^j. Similarly, $|C_i|$ is the largest j that C_i^j is not empty.

For example, the components of a 3-level L_{MIL} in Fig. 3(b) are classified into two groups, C_1 and C_2. In C_1, C_1^1 has an L-shape polygon while C_1^2 contains a rectangle and a line. Meanwhile, C_2^1 has a rectangle while C_2^2 contains a cross shape.

It is worth noting that a polygon and the same polygon in another orientations may have different combinations of rectangle lists after slicing a polygon into a set of rectangles. The resultant MIL layouts of a polygon and its rotated one are thus also different. We have to generate all possible rotating orientations, i.e. 0°, 90°, 180°, 270°, for a MIL layout to achieve equivalent verification. If L_{MIL} in any orientation is equivalent to L'_{MIL}, $L_{MIL} = L'_{MIL}$.

3.1 Inflation Size Determination

As inflation size changes, the MIL layout may also change. If a MIL layout is sensitive to the change of inflation size, the quality of proposed methodology may not be stable. Given two polygons p_1 and p_2, the distance between them is defined as a minimum inflation size to make the merging of p_1 and p_2 happen (Fig. 4). In Fig. 4, the minimum horizontal and vertical spacing between p_1 and p_2 are D_h and D_v, respectively.

Definition 8. The distance between two polygons p_1 and p_2 is $max(D_h, D_v)$, where D_h and D_v are the minimum horizontal and vertical distance between p_1 and p_2. The minimum inflation size required to merge p_1 and p_2 is then $max(D_h, D_v) / 2$.

Fig. 5 shows the statistics of polygon spacing of an industrial layout with about 400,000 vias. Spaces larger than 23s are not shown in Fig. 5, where s is the minimum metal spacing. The observation from Fig. 5 implies that several peaks appear in this statistics, and every peak represents the numbers of polygons that are to be merged if the inflation size is set as the half of the corresponding space. For example, 276630 pairs of polygons have the spacing of 7s. And then 276630 pairs of polygons are going to be merged in the associated MIL layout if the inflation size is set to $7s / 2 = 3.5s$. Except these peak numbers, there are few number of polygon pairs with the other spacing values. Based on this observation, given a desired window size W, we can identify these peaks from polygon spacing distribution.

Table I. Numbers of components of every level with respect to inflation sizes

d	Lev 1	Lev 2	Lev 3	Lev 4	Lev 5	Lev 6
1.5*s*	**348587**	**98828**	**0**	**0**	**0**	**0**
2*s*	301781	95558	1687	0	0	0
2.5*s*	301610	95708	1706	0	0	0
3*s*	301435	95867	1719	0	0	0
3.5*s*	**222704**	**66445**	**1603**	**1962**	**0**	**0**
4*s*	209268	144411	43369	1859	16	0
4.5*s*	209122	144543	43392	1871	17	0
5*s*	208922	144588	43422	1887	17	0
5.5*s*	**150274**	**95781**	**26742**	**1675**	**15**	**35**

Table I shows how inflation sizes affects the number of components of a MIL layout. In this table, we can observe that the number of components of a MIL layout does not have a major change as the inflation size increases; on the contrary, it has little change for several consecutive inflation sizes and then has a big change at some points, i.e., 1.5s, 3.5s, and 5.5s. The points having big change conform to the peaks in Fig. 5 (3s, 7s, and 11s). The reason behind this phenomenon is that vias are located on routing pitches, implying that the spacing between vias is close to multiples of minimum metal spacing. As a result, the candidates of inflation size are discrete values at the peaks of polygon spacing distribution figure (Fig. 5) rather than continuous values.

4. CHARACTER GENERATION

In this section, we propose a methodology for FUP identification in via layers. All vias in a layer have identical shape but the complexity of via number is high. The MIL layout for a via layer is beneficial to diminish the problem complexity since a polygon in L_{MIN}^1 generally contains several vias and the space among them, which implies to lower the problem complexity as solving the problem in L_{MIN}^1 rather than original via set. Thus identifying FUPs in a via layer is equivalent to finding frequently used polygons (FUPGs) in L_{MIN}^1. For identifying FUPGs in L_{MIN}^1, a one-to-one mapping from a pattern in a via-layer layout to a polygon in L_{MIN}^1 is necessary.

Lemma 1. *Given two sets of via patterns, R and R', containing non-overlapping rectangles of equal size with a desired inflation size d, the associated MIL layouts* $L_{MIL} = L'_{MIL}$ *iff R = R'.*

Proof. Obviously, if $R = R'$, $L_{MIL} = L'_{MIL}$. Next, we are going to show that if $L_{MIL} = L'_{MIL}$, $R = R'$.

Since all inflated rectangles have the same dimension in width and length, say $w \times w$, we can derive the inflated rectangles from L_{MIN}^2 and L_{MIN}^1. The intersection of any two inflated rectangles could be a rectangle, a line or a point. If the intersection outcome is a rectangle r_s, two cases are considered. The first case is that r_s has the dimension of w in width or length, and then two inflated rectangles can be derived easily (Fig. 6(a)). The other case is that the dimension of r_s in width and length is less than w. Although the inflated rectangles can be one of two cases, the result can be determined with the aid of L_{MIN}^1. As the intersection outcome is a line, the situation is the same as that for the result that is a rectangle. As the intersection outcome is a point, two original inflated rectangles also can be derived with the aid of L_{MIN}^1. Since $L_{MIN}^1 = L'^1_{MIN}$, for the case that we have to choose one out of two combinations of two

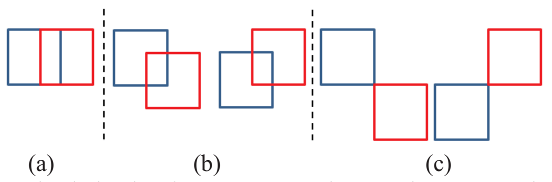

Fig. 6. The derivation from non-empty intersection to two original inflated rectangles. (a) and (b) the intersection is a rectangle; (c) the intersection is a point.

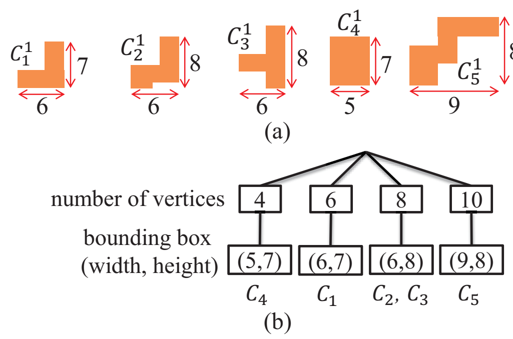

Fig. 7. Example of comparison tree construction. (a) Polygons of $\boldsymbol{C_1^1} \sim \boldsymbol{C_5^1}$; (b) corresponding comparison tree.

rectangles, the same combination has to be selected for both R and R' because $L_{MIN}^1 = L'^1_{MIN}$. Finally the equality between R and R' is obtained by choosing the same combination of two rectangles. ■

However, each frequently used polygon in L_{MIN}^1 is not generally fit to the user-defined window of size W. A window clipping method is thus applied on each p_i^1, which is larger than W, to locate the center of each pattern window based on C_i. With these identified window patterns, exact matching algorithm can be applied to the original layout to identify the locations of all instances for every window pattern.

4.1 Comparison Tree Classification

An intuitive method to investigate identical polygons in L_{MIN}^1 (level-1 polygons) is to check the equivalence of every pair of level-1 polygons. It requires time complexity at least $O(m^2)$, where m is the number of level-1 polygons. A comparison tree is designed to classify level-1 polygons into groups. Then, the equivalent checking of two level-1 polygons is only required within the same group.

The proposed comparison tree is a two-level tree. The level-1 polygons are categorized in the first level in terms of the numbers of corner points of each polygon while the width and height of the bounding box of each polygon are used in the second level. Then, every leaf contains a list of level-1 polygons satisfying the key in each level. A comparison tree example in Fig. 7 has five level-1 polygons C_1 to C_5 and C_i^1 is illustrated in Fig. 7(a). The associated comparison tree is shown in Fig. 7(b). Obviously, level-1 polygons in two leaf groups are different, and thus the exact matching of two polygons only happens in the same group. In this example we only need to check the equivalence of C_2 and C_3, resulting in a considerable performance improvement.

4.2 Window Clipping and Pattern Reconstruction

The FUPGs in L_{MIN}^1 is generally larger than the user-defined window. Next step is to locate the window centers based on $L_{MIN}^{|L_{MIN}|}$. For example, Fig. 10(a) and 10(b) are a pattern and its associated MIL layout L_{MIL}, respectively. The square with blue dotted lines in Fig. 10(a) is the desired window of size W.

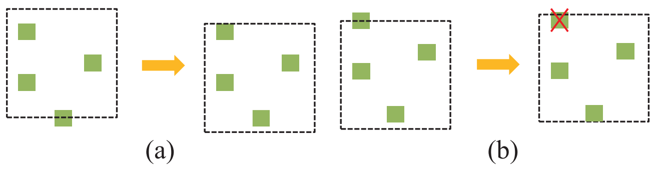

Fig. 8. Pattern candidates in Fig. 10(i) and 10(j) require pattern boundary adjustment to avoid rectangle clipping on the boundary. (a) Shift the pattern boundary downwards; (b) expel the clipped rectangle if window shifting to make the clipped rectangle un-clipped yields another newly clipped rectangle.

Algorithm 2. Pattern_Generation

Input: A set of polygons P and window size W.
Output: A set of generated patterns.

```
1. Determine inflation size $d$ with $W$.
2. MIL_Layout_Construction($P$, $d$).
3. Construct comparison tree for $L_{MIL}$.
4. $\varDelta \leftarrow$ A set of frequently used polygons $C_i \subseteq L_{MIL}$.
5. $\mathbb{C} \leftarrow \forall$ centers of $C_i^{|L_{MIL}|}$, where $C_i \in \varDelta$.
6. $\forall\ c_j \in \mathbb{C}$, do
7.     Generate candidate pattern with respect to $c_j$ and $W$.
8.     if exists incomplete rectangles, then
9.         $\forall$ possible moving directions, do
10.            make the most clipped rectangles un-clipped.
11.            Remove remaining clipped rectangles.
12.        Choose the pattern with the most rectangles.
```

Fig. 9. Algorithm of pattern generation.

According to Proposition 1, the regions in the topmost layer of a MIL layout involve the most inflated rectangles to overlap, which implies locating a window center at the center of a component in the topmost layer of L_{MIL} can accommodate the most vias in the window. Because the L_{MIL} in Fig. 10(b) has four components in C^3, we have four clipping candidates as shown in Fig. 10(c) to 10(f). With these clipping center candidates, pattern candidates fitting the window of size W can be generated. Fig. 10(g) to 10(j) show that the pattern candidates with respect to the clipping center candidates shown in Fig. 10(c) to 10(f).

One important issue of generating pattern candidates is that some rectangles are not totally inside the window and become clipped as shown in Fig. 10(i) and 10(j). Herein, we consider a clipped rectangle in a pattern as a buzz for pattern matching and try to make a clipped rectangle un-clipped by window shifting (Fig. 8(a)) or just remove the clipped rectangle if window shifting to make the clipped rectangle un-clipped yields another newly clipped rectangle (Fig. 8(b)). Fig. 9 shows the proposed pattern generation algorithm.

4.3 FUC Generation Flow

Algorithm 3 (Fig. 11) is the overall flow of the proposed FUC generation algorithm. Given the layout and the windows size W, the proposed pattern generation algorithm (Algorithm 2) is firstly invoked to generate a set of FUPGs in L_{MIN}^1. Window clipping is then applied to every FUPG to obtain a set of pattern candidates. Exact pattern matching method [9] can thus be applied to the layout with the generated pattern candidate set to find all repeated instances of every pattern. Thereafter, the most FUP is identified and added to the set of FUCs. All instances of this pattern are removed from the layout and this pattern is also removed from the set of pattern candidates. This procedure continues until the number of characters is

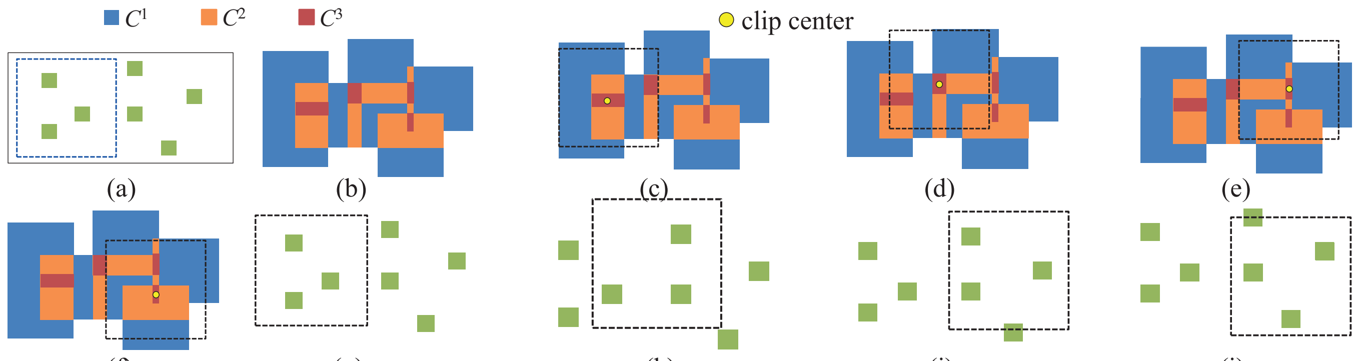

Fig. 10. Example of pattern clip. (a) Original layout; (b) the corresponding MIL layout; (c)~(f) clipping candidates with centers and corresponding pattern boundaries; (g)~(j) associated clips of (c) ~ (f).

larger than a threshold due to the limit of stencil area, or we cannot extract more characters from the layout.

4.4 FUC Generation Flow for Metal Layers and Panel Layouts

The rectangles in metal layers are much more diverse in length than those in via layers. Thus the polygons of L^1_{MIN} in metal layers are much larger than those in via layers; as a result, the frequency to repeat a polygon of L^1_{MIN} in metal layers is not so high. The proposed FUC generation flow for via layers is simplified for the application to metal layers by removing the step of identifying FUPGs in L^1_{MIN}. The flow is updated as follows. Instead of conducting window clipping on the FUPGs in L^1_{MIN}, window clipping is performed on each polygon of L^1_{MIN} for metal layers to obtain a set of pattern candidates. Next, the same procedure as that for via layers is invoked to obtain FUCs.

Algorithm 3. Character Generation Algorithm

Input: A set of polygons P, window size W, a set of pattern candidates P_c, and the maximum number of characters $\mathbb{T}$.
Output: A set of FUCs Ψ.

```
1. while (|Ψ| ≦ 𝕋 & P_c is not empty) do,
2.    Pattern_Generation(P, W). (Algorithm2)
3.    Find pattern frequencies with [9].
4.    φ ← pattern with the highest frequency.
5.    Ψ ← Ψ ∪ {φ}.
6.     Cut all φ from the given layout.
7.    P_c = P_c - φ.
```

Fig. 11. Algorithm of the proposed character generation.

5. EXPERIMENTAL RESULTS

The algorithm is implemented using C++ on a Linux platform with a 2.4 GHz quad-core CPU and 80 GB RAM. Three benchmarks are adopted for the experiment. The first benchmark is a module b15 from OpenSparc T1 design. The layout is synthesized with 15nm Open Cell Library using SOC Encounter. The second and the third benchmarks are industrial designs, where the second benchmark is a layout in via and interconnection layers, and the third benchmark is a panel layout. Table II shows the corresponding numbers of polygons (*psize*), and the numbers of rectangles (*rsize*) after slicing polygons to non-overlap rectangles.

In this experiment, we compute the character covering rate that is evaluated as the percentages of all rectangles in a layout printed by characters under different window sizes (W) and inflation sizes (d). According to the maximum character size in previous work [6] (maximum character size = 30 grids with a 1 pitch × 1 pitch square as one grid), we use three window sizes in this experiment, which are 3, 5, and 7 pitches. Besides, the levels of a MIL layout and its construction runtime increase quickly if d becomes too large. A large d results in the worthless intersection among the rectangles that are far away. Under these concerns, inflation size is chosen to set the level number of a MIL layout as about three to six. Furthermore, the chosen inflation sizes are the half of polygon spacing at the peaks in the polygon spacing distributions. The coverage rates of entire layouts are presented in Table III while the sizes of generated character sets and the overall runtime are presented in Table IV and Table V, respectively. In addition, we do not set the upper bound for the size of the character set ($\mathbb{T}$ in Algorithm 3). We compare the average number of rectangles in an E-beam shot (*ANRE*) and CP-efficiency (E_{CP}). E_{CP} is defined as:

$$E_{CP} = \frac{\#E\text{−}beam\ shot\ by\ VSB\ (\#sliced\ rectangle)}{\#E\text{−}beam\ shot\ by\ CP\text{−}EBL}$$

Notably, the remaining rectangles not printed by CP are printed using VSB instead. On the other hand, E_{CP} is evaluated as the average number of rectangles in an E-beam shot. Both *ANRE* and E_{CP} are presented in Table VI. In this table, only the inflation sizes producing the maximum coverage rates under the given W are listed. Finally, in Table VII, we show the statistics of the top 5 patterns of the benchmarks using the settings of W and d that produce the highest coverage rate in Table III. These statistics summarize the numbers and the size, i.e. the number of polygons in the pattern, of these top 5 patterns.

As we can observe from these tables, the rectangle density in b15 is very low, so it requires larger inflation sizes to produce a 3-level MIL layout. As a result, it is hard to produce FUCs, and the corresponding *ANRE* and E_{CP} are also low for b15. On the contrary, the panel layout is much more regular, and thus has 100% coverage rate and considerable throughput improvement ($E_{CP} > 8$). According to Table VI, 99.7% of the sliced rectangles can be printed by top 3 FUCs of the panel layout when W = 5 pitch and d = 1.5s. Even for b15, 45.2% and 67.4% of the rectangles in via and metal layer can be printed by top 5 FUCs, respectively. These results show that the proposed algorithm can efficiently generate good character set for improving CP throughput in via and interconnection layers, and even on a panel layout. Fig. 12 shows the top 9 FUCs for the panel layout to achieve 100% covering rate..

Table II. Benchmarks statistics.

	b15/via	b15/metal	industry/via	industry/metal	panel
psize	25,479	8204	399,012	170,377	545,790
rsize	25,479	12,864	399,012	873,680	1,090,138

Table III. Character covering rate (%) under different inflation sizes (*d*) and different window sizes (*W*).

W \ *d*	b15/via (%)			b15/metal (%)			industry/via (%)			industry/metal (%)			panel (%)		
	5*s*	9*s*	13*s*	5.3*s*	6.9*s*	8.5*s*	3.5*s*	5.5*s*	7.5*s*	2*s*	2.5*s*	3.2*s*	1.5*s*	2*s*	2.5*s*
3 pitch	**72.3**	40.52	37.34	**81.23**	66.78	57.9	69.2	**93.3**	78.12	**73.2**	61.54	51.9	97.3	97.3	97.3
5 pitch	27.3	**57.3**	21.07	49.71	61.55	**70.62**	32.3	87.3	**89.07**	46.91	80.19	**81.23**	**100**	**100**	**100**
7 pitch	**21.19**	25.31	19.6	31.1	53.1	**66.8**	19.1	46.17	**77.84**	31.28	40.9	**72.45**	95.1	95.1	99.9

Table IV. Sizes of character set under different inflation sizes and different window sizes.

W \ *d*	b15/via			b15/metal			industry/via			industry/metal			panel		
	5*s*	9*s*	13*s*	5.3*s*	6.9*s*	8.5*s*	3.5*s*	5.5*s*	7.5*s*	2*s*	2.5*s*	3.2*s*	1.5*s*	2*s*	2.5*s*
3 pitch	530	413	312	547	428	292	65487	133295	96574	205684	121541	115003	5	5	5
5 pitch	110	237	78	192	277	387	10853	118833	58208	93698	135478	131471	9	9	9
7 pitch	71	195	126	109	183	269	96574	126574	111947	15472	56489	102453	13	13	11

Table V. Overall runtime (minute).

W \ *d*	b15/via			b15/metal			industry/via			industry/metal			panel		
	5*s*	9*s*	13*s*	5.3*s*	6.9*s*	8.5*s*	3.5*s*	5.5*s*	7.5*s*	2*s*	2.5*s*	3.2*s*	1.5*s*	2*s*	2.5*s*
3 pitch	13.77	12.06	16.18	13.22	14.76	32.7	16.01	30.2	38.58	79.7	86.9	36.65	4.48	7.92	10.17
5 pitch	5.39	17.72	7.65	10.23	10.6	57.98	6.65	32.66	58.3	160.78	166.36	116.18	6.3	8.56	15.23
7 pitch	2.83	13.32	11.87	5.2	9.23	28.15	2.01	16.45	50.16	232.7	237.2	194.98	6.68	8.8	16.9

Table VI. Required CP-efficiency (E_{CP}) comparisons.

W	b15/via			b15/metal			industry/via			industry/metal			panel		
	d	E_{CP}	*ANRE*	*d*	E_{CP}	*ANRE*	*d*	E_{CP}	*ANRE*	*d*	E_{CP}	*ANRE*	*d*	E_{CP}	*ANRE*
3 pitch	5*s*	1.97	3.6	5.3*s*	2.43	3.5	5.5*s*	4.12	3.3	2*s*	2.67	4.7	1.5*s*	7.12	7.1
5 pitch	9*s*	1.87	3.3	8.5*s*	1.92	4.3	7.5*s*	3.64	4.1	3.2*s*	2.92	6.1	1.5*s*	8.53	8.9
7 pitch	5*s*	1.28	4.3	8.5*s*	1.88	5.5	7.5*s*	2.61	4.9	3.2*s*	2.77	7.3	1.5*s*	6.55	10.1

Table VII. Top 5 FUPs of benchmarks with settings resulting in the highest coverage in Table III.

Benchmark	***W / d***	1st pattern		2nd pattern		3rd pattern		4th pattern		5th pattern	
		number	size	number	size	number	size	number	size	number	size
b15/via	3 pitch / 5*s*	1396	3	973	3	736	3	447	3	324	3
b15/metal	3 pitch / 5.3*s*	738	3	601	4	591	3	456	3	227	4
industry/via	3 pitch / 5.5*s*	25548	3	19964	3	17745	3	11439	4	8416	4
industry/metal	5 pitch / 3.2*s*	17652	4	10039	5	9127	5	8365	6	6033	5
panel	5 pitch / 1.5*s*	105,651	9	10892	6	10127	7	375	8	320	8

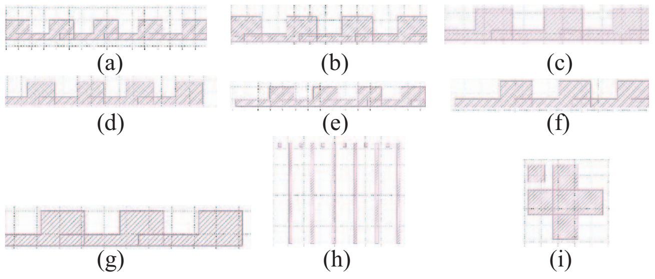

Fig. 12. FUPs of the panel under $W = 5$ pitch and $d = 1.5s$. (a) ~ (i) are top 1 to top 9 FUCs.

6. CONCLUSIONS

Pattern diversity is a critical challenge of using CP on interconnection layers. Existing works is hard to generate good characters for 1-D IC layout and 2-D panel layout. Hence, CP-EBL cannot fully utilize its power of high throughput fabrication on interconnection layers and retina-resolution panel layout. This work proposes an efficient character generation algorithm for CP-EBL based on the presented MIL layout. The inflated layer reduces the problem instance size for identifying the frequently used patterns while the intersection layers help in clipping windows to obtain ideal character set. Experimental results show that the proposed methodology can efficiently yield the frequently used character set with up to 93.3% and 81.23% covering rate in via layer and metal layer. Besides, for a panel layout, a set of frequently used characters to reach 100% covering rate is successfully identified.

7. REFERENCES

[1] Takeshi Fujino, *et al*, "Character-Build Standard-Cell Layout Technique for High-Throughput Character-Projection EB Lithography," In *Proc. of SPIE*, 2005.

[2] K. Yuan, B. Yu, and David Z. Pan, "E-Beam Lithography Stencil Planning and Optimization with Overlapped Characters," In *IEEE TCAD*, Vol. 31, Issue 2, pp. 167-179, 2012.

[3] C. Chu, and W.-K. Mak, "Flexible Packed Stencil Design With Multiple Shaping Apertures and Overlapping Shots for E-beam Lithography," In *IEEE TCAD*, Vol. 34, Issue 10, pp. 1652-1663, 2015.

[4] D. Guo, Y. Du, and Martin D.F. Wong, "Polynomial Time Optimal Algorithm for Stencil Row Planning in E-Beam Lithography," In *Proc. of ASP-DAC*, 2015.

[5] P. Du *et al*, "Character Design and Stamp Algorithms for Character Projection Electron-Beam Lithography," In *Proc. of ASP-DAC*, 2012.

[6] R. Ikeno *et al*, "High-throughput Electron Beam Direct Writing of VIA Layers by Character Projection using Character Sets Based on One-dimensional VIA Arrays with Area-efficient Stencil Design," In *Proc. of ASP-DAC*, 2013.

[7] Masahiro Shoji, *et al*, "Practical Use of The Repeating Patterns in Mask Writing," In Proc. of SPIE, 2010.

[8] Rimon Ikeno, *et al*, "Line-Edge Quality Optimization of Electron Beam Resist for High-Throughput Character Projection Exposure Utilizing Atomic Force Microscope Analysis," In *Proc. of SPIE*, 2017.

[9] H.-Y. Su *et al*, "A Novel Fast Layout Encoding Method for Exact Multilayer Pattern Matching with Prüfer Encoding", In *IEEE TCAD*, Vol. 34, Issue 1, pp. 95-108, 2015.