A Highly-Efficient Row-Structure Stencil Planning Approach for E-Beam Lithography with Overlapped Characters

Jian Kuang
Dept. of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong
jkuang@cse.cuhk.edu.hk

Evangeline F. Y. Young
Dept. of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong
fyyoung@cse.cuhk.edu.hk

ABSTRACT
Character projection is a key technology to enhance throughput of e-beam lithography, in which characters need to be selected and placed on the stencil. This paper solves the problem of planning for overlapping-aware row-structure stencil, and also considers multi-column cell system for further throughput improvement. We propose an integrated framework to solve the subproblems of character selection, row distribution, single-row ordering and inter-row swapping efficiently. Experiments show that our approach outperforms the existing methods on all the benchmarks. We can achieve significant throughput improvement and up to $1782 \times$ speedup comparing with previous works. The average speedup is $704 \times$.

Categories and Subject Descriptors
B.7.2 [Integrated Circuits]: Design Aids

Keywords
Design for Manufacturability, E-Beam Lithography, Character Projection, the MCC System

1. INTRODUCTION
Manufacturability is one of the most critical issues in semiconductor industry. As technology advances and the minimum feature size keeps scaling down, it becomes even more challenging. The major printing technique, on the other hand, is still 193nm ArF immersion lithography. To handle the mismatch between current lithography capacity and the feature size in the sub-22nm technology node, double patterning lithography (DPL) and triple patterning lithography (TPL), that decompose one layer of layout into multiple masks, have been proposed for years [8, 11, 3, 7]. However, the cost of manufacturing will become much higher with more masks. As a consequence, the next generation lithography (NGL) technologies, such as extreme ultra-violet (EUV) and electron-beam lithography (EBL), are expected to be promising solutions in 14nm/10nm technology node.

However, both EUV and EBL are not ready for mass production yet. The availability of EUV is further delayed by the technological difficulties such as mask blank defects [13]. On the other hand, EBL suffers from the bottleneck of low throughput. Several key technologies, like character projection (CP) and multi-column cell (MCC) system, have been proposed to overcome this limitation.

EBL is a maskless lithography technology that shoots a beam of electrons onto a wafer and directly creates desired shapes there. Benefiting from this, EBL can achieve high resolution with relatively lower cost, compared with the cost of making masks. The total writing time of a layout is proportional to the number of shots that are made, thus fewer shots means higher throughput. Conventional EBL use variable shaped beam (VSB), by which every shot can only create one rectangle, hence the total number of shots will be unacceptable for high-volume manufacturing. CP is then proposed to handle this problem. With CP, various characters will be pre-designed. Each character is composed of some complex patterns. A limited number of characters can be selected and placed on the stencil. A character on the stencil needs only one shot to be printed, whereas a character not on the stencil still needs VSB. CP can improve
the throughput of EBL greatly. For example, the patterns in Figure 1(a) needs 5 shots by VSB, but if a character is designed to contain the patterns and is put on the stencil, as shown in Figure 1(b), only 1 shot is needed. Note that a character may appear more than once in a layout, thus many more shots can actually be saved by CP.

There are a lot of optimization problems with CP. First of all, there will be restrictions on design rules to allow common CP patterns in the layout. Furthermore, character design is also a hard problem and attracts extensive research in these few years [6, 5, 2]. After the characters are ready, how to select appropriate characters and place them on the stencil (stencil planning) is another big challenge. In practice, each character has some blank areas surrounding it. By sharing the blank areas, the characters can be overlapped to save more stencil space and thus more characters can be placed on the stencil [4]. When CP is applied to standard cell design, each character usually contains the patterns of one cell. All characters have the same size and uniform top and bottom blank areas. Vertical overlapping can thus be optimized easily and only horizontal overlapping need to be considered [12, 10]. In this case, the stencil can be viewed as a set of rows, and the selected characters need to be packed into the rows. As shown in Figure 2, two characters i and j are placed in one row, overlapping with each other. Note that the uniform top and bottom blank areas are omitted w.l.o.g as in the previous work [10]. The MCC system [9] is a recent extension of EBL, in which a layout is divided into several regions, and several CPs print on these regions in parallel. The actual printing time of the layout is the maximum printing time among these regions, which is obviously shorter than conventional EBL. Because of the high complexity of designing stencil, the stencils for different CPs will contain the same set of characters and the characters will share the same placement. Therefore, for the MCC system, we still just have one stencil to design, but it is harder than that for conventional EBL, since the effects on different regions need to be considered simultaneously.

1.1 Previous Work

Overlapping-aware stencil planning (OSP) for row-structure stencil was referred as 1D-OSP problem. Yuan et al. [12] proposed the first systematic study on this topic, but the greedy and heuristic methods that were used are time consuming and lack of global view. E-BLOW [10] was then developed for OSP in the MCC system. The problem was formulated as an LP and solved iteratively by successive relaxation. However, the optimality loss because of rounding after LP is unavoidable and the iterative process will be relatively slow. Furthermore, it failed to recognize the difference between conventional EBL and the MCC system. So, although the approach in [10] already performs much better than that in [12], the quality and efficiency can be further optimized for both types of EBL.

1.2 Our Contributions

In this paper, we propose a new approach to solve 1D-OSP, our main contributions can be summarized as follows:

- We propose an efficient and optimal algorithm to solve the single-row ordering problem under a condition. We will also present a simple way to distribute the characters into rows such that the condition can be satisfied.
- We present a greedy approach to solve the character selection problem and an accelerated ILP method to improve the solution quality.
- We propose an inter-row swapping algorithm to improve the result further. It is a deterministic algorithm instead of a random approach as in [12].
- Our approach performs very well in experiments, with remarkable reduction in shot numbers and up to 1782× speedup comparing with the most updated result.

The rest of the paper is organized as follows. Section 2 introduces some preliminaries for the 1D-OSP problem. Section 3 describes the whole stencil planning approach. Section 4 reports experimental results and Section 5 concludes this paper.

2. PRELIMINARIES

First, some notations will be introduced. Suppose that the character design has already been done, and a set of n characters \( C = \{c_1, c_2, \ldots, c_i, \ldots, c_n\} \) is given. The width of a character is \( w_i \). The left and right blank length of \( c_i \) are \( l_i \) and \( r_i \) respectively. A stencil consisting of \( k \) rows is given. We need to select a set of characters \( C' \subseteq C \) to put into the rows of the stencil. If \( c_i \) is not on the stencil and printed by VSB, it needs \( v_i \) shots, otherwise it will be printed by CP, and only 1 shot is needed. Suppose that character \( c_i \) appears \( t_i \) times in the layout. If only VSB is used, the original shot number for the whole layout will be:

\[
S_o = \sum_{i=1}^{n} t_i \cdot v_i
\]

Putting \( c_i \) on the stencil will reduce the total number of shots, this reduction is called the gain of \( c_i \). The gain can be calculated as:

\[
gain_i = (v_i - 1) \cdot t_i
\]

<table>
<thead>
<tr>
<th>Gain of Character</th>
<th>Shot Number</th>
<th>Time Constant</th>
</tr>
</thead>
</table>

Notice that in this paper we assume that CP shot and VSB shot have equal cost, as in the previous work[12, 10], but our approach can be easily extended to process CP shot that requires longer time. After putting the characters in \( C' \), the total shot number will be:

\[
S = S_o - \sum_{c_i \in C'} gain_i = \sum_{i=1}^{n} t_i \cdot v_i - \sum_{c_i \in C'} (v_i - 1) \cdot t_i
\]

Although in this paper we assume that all characters have the same with, which is also true in all the benchmarks given, our formulations and algorithms can be easily extended to handle characters with different width.
Characters can be overlapped when they are put into rows. Suppose character $i$ is on the left of $j$, the overlapping space between them will be $\min\{t_i, r_j\}$ (Figure 2). The total length of the characters in a row after compacting cannot exceed the width of the stencil $W$.

The notations for the MCC system will be slightly different. The layout is divided into $m$ regions: $R_1, R_2, \ldots, R_i, \ldots, R_m$. Character $c_i$ appears $t'_i$ times in the region $R_j$. The shot number of $R_j$ after stencil design will be:

$$S_j = \sum_{i=1}^{n} t'_i \cdot v_i - \sum_{c_i \in C} (v_i - 1) \cdot t'_i$$  \hspace{1cm} (4)

where $(v_i - 1) \cdot t'_i$ is the gain of $c_i$ in $R_j$. The total shot number of the layout that we want to minimize is:

$$S = \max_j \{S_j\}$$  \hspace{1cm} (5)

Now we can give the problem formulation of 1D-OSP.

**Problem 1.** Given a set $C$ of characters, and a stencil of $k$ rows and width $W$, select a subset of $C$ and decide their positions in the rows of the stencil, such that the width of the stencil is not exceeded. The objective is to minimize the objective equation (3) for conventional EBL or to minimize the objective equation (5) for the MCC system.

3. **THE STENCIL PLANNING APPROACH**

In this section, we give details of our approach for 1D-OSP. We first solve several subproblems, and then integrate them together.

3.1 Single-Row Ordering

After we select $m$ characters into a particular row, we need to minimize their total length in order to make sure that all characters are within the row and try to put more characters into the row. This problem was formulated as a traveling salesman problem in [12], which is well-known to be NP-hard. (This is also an important reason why the approach in [12] is much slower than that in [10] and our approach). However, we will show below that this problem can be solved optimally in a special case.

Suppose the characters in a row are $\{c_1, c_2, \ldots, c_l, \ldots, c_p\}$, then the total length of these characters with a specified order is

$$L = p \cdot w - \text{TotalOverlappingSpace}$$  \hspace{1cm} (6)

Since $p \cdot w$ is a constant, minimizing $L$ is equivalent to maximizing $\text{TotalOverlappingSpace}$. To solve this problem, bipartite matching between the left and right blanks seems to be a good candidate approach. We describe our matching-based method that can find the optimal $\text{TotalOverlappingSpace}$ under a reasonable constraint as follows:

**Step 1:** Construct a bipartite graph $G=(U, V, E)$. Each character $c_i$ has two nodes $u_i$ and $v_i$ in $U$ and $V$, representing the left and right blank area of $c_i$ respectively. Every node is weighted: $\text{weight}(u_i) = l_i$, $\text{weight}(v_i) = r_i$. Edge set $E = \{(u_i, v_j) : i \neq j\}$. Edges are also weighted: $\text{weight}(e(u_i, v_j)) = \min\{\text{weight}(u_i), \text{weight}(v_j)\}$. An example is shown in Figure 3(a), where there are 8 nodes representing the left and right blank areas of $c_1, c_2, c_3$ and $c_4$ respectively.

By now, one possible thought is to find a maximum weighted bipartite matching on $G$, which can be done in polynomial time. If $u_i$ and $v_j$ are matched, the left blank area of $c_i$ and the right blank area of $c_j$ should be put together (thus $c_i$ is on the right of $c_j$) and an overlapping space with length $= \text{weight}(e(u_i, v_j))$ will be realized. For example, the matching solution in Figure 3(b) means that $c_1$ should be put on the right of $c_2$, etc. With the matching solution, we can modify the graph to find the order of placing the characters. First we can remove the edges in $G$ that are not in the matching solution, and then add edges $E' = \{(u_i, v_j) : 1 \leq i \leq m\}$ to $G$. The resulting graph of our example is shown in Figure 3(c). The least-weighted edge in the matching solution should be removed, since the matching edges together with the edges in $E'$ form a loop in this example but the characters are supposed to be placed in a row. Thus we choose to give up the smallest overlapping space to break the loop. For example, edge $e(u_2, v_3)$ is removed in Figure 3(c). Then we can check the directed path $p$ from the degree-1 node in $U$ to another degree-1 node in $V$. If $p$ covers all the nodes in $G$, the optimal order of the characters can be derived from the order of the nodes appeared in $p$. For example, the path from $u_2$ to $v_3$ in Figure 3(d) implies the order $c_2 \rightarrow c_1 \rightarrow c_3$. However, this method will work only when $p$ covers all the nodes. For example, if the maximum matching solution is as shown in Figure 3(e), the modified graph will be like the one shown in Figure 3(f). Here we cannot find any single path $p$ that covers all the nodes in $G$ (see Figure 3(g)), thus a full ordering of the characters cannot be obtained from the matching solution. Although the method above does not work in general, it can be modified to give the optimal solution effectively under some reasonable constraint on the left and right blanks as described below:

**Step 2:** Let $\text{weight}_{\text{min}}$ be the minimum weight among all the nodes in $G$. Update the weight of every node $v$ as $\text{weight}(v) - \text{weight}_{\text{min}}$. This can simplify the graph and
will not lose any optimality since the overlapping space of the minimum blank area can always be achieved between any two adjacent characters in any order.

Step 3: Remove those nodes with weight 0 and their corresponding edges. There is no need to consider the removed nodes and edges further.

Step 4: Update all the edge weights according the new node weights to obtain graph \( G' \).

We have the following theorem about the maximum matching on the graph \( G' \).

**Theorem 1.** When all the edges in \( G' \) are with equal weight, the maximum weighted matching on \( G' \) can always give a character ordering with the optimal overlapping space.

**Proof.** Let \( N_U \) and \( N_V \) be the numbers of nodes of \( G' \) in \( U \) and \( V \) respectively. Let \( DS \) be the set of characters that have corresponding nodes in both \( U \) and \( V \), e.g., in Figure 4(d), \( DS = \{c_1, c_2\} \). Suppose all the edges in \( G' \) are with the same weight \( w_c \). There are two cases.

Case 1: \( N_U = N_V = |DS| \). An example is shown in Figure 4(a). In this case, the maximum weighted matching \( w_c \cdot DS \) can be easily achieved as shown in Figure 4(b). With the matching solution, just as what we did above (adding and removing some edges), we can get the path \( p \) and the order, e.g., we can get order \( c_1 \rightarrow c_3 \rightarrow c_2 \) as shown in Figure 4(c), which is resulted from the graph in Figure 4(b). Obviously, the optimal overlapping space \( w_c \cdot (|DS| - 1) \) is achieved.

Case 2: \( N_U \neq |DS| \) or \( N_V \neq |DS| \). An example is shown in Figure 4(d). The optimal overlapping space in this case is \( w_c \cdot \min\{N_U, N_V\} \). We first divide the nodes into two groups, the nodes corresponding to the characters in \( DS \) are in group \( A \), others are in group \( B \). Then we do two matchings on the two subgraphs induced from the two groups of nodes. The first matching is just like the matching in case 1, the second one is trivial since all the nodes in group \( B \) are independent from each other. Figure 4(e) shows a possible matching result on the graph in Figure 4(d), where only matched edges are shown and a loop between \( u_1, u_2, v_1 \) and \( v_2 \) exists. If we just remove one matched edge to break the loop like what we did in case 1, we will get suboptimal solution. Actually, we can exchange one node in \( U \cup V \) of group \( A \) and another node in \( U \cup V \) of group \( B \) to break the loop and get the optimal overlapping space, e.g., we exchange \( u_2 \) and \( u_4 \) as shown in Figure 4(f). We can then add back the removed nodes and get the path. In this way the optimal order can be found (see Figure 4(g), where uncolored nodes are the ones that were removed previously).

Note that there is no need to really call a matching solver. The matching solution can be obtained following the steps in our proof.

### 3.2 Row Distribution

#### 3.2.1 Constraint Satisfaction

With Theorem 1, an immediate problem is how to ensure that the constraint will be satisfied. In the following, we will first present a simple sorting-based approach to solve this problem. We sort all the selected characters to be put on the stencil according to the following criteria: \( c_i \) is before \( c_j \) iff \( l_i > l_j \) or \( (l_i = l_j ) \wedge (r_i > r_j ) \). In practice, the length of the left and right blanks of a character \( c_i \) are integers and very similar to each other, typically we have \( |l_i - r_i| \leq 1 \). Thus a sorted list of the characters will look like the one shown in Figure 5(a), in which characters are divided into clusters, e.g., characters with left blank of length \( a+1 \) and right blank of length \( a \) form a cluster. When we distribute the characters into rows following the sorted order, characters in clusters that are close to one another in the sorted list will be placed into a row. Let \( N_c \) and \( N_e \) be the numbers of characters in a cluster and in a row respectively. In practice, for any \( N_c \) and \( N_e \), we always have \( N_c \leq 2 \cdot N_e \), which means that there are not too many characters in a row, so characters in a row will be from at most three neighbouring clusters\(^2\), e.g., the three clusters at the bottom of Figure 5(a). In this way, the constraint in our optimal row ordering method will be satisfied, which is, edges in the graph \( G' \) are of equal weight 1.

Beside satisfying the constraint, another advantage of our distribution method is that all the blank areas of the character...
Figure 6: (a) Before redistribution, 3 pairs of extra blanks are overlapped in total. (b) After redistribution, 5 pairs of extra blanks are overlapped. Note that nodes corresponding to characters without extra blanks are not shown, so there may be more than 4 characters in each row.

3.2.2 Redistribution

By the greedy distribution above, each row can be optimally ordered independently, but the relationships between rows are not considered, thus redistribution is needed. Let $O_{min}$ denote the minimum blank area among the characters in a row. After the distribution, the rows of the stencil can be divided into groups according to different $O_{min}$, e.g., the group with $O_{min}$ being a has 2 rows in Figure 5(b). We call the extra part of the blank area other than $O_{min}$ of each character the extra blank. Following the steps in our proof of Theorem 1, $N_U$ and $N_V$ will be the numbers of left and right extra blanks in a row, respectively. Extra blanks are important for us to increase $TotalOverlappingSpace$ and thus to put more characters into a row, but they may be wasted because of unbalanced distribution, e.g., in a group, one row has $N_U = 3$ and $N_V = 1$, and another row has $N_U = 2$ and $N_V = 4$, as shown in Figure 6(a). After redistribution (exchanging $c_4$ and $c_8$ in this example), there will be more extra blanks overlapped (Figure 6(b)), and $TotalOverlappingSpace$ of both $row_1$ and $row_2$ will be increased.

To make full use of extra blanks, we will redistribute the characters among the rows in a group so as to maximize the number of extra characters (denoted as $x$) we can add to the group. This is not a trivial step. First of all, the value of $x$ is constrained by the number of rows in the group. When a group has $r_g$ rows, the maximum value of $x$ we will test is $2 - r_g$, because we found that one row can take at most 2 more characters through redistribution in practice. Secondly, as one row can only take a limited number of characters because the row length cannot be exceeded, so even if there are enough extra blanks available, we might not be able to distribute them as desired. For example, as shown in Figure 6, if we want to make 5 pairs of overlapped extra blanks in $row_1$ such that the $TotalOverlappingSpace$ of this row will be increased enough and one more character can be put in, one way is to put the 8 characters $c_1 \sim c_8$ into $row_1$, but doing this may violate the constraint of row width since $row_1$ can only take no more than 7 characters. Finally, the characters with extra blanks are with different types. Some have extra blanks only on the left or on the right, whereas the others have extra blanks on both sides. As a result, they need to be combined and utilized carefully. For instance, when $r_g = 2$, to test whether $x$ can be 3, we need to consider the extra blanks available and the extra blanks that we need, and check how the characters with different types of extra blanks in the group can be redistributed such that one row will have enough space to take two more characters and another row will have enough room to accommodate one more character, without any violation in the stencil width.

Notice that after redistribution within each group, characters in each row can still be ordered optimally as the constraint in Theorem 1 is still satisfied.

3.3 Inter-Row Swapping

After row distribution, there is still some room to improve the placement by swapping characters and putting in more characters. This is because a character in one group may be more useful in another group, e.g., character $c_i$ with $l_i = r_i = a + 1$ has no extra blank in the group of $O_{min} = a$, but it has two extra blanks if it is in the group of $O_{min} = a + 1$.

There are two types of swapping: (1) The left (right) blank area of the leftmost (rightmost) character of a row is not used because there is no overlapping in the two ends of a row, so we can replace the leftmost (rightmost) character of a row by one with smaller left (right) blank area, and the length of the characters in this row will not be affected. (2) When the length $L$ of the characters in a row is smaller than $W$, after we replace some characters in this row by some others with smaller blank areas, although $TotalOverlappingSpace$ will decrease and thus $L$ will increase, $L$ can still be not exceeding $W$.

After replacing, the replaced characters can be used in other groups such that these groups will have more extra blanks available to take more extra characters. Our swapping method will find the swapping candidates in a deterministic way, thus randomness as in the method of [12] is avoided.

3.4 Character Selection

The above three steps tell how to distribute characters into rows, how to order the characters in one row, and how to swap characters between rows. However, we still need to resolve a bigger problem of selecting a subset of $C$ of characters to put on the stencil. To solve this problem, assuming that the average blank length among all the characters in $C$ is $b_{avg}$, we first calculate the average number of characters $n_r$ that can be put in a row by

$$w \cdot n_r - b_{avg} \cdot (n_r - 1) \leq W$$  \hspace{1cm} (7)

Then the number of characters to select (denoted as $n_s$) can be estimated by $k \cdot n_r$. However, this can be inaccurate and affect the performance. Therefore, we will first select $n_s$ characters, try to put them onto the stencil, and then adjust $n_s$ according to the actual number of characters that can be placed. After that, we will select again and repeat the whole process. Then we introduce our character selection methods for conventional EBL and the MCC system respectively, both of which are based on the gain of each character.

3.4.1 Selection for the Convention EBL

For conventional EBL, we calculate gains by Equation (3) and rank all characters by their gains in descending order.
Then we will select the highly ranked ones. This method is simple but effective for conventional EBL, in which the whole layout is considered as one region. It rarely happens that replacing a character with a higher gain by one with a lower gain can increase the number of characters in the stencil and reduce $S$.

### 3.4.2 Selection for the MCC System

For the MCC system, we first calculate the total gain of a character by summing up the gains of this character in all regions. However, we cannot simply use the same ranking method as above, because the objective now is to minimize the maximum shot number among several regions, and a character with high total gain may not be actually good. Thus, as shown in Figure 7, we first select a set $P$ of characters with absolutely high total gains, and then identify a set of marginal characters next in the rank according to a parameter $\\alpha$ called the marginal character ratio. After selecting $P$, we will update the gains of the marginal characters, with the information of the shot number in each region, assuming that the selected characters in $P$ are already on the stencil. The gain of a character in a region with a higher shot number will be considered more. The updated gain will be more accurate than the previous simple summation, and marginal characters can thus be chosen more effectively.

However, for large datasets, as the selection ratio $sr = n_s / \left| C \right|$ decreases, the selection process will become harder and the above marginal characters identification method is not sufficient. To handle these datasets with small $sr$, we present below an ILP based selection method. Similarly, we first select a set $Q$ of characters, and then formulate the problem as an accurate ILP to select $\\beta \cdot n_s$ characters out of $2\beta \cdot n_s$ ones (Figure 7). Since solving ILP is time-consuming, $\\beta$ is set to be smaller than $\\alpha$. The ILP formulation is as follows:

\[
\begin{align*}
\text{minimize} \quad & S, \\
\text{subject to} \quad & S \geq S_j, \quad 1 \leq j \leq m \quad (a) \\
& S_j = S_{j'} - \sum_{i=1}^{2\beta \cdot n_s} y_i \cdot (v_i - 1) \cdot t_i', \quad 1 \leq j \leq m \quad (b) \\
& \sum_{i=1}^{2\beta \cdot n_s} y_i \leq \beta \cdot n_s \quad (c) \\
& y_i = 0 \text{ or } 1, \quad 1 \leq i \leq 2\beta \cdot n_s \quad (d)
\end{align*}
\]

In the formulation, $y_i$ is a binary variable. It is 1 if $c_i$ is selected, 0 otherwise. $m$ is the number of regions. $S_j$ and $S_{j'}$ are the shot numbers of $R_j$ before and after the ILP selection respectively. Equation (b) is used to update the shot numbers. Equation (c) is used to constrain the selected character number to be no more than $\beta \cdot n_s$. The number of variables is $O(\beta \cdot n_s)$. The ILP selection will be activated only when $sr$ is smaller than a threshold $\gamma$. Parameters $\\alpha$, $\\beta$ and $\gamma$ are set to 0.1, 0.05 and 0.8 respectively in our implementation.

We employ Gurobi Optimizer [1] as our ILP solver, which will terminate when the gap $g$ between the lower and upper objective bound is achieved$^4$. The solving process may be slow even for a small $\\beta$, hence we use a larger parameter $g$ (double of the original one in our implementation), by which the optimization process can be accelerated a lot but the quality loss is negligible in practice.

### 3.5 Overall Flow

Our algorithm can be summarized as follows. First, we calculate the number $n_s$ and select $n_s$ characters. Then, we update the selected characters with all the marginal characters previously identified. In each row, our optimal ordering method is applied. After that, we swap characters between rows to further improve the quality.

### 4. EXPERIMENTAL RESULTS

The proposed approach is implemented in C++, on a 2.39 GHz Linux machine with 48 GB memory. We test it on the benchmarks provided by the authors of the most updated work [10], and compare it with previous works. The shot numbers of [12] and [10] are taken from the published results. To compare runtime with [10], we run the executable file of [10] provided by the authors on our machine. We do not compare runtime with [12] because [10] is already very much faster than [12] as reported in [10].

Result comparisons are shown in Table 1, where $r\#$ is the number of rows, $c\#$ reports the number of characters on the stencil, $s\#$ denotes shot number, time is wall-clock time, reduce1 represents the reduction of shot number over [12] while reduce2 denotes the reduction over [10]. 1D-1$\sim$1D-4 are benchmarks for conventional EBL, while 1M-1$\sim$1M-9 are benchmarks for the MCC system. The region number is set as 10 for the MCC system. For the benchmarks 1D-1$\sim$1D-4 and 1M-1$\sim$1M-4, there are 1000 candidate characters, and the stencil size is set as $1000\mu m \times 1000\mu m$; for 1M-5$\sim$1M-8, there are 4000 candidate characters and the stencil size is set as $2000\mu m \times 2000\mu m$.

Character numbers on the stencil are just reported for reference because they are not necessarily related to the main objective, i.e., shot numbers. For example, if the characters with the most blank space are selected, there will be more characters on the stencil, but the shot number will also be larger. So it is about trade-offs between blank spaces and gains of characters. There is no way to compare the character numbers on the stencil with previous works, because the selected characters are different.

As shown in the table, for shot numbers, we get the best results on all the benchmarks, and we can achieve 29.6% reduction on average compared with [12] and 9.9% reduction on average compared with [10], thus the throughput for both types of E-beam lithography can be improved significantly. For runtime, comparing with [10], we can attain up

$^4$The details can be found in [1].
to 1782× speedup. Average speedup is 704×. Notice that runtime for 1M-3, 1M-4, 1M-7 and 1M-8 are longer than that for the others because ILP selection is activated for these benchmarks. However, since the ILP selection is only used to select a very small part of the characters and the number of variables is linearly related to the number of selected characters, the ILP formulation has a small size and can be solved very quickly.

For some of the benchmarks, although the numbers of shots are reduced significantly, the percentages of reductions are not that much compared with others. This is because in these benchmarks, there are relatively more characters that cannot be put on the stencil, and they contribute to the shot numbers a lot, so the large shot numbers are inevitable.

For a stencil planning approach, speed is not so important. However, as the design of circuits becomes increasingly complicated and the number of patterns explodes, the problem size of stencil planning will also get larger, and efficiency and scalability of a approach is thus crucial. Furthermore, if we consider redesign of layout and characters, an efficient approach like ours is very beneficial and highly desirable.

5. CONCLUSIONS

In this paper, 1D-OSP problem for E-beam lithography is studied. Several efficient algorithms are proposed to solve the subproblems of 1D-OSP. Experiments verify that our approach is highly efficient and very effective, through which remarkable improvement of throughput can be achieved. As EBL is widely recognized as a promising solution of the next generation lithography, we expect that this result can benefit the industry on manufacturing and attract more research on EBL-related physical design.

6. REFERENCES