Flexible Packed Stencil Design with Multiple Shaping Apertures and Overlapping Shots for E-beam Lithography

Chris Chu, *Fellow, IEEE* and Wai-Kei Mak, *Member, IEEE*

***Abstract*—Electron-beam lithography has long been employed for mask writing but the write time is increasing due to the escalating mask pattern complexity. Besides, electron-beam direct write (EBDW) is being pursued as an alternative solution for chip production in the sub-22nm regime. To improve the throughput of e-beam lithography, character projection method is commonly employed and a critical problem is to pack as many useful characters as possible onto the stencil. In this paper, we consider three enhancements in packed stencil design over previous works. First, the fact that the pattern of a character can be located anywhere within its enclosing projection region is exploited to facilitate flexible blank space sharing. Second, the use of multiple shaping apertures with different sizes is explored. Third, the use of overlapping shots for printing some characters is investigated. For the packed stencil design problem with flexible blank space sharing and multiple shaping apertures, two dynamic programming based algorithms are proposed, one allows overlapping shots and the other does not. Experimental results show that the proposed enhancement and the associated algorithm can significantly reduce the total shot count and hence improve the throughput of e-beam lithography.**

***Index Terms*—Character projection, E-beam lithography, Stencil design**

I. Introduction

There is intense interest and demand for increasing the throughput of e-beam lithography for use in advanced mask lithography and direct wafer writing applications. E-beam lithography has long been employed for mask writing but the write time is increasing due to the escalating mask pattern complexity. Besides, the semiconductor industry is also exploring e-beam direct write (EBDW) as a solution [1]–[3] for manufacturing chips at ever smaller process nodes for its superior resolution and depth of focus over optical lithography.

The character projection method is known to be an effective way to enhance the throughput of e-beam lithography. In the character projection method, complex patterns called characters can be directly printed onto a wafer/mask substrate [4], [5]. An e-beam writing system has a stencil which can hold a set of characters. Patterns in a circuit that correspond to a character in the stencil can be projected in one shot. However, other patterns that do not match any character need to be fractured into constituent rectangles. Then each constituent rectangle requires its own shot and has to be printed in the variable shaped beam (VSB) mode [6].

This work was supported in part by the Ministry of Science and Technology under grant MOST 103-2220-E-007-002.

Chris Chu is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50010

Wai-Kei Mak is with the Department of Computer Science, National Tsing Hua University, 101 Kuan Fu Rd. Sec. 2, Hsinchu, Taiwan 300 R.O.C.

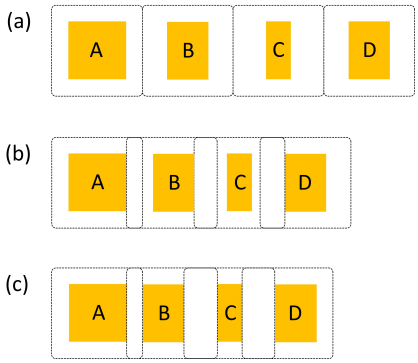

Fig. 1. (a) Four adjacent characters placed without overlapping their blank spaces. (b) The same characters are placed with their blank spaces overlapping to reduce the overall area as considered in [10]–[12]. (c) Flexible blank space sharing by relocating the character patterns within their projection regions to further reduce the overall area as considered in this paper and [13].

Traditionally, a standard stencil adopts a grid-based layout with pre-designated spots for characters [7], [8] as illustrated in Fig. 1(a). Each character pattern can be of any size or shape up to the maximum allowed size, which is dictated by the shaping aperture. Since there is a fixed number of N pre-designated character spots on the grid, one just needs to pick the N most beneficial characters and put them into these spots. While this arrangement is simple, it is too restrictive. It was pointed out in [8] and [9] that by carefully arranging and packing the different sized characters on a stencil, the final number of characters that can be put on a stencil can be greatly increased. It is because two adjacent characters can be placed with their blank spaces overlapping as illustrated in Fig. 1(b). This will allow more patterns on the wafer to be shot as characters leading to reduced write time and cost.

Packed stencil design has been studied in [10]–[12]. But they all overlooked the fact that the location of the pattern of a character is free to be adjusted within its enclosing projection region. For example, consider the characters in Fig. 1(a), we can reduce the total area occupied by the three characters even further compared to Fig. 1(b) through *flexible blank space sharing* by re-locating the pattern of each character within its own enclosing projection region as illustrated in Fig. 1(c). We were the first to point out this flexibility and proposed a nearly optimal approximation algorithm for character and stencil co-optimization in [13].

We note that all previous works including [13] are limited to the case that there is only a single shaping aperture, but the use of multiple shaping apertures with different sizes is possible [4], [14] as shown in Fig. 2. Since the enclosing projection region size of a character is determined by the shaping aperture size, using a smaller shaping aperture for smaller character patterns is helpful for packing more characters into the stencil

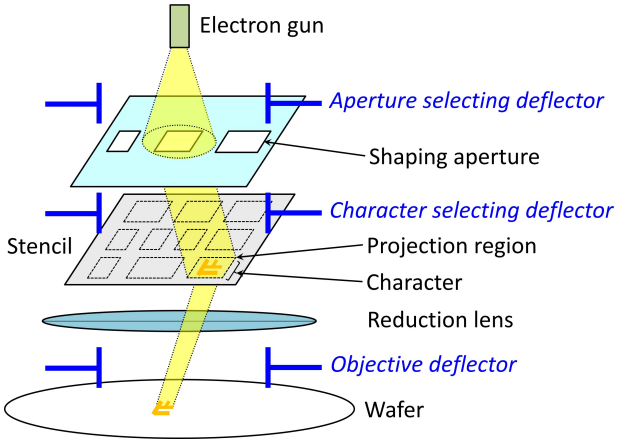

Fig. 2. A e-beam writing system with three shaping apertures.

and thus reducing the writing time further. Here we first address the packed stencil design problem with flexible blank space sharing and multiple shaping apertures. We propose a 3-stage algorithm MSA for the problem and show that it can effectively increase the number of packed characters on a stencil and significantly reduce the total shot count compared to the state of the art.

In addition, we consider the usage of overlapping shots for printing some characters. We show that selectively printing some characters using multiple shots is also an effective way to increase the number of packed characters on a stencil and reduce the total shot count. For the best result, we propose a second algorithm MSA-OS for flexible packed design which take advantage of the use of flexible blank space sharing, multiple shaping apertures, and overlapping shots at the same time.

Our main contributions are as follows.

1) We present an improved algorithm MSA for packed stencil design with multiple shaping apertures compared to our preliminary work [15]. In particular, the second and third stages of MSA are re-designed and lead to even better experimental results than [15].
2) To the best of our knowledge, this is the first work to investigate the usage of overlapping shots to print some characters in packed stencil design. We show that more characters can be put into the stencil as a result of intelligently using overlapping shots.
3) We design the MSA-OS algorithm for packed stencil design with multiple shaping apertures and overlapping shots. It yields 3.6 times and 2.3 times shot count reduction on average compared with [11] and [13], respectively, under two shaping apertures.

The rest of the paper is organized as follows. In Section II, we introduce some preliminary information. In Section III, some basic properties for packed stencil design with multiple shaping apertures are derived, and then the MSA algorithm is presented. In Section IV, we develop the theory of shot composition when using overlapping shots and present the MSA-OS algorithm. Finally, we present some experimental comparisons with [11] and [13] and a conclusion in Section V.

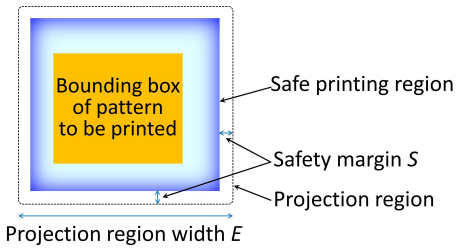

Fig. 3. The projection region and safe printing region for a character.

II. Preliminaries

Refer to the general e-beam machine in Fig. 2, it has multiple differently sized shaping apertures. The aperture selecting deflector can control the direction of the e-beam to shoot it through one of the shaping apertures. Subsequently, the character selecting deflector can direct the e-beam to shoot through any desired character on the stencil. Note that the dimensions of the projection region on the stencil when an e-beam is shot through a particular shaping aperture are dependent on the dimensions of the corresponding shaping aperture.

Due to scattering of electrons, features that are too close to the boundary of projection region may not be printed accurately. Hence, we define a safe printing region as follows.

Def. 1. *A safe printing region is the part of projection region that is at least a safety margin S away from the boundary as shown in Fig. 3.*

In order to print characters on the wafer with no loss of accuracy, the pattern of each character must lie within the safe printing region. In addition, the pattern of any character A cannot overlap with the e-beam projection region of another character B. Otherwise, part of character A would be printed erroneously when printing character B. However, the blank space of two neighboring characters may overlap and is desirable to overlap in order to reduce the total area occupied by the characters.

We assume that standard cells are implemented by the characters. To minimize wastage of stencil area, the heights of different character projection regions should all be set to $2S$ plus the standard cell image height. In addition, characters on the stencil should be arranged in a row-based manner and each character pattern is always placed within its projection region such that the top blank space and the bottom blank space are equal to S. In this way, the bottom blank space of a row of characters can completely overlap with the top blank space of the row of characters below as in Fig. 4 and we can pack the maximum possible number of rows of characters into the stencil. Finally, we assume that the blank space of a character can lie outside of the available character area of the stencil [9], [13] as in Fig. 4.

Next, we introduce the common notations used in the rest of the paper.

- K denotes the number of choices for character projection region widths, which is equal to the number of available shaping apertures.

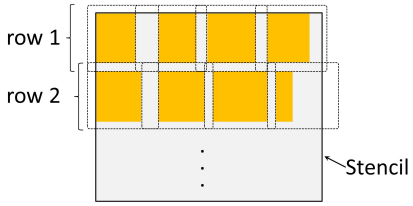

Fig. 4. Row-based stencil design.

- S denotes the safety margin from the safe printing region to the projection region boundary for an e-beam shot.
- W denotes the width of the available character area of the stencil.
- R denotes the number of character rows that can fit in the stencil.
- n denotes the number of different characters extracted from a circuit.
- w_c denotes the width of the pattern of character c. For simplicity, we will refer to the width of the pattern of a character as the character width in the rest of the paper.
- w_{min} and w_{max} denote the minimum character width and the maximum character width in the circuit.
- r_c denotes the number of occurrence of character c in a the circuit.
- n_{VSB_c} denotes the number of e-beam shots required to print character c by the VSB method.

III. Stencil Design with Flexible Blank Space Sharing and Multiple Shaping Apertures

In this section, we focus on the packed stencil design problem with multiple shaping apertures under flexible blank space sharing. The problem is formally stated below.

Problem 1. *Suppose the values of K, S, W, and R for an e-beam machine with multiple shaping apertures are given. A set of n characters extracted from a circuit and the values of w_c, r_c, and n_{VSB_c} for each character c are also given. We have to (i) determine K different character projection region widths, (ii) select the characters to be put on the stencil along with the character projection region width for each character, and (iii) pack the characters on the stencil in a row-based manner with flexible blank space sharing to maximize the total shot saving by character projection for printing the entire circuit under the following constraints.*
(C1) The pattern of each character must lie within the safe printing region of the character's projection region.
(C2) The pattern of each character cannot lie within the projection region of any other character.
(C3) The patterns of all characters must lie within the available character area of the stencil.

Note that in order to satisfy condition (C1), we must use a character projection region width no smaller than $w_c + 2S$ for character c because it requires the effective printing region to be at least as wide as w_c.

Before presenting our solution to the above flexible packed stencil design problem, we first show that it is not an easy problem.

Lemma 1. *The flexible packed stencil design problem with multiple shaping apertures is NP-complete.*

Proof. The flexible packed stencil design problem with multiple shaping apertures is a generalization of the single-row character packing problem in [13], which only considers a single shaping aperture. It is proved in [13] that the single-row character packing is NP-complete. Hence, our problem is also NP-complete. □

A. MSA Algorithm Outline

As the problem of flexible packed stencil design with multiple shaping aperture is very complicated, we present a 3-stage algorithm MSA here. Firstly, we propose a dynamic programming algorithm to determine K shaping aperture sizes (or equivalently, K projection region widths) and select a set of most beneficial characters that roughly can be packed on the stencil. Secondly, we distribute the selected characters to different rows and construct tight linear packing for each row. At last, we refine the solution by checking if any of the remaining characters can be added to the end of the tight linear packing of some row. The outline of the MSA algorithm is given in Algorithm 1.

Algorithm 1 MSA

Stage 1: Determine K projection region widths and select a set of characters to be put on the stencil by dynamic programming.
Stage 2: Assign the characters selected in Stage 1 to rows on the stencil and construct tight linear packing for each row.
Stage 3: Pack some of the remaining characters at the end of each row, if possible.

Stages 1 and 2 of our algorithm are based on the key concepts of tight linear packing and effective character width. So, we will first introduce them in Section III.B. Then, we will describe the details of Stage 1 of Algorithm 1 in Section III.C and the details of Stages 2 and 3 in Section III.D.

B. Tight Linear Packing and Effective Character Width

In this subsection, we introduce some definitions and derive some useful properties related to packed stencil design with multiple shaping apertures under flexible blank space sharing.

Def. 2. *A linear packing of a group of characters is a packing of all the given characters in a row with flexible blank space sharing satisfying constraints (C1) and (C2).*

Def. 3. *The width of a linear packing is the total span of the packing excluding the left blank space of the leftmost character and the right blank space of the rightmost character. (See Fig. 5.)*

In the following discussion, we assume that the characters in a linear packing are indexed from left to right by $1, 2, 3, \ldots$. And the enclosing projection region width for character c is denoted by E_c (see Fig. 6).

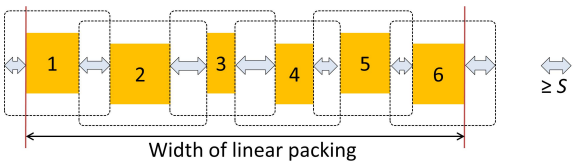

Fig. 5. A tight linear packing. Characters with even indexes are shifted down a bit to show the projection regions of all characters more clearly.

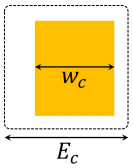

Fig. 6. A character c shown with its character width w_c and enclosing projection width E_c.

Def. 4. *A tight linear packing is a linear packing such that for any two neighboring characters i and $i+1$, the right blank space of character i and the left blank space of character $i+1$ are exactly equal and completely overlap. (See Fig. 5 for an example.)*

In a tight packing, the blank space on the two sides of each character, except the first and the last characters, is completely shared with its two neighboring characters. This motivates us to define the effective width of a character as its character width plus half of its blank space width.

Def. 5. *The effective width of a character c is defined as $w_c + (E_c - w_c)/2$, i.e., $\frac{w_c+E_c}{2}$.*

We may conveniently express the width of a tight linear packing and bound the width of an arbitrary linear packing in terms of the effective width of its characters as in Lemma 2 below.

Lemma 2. *For a linear packing P of m characters, let $W(P)$ denote the width of P, $w_i(P)$ and $E_i(P)$ denote the character width and enclosing projection width of the i-th character ($i = 1, 2, \ldots, m$), $s_0(P)$ denote the width of the left blank space of the first character, and $s_m(P)$ denote width of the right blank space of the last character in P.*
For a tight linear packing P_t,

$$W(P_t) = \sum_{i=1}^{m} \frac{w_i(P_t) + E_i(P_t)}{2} - \frac{s_0(P_t)}{2} - \frac{s_m(P_t)}{2} \quad (1)$$

For an arbitrary linear packing P_a,

$$W(P_a) \geq \sum_{i=1}^{m} \frac{w_i(P_a) + E_i(P_a)}{2} - \frac{s_0(P_a)}{2} - \frac{s_m(P_a)}{2} \quad (2)$$

Proof. For a tight linear packing P_t, the right blank space of the i-th character and the left blank space of the $(i+1)$-th character are exactly equal and completely overlap for $i = 1, 2, \ldots, m-1$. So, its width can be expressed as $W(P_t) = \sum_{i=1}^{m} w_i(P_t) + \sum_{i=1}^{m-1} s_i(P_t)$ where $s_i(P_t)$ denote the width of the right blank space of the i-th character in P_t.

For the i-th character in P_t, we have $s_{i-1}(P_t) + w_i(P_t) + s_i(P_t) = E_i(P_t)$ since its left and right blank spaces are equal to $s_{i-1}(P_t)$ and $s_i(P_t)$, respectively. Hence, $\sum_{i=1}^{m}(s_{i-1}(P_t) + w_i(P_t) + s_i(P_t)) = \sum_{i=1}^{m} E_i(P_t)$. It implies that $2 \times \sum_{i=1}^{m-1} s_i(P_t) = \sum_{i=1}^{m} E_i(P_t) - \sum_{i=1}^{m} w_i(P_t) - s_0(P_t) - s_m(P_t)$. So, the width of a tight linear packing P_t can be re-written as

$$\begin{aligned} W(P_t) &= \left(\sum_{i=1}^{m} w_i(P_t) + \sum_{i=1}^{m} E_i(P_t) - s_0(P_t) - s_m(P_t) \right) /2 \\ &= \sum_{i=1}^{m} \frac{w_i(P_t) + E_i(P_t)}{2} - \frac{s_0(P_t)}{2} - \frac{s_m(P_t)}{2}. \end{aligned}$$

For an arbitrary linear packing P_a, we let $g_i(P_a)$ be the width of the gap between the i-th and $(i+1)$-th character patterns in P_a for $i = 1, 2, \ldots, m-1$, and let $g_0(P_a)$ be $s_0(P_a)$ and $g_m(P_a)$ be $s_m(P_a)$. Then P_a's width can be expressed as $W(P_a) = \sum_{i=1}^{m} w_i(P_a) + \sum_{i=1}^{m-1} g_i(P_a)$. Note that $g_{i-1}(P_a) + w_i(P_a) + g_i(P_a) \geq E_i(P_a)$ for each character i. It implies that $2 \times \sum_{i=1}^{m-1} g_i(P_a) \geq \sum_{i=1}^{m} E_i(P_a) - \sum_{i=1}^{m} w_i(P_a) - s_0(P_a) - s_m(P_a)$. So,

$$W(P_a) \geq \sum_{i=1}^{m} \frac{w_i(P_a) + E_i(P_a)}{2} - \frac{s_0(P_a)}{2} - \frac{s_m(P_a)}{2}.$$

□

By applying Lemma 2, we can derive an upper bound of the difference between the width of a tight linear packing and the width of the minimum width linear packing and get Lemma 3. Note that the bound provided by this lemma is tighter than the corresponding one in our preliminary work [15].

Lemma 3. *Given a group of m characters with known enclosing projection widths, the width of any tight linear packing is less than $B_{max} - 2S$ away from the minimum width linear packing where B_{max} is the maximum blank space of a character.*

Proof. Consider a tight packing P_t and an optimal packing P^*.

By Equation (1) and Equation (2), we get

$$W(P_t) - W(P^*) \leq (s_0(P^*) + s_m(P^*) - s_0(P_t) - s_m(P_t))/2$$

as $\sum_{i=1}^{m} w_i(P_t) = \sum_{i=1}^{m} w_i(P^*)$.

Since $s_0(P^*) \leq E_1(P^*) - w_1(P^*) - S$ and $s_m(P^*) \leq E_m(P^*) - w_m(P^*) - S$, while $s_0(P_t) \geq S$ and $s_m(P_t) \geq S$, it implies that

$$\begin{aligned} & W(P_t) - W(P^*) \\ &\leq ((E_1(P^*) - w_1(P^*) - S) + (E_m(P^*) - w_m(P^*) - S) \\ &\quad -S - S)/2 \\ &\leq ((B_{max} - S) + (B_{max} - S) - S - S)/2 \\ &= B_{max} - 2S \end{aligned}$$

Hence, the width of P_t is less than $B_{max} - 2S$ away from the width of P^*. □

By Lemma 3, it is clear that if we can construct a tight linear packing, it will be a near optimal linear packing. Next, we propose an efficient way to construct a tight linear packing given m characters. Note that the way proposed here is more

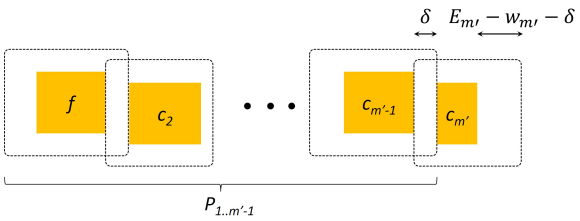

Fig. 7. Tight linear packing construction.

general than the one in our preliminary work [15] as any character can be used as the first character.

Lemma 4. *Given a group of m characters with known enclosing projection widths, a tight linear packing can be constructed with any character as the first character and the remaining characters ordered in increasing blank space, i.e., $E_i - w_i \leq E_{i+1} - w_{i+1}$ for $i = 2, \ldots, m-1$.*

Proof. We can prove the lemma by induction on m.
1. The base case that $m = 1$ is trivially true.
2. We show that if the statement is true for $m = m' - 1$, then it is also true for $m = m'$. Suppose the statement is true for $m = m' - 1$ and we have a group C of m' characters. Let f be an arbitrary character in C and $c_{m'}$ be the character in $C \setminus \{f\}$ with the largest blank space. By the assumption, we can construct a tight packing $P_{1..m'-1}$ for $C \setminus \{c_{m'}\}$ with f as the first character and the remaining characters ordered in increasing blank space, i.e., $E_i - w_i \leq E_{i+1} - w_{i+1}$ for $i = 2, \ldots, m'-2$. Now we can append $c_{m'}$ to the right end of $P_{1..m'-1}$ in such a way that the left blank space of $c_{m'}$ completely overlaps with the right blank space of $c_{m'-1}$ as shown in Fig. 7. It follows that the left blank space of $c_{m'}$ must be no smaller than the safety margin S since the right blank space of $c_{m'-1}$ is at least S. But it remains to show that the resultant right blank space of $c_{m'}$ is also at least S for the constructed packing to be a feasible tight linear packing. Let δ denote the left blank space of $c_{m'}$ (equivalently, the right blank space of $c_{m'-1}$). The resultant right blank space of the m'-th character is equal to $E_{m'} - w_{m'} - \delta$. Since $E_{m'} - w_{m'} \geq E_{m'-1} - w_{m'-1}$, so $E_{m'} - w_{m'} - \delta \geq E_{m'-1} - w_{m'-1} - \delta \geq S$. Hence, the constructed packing is a tight linear packing and satisfies $E_i - w_i \leq E_{i+1} - w_{i+1}$ for $i = 2, \ldots, m'-1$. □

C. Projection Region Width and Character Selection by DP

In Stage 1 of Algorithm 1, in order to determine the projection region widths and characters to be put into the stencil, we merge all rows of the stencil into a single row and use dynamic programming to maximize the overall shot saving subject to a total effective character width constraint.

Assume the characters are sorted in increasing order of width. We define $\mathcal{S}[e, i, k, w]$ as the maximum shot saving using at most k different projection region widths for printing a subset of the first i characters such that the largest projection region width is e and the total effective width of the subset is at most w. The ranges of the parameters are $w_{min} + 2S \leq e \leq w_{max} + 2S$, $0 \leq i \leq n$, $1 \leq k \leq K$, and $0 \leq w \leq RW$.

$\mathcal{S}[e, i, k, w]$ can be expressed recursively as follow:

$$\mathcal{S}[e, 0, k, w] = 0 \quad \text{for all } e, k, w$$
$$\mathcal{S}[e, i, k, 0] = 0 \quad \text{for all } e, i, k$$
$$\mathcal{S}[e, i, k, w] = \max \begin{cases} \mathcal{S}[e, i-1, k, w] \\ \begin{cases} r_i(n_{VSB_i} - 1) + \mathcal{S}[e, i-1, k, w - \frac{w_i+e}{2}] \\ \qquad \text{if } w_i + 2S \leq e \text{ and } \frac{w_i+e}{2} \leq w \\ 0 \qquad \text{otherwise} \end{cases} \\ \begin{cases} \mathcal{S}[w_i + 2S, i, k-1, w] \\ \qquad \text{if } k > 1 \text{ and } w_i + 2S \leq e \\ 0 \qquad \text{otherwise} \end{cases} \end{cases}$$
$$\text{for all } e, i \neq 0, k, w \neq 0$$

In the recursive expression above, $\mathcal{S}[e, i, k, w]$ is the maximum over three cases. In the first case, character i is skipped. In the second case, character i is selected. It results in a shot saving of $r_i(n_{VSB_i} - 1)$ and a reduction of remaining effective width by $\frac{w_i+e}{2}$. In the third case, another projection region width of $w_i + 2S$ is used.

One key insight in the above recursion is that each time we decrease e, we can safely skip over all e not equal to $w_i + 2S$ for any $1 \leq i \leq n$ for the following reason. First, a projection region width smaller than $w_1 + 2S$ is not useful at all since it is too small for printing any character. Second, using a projection region width $e (> w_1 + 2S)$ not equal to $w_i + 2S$ for any $1 \leq i \leq n$ offers no advantage since a projection region width $e' = \max\{w_i + 2S : 1 \leq i \leq n \text{ and } w_i + 2S < e\}$ is always better than e. e' can print the same characters as e and results in a smaller effective width than e.

The maximum possible shot saving when we can select K different projection region widths subject to a total effective width constraint of RW is given by $\mathcal{S}^* = \max_{1 \leq i \leq n} \{\mathcal{S}[w_i + 2S, n, K, RW]\}$ since it is sufficient to consider all $w_i + 2S$ for $1 \leq i \leq n$ as the largest project region width selected as explained above. It is clear that the above recursion to compute the values of $\mathcal{S}[e, i, k, w]$ for all e, i, k, w can be implemented as a dynamic program. However, as we will see in Section V, the memory requirement for typical problem instances can be up to tens of gigabytes. In other words, this dynamic programming formulation is not practical.

In order to reduce the memory requirement, we take advantage of the fact that many characters have the same width. Instead of considering each character separately, we group characters of the same width together. Let G_j be the group of characters of width w_j. Assume that the characters within each group are sorted in decreasing order of shot saving.

We define $\mathcal{S}'[e, j, k, w]$ as the maximum shot saving using at most k different projection region widths for printing some subset of characters in the first j groups such that the largest projection region width is e and the total effective width of the subset is at most w.

$\mathcal{S}'[e, j, k, w]$ can be expressed recursively as follow:

$$\mathcal{S}'[e,0,k,w] = 0 \quad \text{for all } e,k,w$$
$$\mathcal{S}'[e,j,k,0] = 0 \quad \text{for all } e,j,k$$
$$\mathcal{S}'[e,j,k,w] = \max \begin{cases} \mathcal{S}'[e,j-1,k,w] \\ \max_{1\le i\le |G_j|} \mathcal{R}(e,j,k,w,i) \\ \begin{cases} \mathcal{S}'[w_j+2S,j,k-1,w] \\ \qquad \text{if } k>1 \text{ and } w_j+2S\le e \\ 0 \qquad \text{otherwise} \end{cases} \end{cases}$$
$$\text{for all } e, j\neq 0, k, w\neq 0$$

where $\mathcal{R}(e,j,k,w,i)$ is the shot saving if the first i characters in G_j (i.e., the i highest shot saving characters in G_j)[1] are included in the stencil. Let $G_j[i]$ be the set of the first i characters in G_j. Then $\mathcal{R}(e,j,k,w,i)$ is given by the following expression:

$$\mathcal{R}(e,j,k,w,i) = \begin{cases} \sum_{c\in G_j[i]} r_c(n_{VSB_c}-1) + \mathcal{S}'[e,j-1,k,w-i\times\frac{w_j+e}{2}] \\ \qquad \text{if } w_j+2S\le e \text{ and } i\times\frac{w_j+e}{2}\le w \\ 0 \qquad \text{otherwise} \end{cases}$$

The three cases for $\mathcal{S}'[e,j,k,w]$ are similar to those for $\mathcal{S}[e,i,k,w]$ except that a set of i characters in G_j is selected instead of a single character in the second case. Note that $\mathcal{S}'^{*} = \max_{1\le j\le g}\{\mathcal{S}'[w_j+2S,g,K,RW]\}$ where g = number of groups gives the maximum possible shot saving when we can select K different projection region widths subject to a total effective width constraint of RW since the first g groups of characters is equivalent to all n characters. In other words, the values of $\mathcal{S}'^{*}$ and $\mathcal{S}^{*}$ are the same.

D. Tight Packing Construction

For Stages 2 and 3 of Algorithm 1, which tightly pack the selected characters and some remaining characters into the stencil, we introduce a procedure (Procedure 1) that is different from and more sophisticated than the one in our preliminary work [15]. Lines 1-17 of Procedure 1 correspond to Stage 2 and Lines 18-21 correspond to Stage 3. We assume that $\mathcal{E}$ is the set of K projection region widths selected in Stage 1 of Algorithm 1.

For a selected character c, it should always use the smallest available character projection region width no less than w_c+2S in order to reduce wastage of stencil area. Hence, projection region width E_c for each character is set as in Line 2 of Procedure 1. Since the characters were selected based on the assumption that there is a single row of width RW rather than R rows of width W, there is a chance that not all selected characters can be packed onto the stencil. So, in order to maximize the final shot saving after packing, we will pack a character with higher efficiency onto the stencil with higher priority as in Lines 5-16. The *efficiency* θ_c of a character c is defined as its shot saving per unit effective width, i.e., $\theta_c = 2r_c(n_{VSB_c}-1)/(w_c+E_c)$.

[1]It will not lose optimality here because if an optimal decision requires i characters from G_j, then it must be the i highest shot saving characters in G_j since all characters in G_j have the same width.

Procedure 1 Character Packing for MSA

1: **for** each selected character c **do**
2: Set E_c to be the smallest value in $\mathcal{E}$ s.t. $E_c \ge w_c+2S$;
3: c's efficiency $\theta_c = 2r_c(n_{VSB_c}-1)/(w_c+E_c)$;
4: **end for**
5: Sort all selected characters in decreasing order of efficiency;
6: $j=1$;
7: **for** each character c in sorted order **do**
8: **if** c can be inserted into row j according to Equation (3) **then**
9: Insert c into row j;
10: **else if** $j<R$ **then**
11: Insert c into row $j+1$;
12: $j=j+1$;
13: **else**
14: break;
15: **end if**
16: **end for**
17: For each row, construct a tight packing by putting its maximum blank space character f first and then other characters in increasing blank space, and setting the left blank space of f to E_f-w_f-S;
18: Sort the rows in increasing order of remaining usable width;
19: **for** each row j in sorted order **do**
20: Tightly pack at the end of row j a remaining character with largest possible shot saving without exceeding the stencil width, if possible;
21: **end for**

We utilize a simple condition to check whether a set of characters C can be packed within a row without exceeding the stencil width W. Suppose

$$\sum_{c\in C}\frac{w_c+E_c}{2} - \frac{E_f-w_f}{2} \le W \tag{3}$$

where f is the maximum blank space character in C, then all characters in C can be packed within a row of the stencil. The above is a sufficient condition since Lemma 4 and Equation (1) imply that there exists a tight packing of C with width no more than $\sum_{c\in C}(w_c+E_c)/2-(E_f-w_f-S)/2-S/2$ if we use f as the first character and set the width of its left blank space to E_f-w_f-S.

We note that the packing procedure above is similar in spirit to that of the CASCO algorithm we introduced in [13]. But it is more general here because E_c is not necessarily the same for all character c. In addition, here we improved the chance of inserting an extra character at the end of the rows by processing the rows in increasing order of remaining usable width as in Lines 18-21.

We illustrate Procedure 1 by an example. Consider a stencil with 2 rows, $W=20$, and $S=1$. Suppose 2 projection region widths of 15 and 10 are selected. Besides, suppose 5 characters indexed 1 to 5 with widths $w_1=7, w_2=6, w_3=12, w_4=7, w_5=10$ and efficiencies $\theta_1=5, \theta_2=4, \theta_3=3, \theta_4=2, \theta_5=1$ are selected. Therefore, Line 2 will set

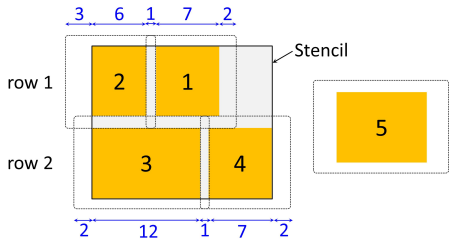

Fig. 8. An example illustrating Procedure 1.

$E_1 = 10, E_2 = 10, E_3 = 15, E_4 = 10$, and $E_5 = 15$. Line 5 will sort the characters in the order of 1, 2, 3, 4, 5. Line 8 will insert character 1 and then character 2 into row 1. After that, character 3 cannot be inserted into row 1 and it will be inserted into row 2. Line 8 cannot insert character 4 into row 2 (as $(12+15)/2+(7+10)/2-(15-12)/2 = 20.5 > 20$). Hence the for loop in Lines 7-16 is exited. After constructing the tight packings for all rows in Line 17, the rows will be sorted in the order of 1, 2 in Line 18. Finally, character 4 will be packed at the end of row 2 in Line 20. The final stencil design is shown in Figure 8.

E. Time and Space Complexity Analysis

First, we consider the time complexity for Stage 1 of the MSA algorithm. It takes $O(n \log n)$ time to sort all the characters in increasing width. If the memory saving technique is not applied, it is apparent that computing $\mathcal{S}[e, i, k, w]$ for $w_{min} + 2S \leq e \leq w_{max} + 2S$, $0 \leq i \leq n$, $1 \leq k \leq K$, and $0 \leq w \leq RW$ by dynamic programming takes $O(\pi nKRW)$ time where $\pi = w_{max} - w_{min} + 1$. If the memory saving technique is applied, it requires some extra time on sorting each group G_j in decreasing shot saving. The extra sorting time is bounded by $O(\pi n \log n)$. So, without the memory saving technique, Stage 1 takes $O(n \log n + \pi nKRW)$ time which is $O(\pi nKRW)$ in practice. With the memory saving technique, it takes $O(\pi n \log n + \pi nKRW)$ time which is also $O(\pi nKRW)$ in practice. Second, the time spent on Stages 2 and 3 (i.e., Procedure 1) is dominated by the time for sorting the selected characters in decreasing efficiency, i.e., $O(n \log n)$ time. As a result, the overall time complexity of MSA is $O(\pi nKRW)$.

We analyze the space complexity of both the original version and the memory-efficient version of MSA below. For the original version, the only major memory requirement is to store $\mathcal{S}[e, i, k, w]$ for all e, i, k, w. So its space complexity is $O(\pi nKRW)$. For the memory-efficient version, note that $\mathcal{R}(e, j, k, w, i)$ is a function and we do not store its values. So the only major memory requirement is to store $\mathcal{S}'[e, j, k, w]$ for all e, j, k, w. As the number of groups is bounded by π, the space complexity is $O(\pi^2 KRW)$. As the number of groups are typically at least tens of times less than the number of characters, the memory requirement of the memory-efficient version is much less than the original version. We show in Section V that $\mathcal{S}'[e, j, k, w]$ can be stored in less than 1 gigabyte in practice.

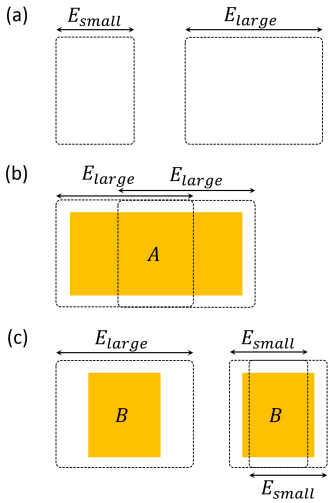

Fig. 9. (a) Two fixed projection widths of an e-beam writing system. (b) Character A can be printed in two shots but not one. (c) Character B can be printed in one shot or two shots.

IV. Using Overlapping Shots

Older generations of e-beam machine for mask writing or direct writing of wafers did not support overlapping shots. However, employing new model-based mask data preparation (MB-MDP) and new generation e-beam machines allow writing with overlapping e-beam shots. With careful simulation, MB-MDP enables shot overlapping, dose modulation of individual e-beams and character projection to draw complex shapes accurately with less shot count [16], [17].

The benefits of allowing some characters to be printed in more than one shot are illustrated in Fig. 9. Suppose an e-beam machine has two shaping apertures and can produce shots of widths E_{small} and E_{large} (Fig. 9(a)). It is clear that there is no way to print the wide character A in Fig. 9(b) using only one shot. However, A can be printed using two overlapping shots as in Fig. 9(b).[2] In Fig. 9(c), though character B can be printed in one shot of width E_{large}, this option requires a significant amount of blank space on both sides of the character pattern to be reserved on the stencil. On the other hand, character B can be printed in two shots of projection width E_{small} each such that a minimum amount of blank space on both sides of the character pattern needs to be reserved on the stencil as shown on the right of Fig. 9(c). So, in some cases, we may want to print B in two shots of width E_{small} each instead of a single shot of width E_{large}.

In this section, we address the flexible packed stencil design problem with multiple shaping apertures and overlapping shots. We consider Problem 2 below which assumes that we use K shaping apertures and the K different shaping aperture sizes (hence K projection region widths) are given. If the K projection region widths are not given, they can be determined by enumerating all combinations of K distinct projection widths from $w_{min} + 2S$ to $w_{max} + 2S$.

[2]We note that the overlapped part of the two e-beam shots will receive a higher dose and cause the patterns within the overlapped part to be printed larger. So, the character pattern must be pre-adjusted accordingly by careful simulation in order to obtain the intended final shapes accurately.

Problem 2. *Suppose the values of S, W, and R for an e-beam machine with a set $\mathcal{E}$ of K fixed shaping aperture sizes are given. A set of n characters extracted from a circuit and the values of w_c, r_c, and n_{VSB_c} for each character c are also given. Each character may be printed by one shot or two overlapping shots by the character projection method. We have to (i) select the characters to be put on the stencil along with the character projection region width and shot composition for each character, and (ii) pack the characters on the stencil in a row-based manner with flexible blank space sharing to maximize the total shot saving by character projection for printing the entire circuit under the following constraints.*
(C1) The pattern of each character must lie within the safe printing region of the character's projection region.
(C2) The pattern of each character cannot lie within the projection region of any other character.
(C3) The patterns of all characters must lie within the available character area of the stencil.

We propose a 3-stage algorithm MSA-OS below (Algorithm 2) for Problem 2.

Algorithm 2 MSA-OS

Stage 1: Select a set of characters to be put on the stencil and determine the shot composition of each character by dynamic programming.
Stage 2: Assign the resultant characters of Stage 1 to rows on the stencil and construct tight linear packing for each row.
Stage 3: Pack some of the remaining characters at the end of each row, if possible.

Stages 2 and 3 of Algorithm 2 are similar to those of Algorithm 1 with some modification that will be explained in Section IV.B. In Stage 1, we use dynamic programming to simultaneously select a set of characters and determine the shot composition of each selected character such that the resultant characters can roughly be packed on the stencil to maximize the potential shot saving.

A. Character Selection and Shot Composition Determination

For simplicity, here it is assumed that no more than two shots are used to compose each character.[3] Before describing the dynamic program for character selection and shot composition determination, we first present a useful lemma related to shot composition of character to best utilize it. It shows how shot composition of a character c can help minimize the amount of blank space reserved on either side of the character pattern on the stencil to the safety margin S.

Lemma 5. *Suppose $w_c > E' - 2S$, $w_c > E'' - 2S$, and $w_c \leq E' + E'' - 4S$. Then we can print character c by combining two shots with projection widths E' and E''. Moreover, we can do it in such a way that the enclosing width of the two shots becomes $w_c + 2S$ and the effective width of c on the stencil becomes $w_c + S$.*

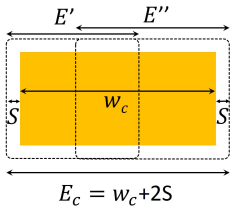

Fig. 10. A shot composition for character c that results in an enclosing width of $w_c + 2S$.

Proof. Note that $w_c > E' - 2S$ and $w_c > E'' - 2S$ imply that the character width of c exceeds the safe printing region width of either shot, so either shot alone is insufficient to print character c in its entirety. However, by arranging the safe printing regions of two such shots side-by-side, we can obtain a single safe printing region of width up to $(E' - 2S) + (E'' - 2S) = E' + E'' - 4S$. Hence, character c with $w_c \leq E' + E'' - 4S$ can be printed by composing two shots with projection widths E' and E''.

Moreover, $w_c > E' - 2S$ and $w_c > E'' - 2S$ imply that $E' \leq w_c + 2S$ and $E'' \leq w_c + 2S$. Hence, it is possible to partially overlap the two shots as in Fig. 10 to obtain a single safe printing region of width w_c in an enclosing width of $w_c + 2S$. By definition, the effective width of character c under an enclosing width of $w_c + 2S$ is $(w_c + (w_c + 2S))/2 = w_c + S$. □

To select the appropriate characters and determine their shot compositions for packing onto the stencil, we merge all row of the stencil into a single row and use dynamic programming to maximize the overall shot saving subject to a total effective character width constraint.

We group the characters of the same width together. Let G_j be the group of characters of width w_j. Let the characters within group G_j be $c_{j1}, c_{j2}, \ldots, c_{j|G_j|}$. Subsequently, when we say the first p characters of G_j, we mean $c_{j1}, c_{j2}, \ldots, c_{jp}$. Let $\mathcal{E}$ denote the given set of K different projection widths.

We define $\hat{\mathcal{S}}[j, w]$ as the maximum shot saving using the projection region widths in $\mathcal{E}$ for printing some subset of characters in the first j groups such that the total effective width of the subset is at most w.

$\hat{\mathcal{S}}[j, w]$ can be expressed recursively as follow:

$$\hat{\mathcal{S}}[0, w] = 0 \quad \text{for all } w$$
$$\hat{\mathcal{S}}[j, 0] = 0 \quad \text{for all } j$$
$$\hat{\mathcal{S}}[j, w] = \max \begin{cases} \hat{\mathcal{S}}[j-1, w] \\ \max_{1 \leq i \leq |G_j|} \hat{\mathcal{R}}(j, w, i) \end{cases} \quad \text{for all } j \neq 0, w \neq 0$$

where $\hat{\mathcal{R}}(j, w, i)$ is the maximum possible shot saving using the projection region widths in $\mathcal{E}$ for printing some subset of characters in the first j groups such that the total effective width of the subset is at most w and exactly i characters in G_j are printed.

We consider how to determine $\hat{\mathcal{R}}(j, w, i)$. If we want to

[3]The ideas presented here can be readily extended when we use no more than three shots (or even more) for each character. But using more than two shots to compose a character may not be necessary in practice especially when there are multiple shaping apertures.

print any character in G_j in one shot, we should always use the smallest feasible projection region width available to minimize the resultant effective character width. So, let E_j^1 be the smallest projection region width in $\mathcal{E}$ that is no smaller than $w_j + 2S$. If all given K projection region widths are smaller than $w_j + 2S$, then $E_j^1 = \infty$. Let E_j^2 be the largest projection region width in $\mathcal{E}$ such that $w_j > E_j^2 - 2S$ and $w_j \leq 2E_j^2 - 4S$. By Lemma 5, only if there exists such E_j^2, we consider printing some characters in G_j in two shots using projection region width E_j^2. We use a Boolean value TWO_j to indicate if there exists such E_j^2 or not.

If TWO_j is true, the advantage of printing a character in G_j by two shots instead of one is that effective width of the character will be reduced from $\frac{w_j+E_j^1}{2}$ to w_j+S by Lemma 5. But the disadvantage is that the shot saving for it will be reduced by r_c from $r_c(n_{VSB_c} - 1)$ to $r_c(n_{VSB_c} - 2)$. If i characters in G_j are to be printed and TWO_j is true, we may choose to print $i' (\leq i)$ of them by one shot each and the others by two shots each, in which case the total effective width of these i characters denoted by $w_j(i, i')$ will be equal to $i' \times \frac{w_j+E_j^1}{2} + (i - i') \times (w_j + S)$.

By the analysis above, $\hat{\mathcal{R}}(j, w, i)$ is given by the following expression:

$$\hat{\mathcal{R}}(j, w, i) = \begin{cases} \max_{0 \leq i' \leq i \wedge w_j(i,i') \leq w} \{\hat{T}_j(|G_j|, i, i') \\ \quad + \hat{\mathcal{S}}[j-1, w - w_j(i, i')]\} \\ \qquad \text{if } TWO_j \text{ is true} \\ \sum_{c \in G_j[i]} r_c(n_{VSB_c} - 1) + \hat{\mathcal{S}}[j-1, w - i \times \frac{w_j + E_j^1}{2}] \\ \qquad \text{if } TWO_j \text{ is false and } i \times \frac{w_j+E_j^1}{2} \leq w \\ 0 \qquad \text{otherwise} \end{cases}$$

where $\hat{T}_j[|G_j|, i, i']$ is the maximum shot saving of printing i characters in G_j such that i' of them are printed by one shot each and $i - i'$ of them are printed by two shots each.

$\hat{T}_j[|G_j|, i, i']$ can in turn be pre-computed by dynamic programming. Let $\hat{T}_j[p, i, i']$ denote the maximum shot saving of printing i characters from the first p characters in G_j such that i' of them are printed by one shot each and $i - i'$ of them are printed by two shots each ($0 \leq p \leq |G_j|$, $0 \leq i \leq p$, $0 \leq i' \leq i$). We have

$$\hat{T}_j[0, i, i'] = 0 \quad \text{for all } i, i'$$
$$\hat{T}_j[p, 0, 0] = 0 \quad \text{for all } p$$
$$\hat{T}_j[p, i, i'] = \max \begin{cases} \begin{cases} \hat{T}_j(p-1, i, i') & \text{if } i \leq p-1 \\ 0 & \text{otherwise} \end{cases} \\ \begin{cases} r_{c_{jp}}(n_{VSB_{c_{jp}}} - 1) + \hat{T}_j(p-1, i-1, i'-1) \\ \qquad \text{if } p \geq 1, i \geq 1, i' \geq 1 \\ 0 \qquad \text{otherwise} \end{cases} \\ \begin{cases} r_{c_{jp}}(n_{VSB_{c_{jp}}} - 2) + \hat{T}_j(p-1, i-1, i') \\ \qquad \text{if } p \geq 1, i \geq 1, i > i' \\ 0 \qquad \text{otherwise} \end{cases} \end{cases}$$

In the recursive expression for $\hat{T}_j[p, i, i']$, we have three cases since character c_{jp} can be skipped (case 1), chosen and printed in one shot (case 2), or chosen and printed in two shots (case 3).

B. Tight Packing Construction

In Stage 1 of Algorithm 2, we compute $\hat{\mathcal{S}}[g, RW]$ where g is the number of groups. The characters selected and the shot composition for each selected character are recorded. Then Stages 2 and 3 of Algorithm 2 for packing the characters into the stencil are performed by Procedure 2. For a selected character c composed of one shot, we should use the smallest projection region width in $\mathcal{E}$ that is no smaller than $w_c + 2S$ which corresponds to Line 3 of Procedure 2. On the other hand, for a selected character c composed of two shots, we should utilize the minimum possible enclosing width which is $w_c + 2S$ as in Line 5 of Procedure 2. As a result, the efficiency for a character c printed in one shot is $\theta_c = 2r_c(n_{VSB_c} - 1)/(w_c + E_c)$. The efficiency for a character c printed in two shots is $\theta_c = r_c(n_{VSB_c} - 2)/(w_c + S)$. The rest of the character packing procedure for MSA-OS is similar to that for MSA.

Procedure 2 Character Packing for MSA-OS

1: **for** each selected character c **do**
2: **if** c is composed of one shot **then**
3: Set E_c to be the smallest value in $\mathcal{E}$ s.t. $E_c \geq w_c + 2S$;
4: c's efficiency $\theta_c = 2r_c(n_{VSB_c} - 1)/(w_c + E_c)$;
5: **else**
6: $E_c = w_c + 2S$;
7: c's efficiency $\theta_c = r_c(n_{VSB_c} - 2)/(w_c + S)$;
8: **end if**
9: **end for**
10: The rest of the procedure is exactly the same as Lines 5-21 of Procedure 1.

C. Determination of Projection Widths

Algorithm MSA-OS assumes that K different projection widths are already given. If not, we can try all combinations of K distinct projection widths from $w_{min} + 2S$ to $w_{max} + 2S$. Note that $\hat{T}_j[|G_j|, i, i']$ in Section IV.A just has to be pre-computed once since it is not related to what projection region widths we use. In other words, when we change to another set of K projection region widths, we do not need to re-compute $\hat{T}_j[|G_j|, i, i']$ again.

D. Time and Space Complexity Analysis

We analyze the runtime of the MSA-OS algorithm. Similar to the MSA algorithm, Stages 2 and 3 of MSA-OS take $O(n \log n)$ time. For Stage 1, it takes $O(n^3)$ time to compute $\hat{T}_j[|G_j|, i, i']$ for each group G_j. So, computing $\hat{T}_j[|G_j|, i, i']$ for all g groups take $O(gn^3)$ time. Computing each $\hat{\mathcal{R}}(j, w, i)$ takes $O(n)$ time, hence computing each $\hat{\mathcal{S}}[j, w]$ takes $O(n^2)$ time. As a result, computing $\hat{\mathcal{S}}[j, w]$ for $0 \leq j \leq g$ and $0 \leq w \leq RW$ takes $O(gn^3 + gRW \times n^2)$ time which is $O(gRWn^2)$ in practice. The overall runtime of MSA-OS is $O(gRWn^2)$.

If the K projection region widths need to be determined by enumeration, MSA-OS should be called $C(\pi, K)$ times where $\pi = w_{max} - w_{min} + 1$, as there are $C(\pi, K)$ combinations of K distinct projection widths from $w_{min} + 2S$ to $w_{max} + 2S$. Thus the total runtime is $C(\pi, K) \times O(gRWn^2) = O(\pi^K \times gRWn^2)$.

We analyze the space complexity of MSA-OS below. Note that $\hat{\mathcal{R}}(j, w, i)$ is a function and we do not store its values. The memory requirement to store $\hat{\mathcal{S}}[j, w]$ for all j, w is $O(gRW)$. The memory requirement to store $\hat{T}_j[p, i, i']$ for all j, p, i, i' is $O(gn^3)$. Hence the space complexity is $O(g(RW + n^3))$.

V. Experimental Results and Conclusions

We implemented our approaches MSA and MSA-OS in C and obtained the executable codes of E-BLOW [11] and CASCO [13] for comparison. All experiments were done on a Linux server powered by a 2.67 GHz Intel processor with 47 GB of memory. For each circuit, the result and runtime of MSA-OS were obtained by trying all combinations of K distinct projection widths from $w_{min} + 2S$ to $w_{max} + 2S$.

In the first experiment, benchmarks 1D-1 to 1D-4 from [11] were used. The available character area of the stencil is $1000\mu m \times 1000\mu m$, and the number of character candidate in each benchmark is 1000. Recall that E-BLOW assumes the left and right blank spaces of each character to be fixed while MSA and MSA-OS consider the re-location of the pattern of each character within the safe printing region. We set the safety margin S to the minimum left/right blank space of the original characters in each benchmark. Besides, E-BLOW and CASCO assume that there is only a single shaping aperture and hence a single projection region width (i.e., $K = 1$). For MSA and MSA-OS, we tried $K = 1$ and $K = 2$. Table I reports the comparison on total shot count, number of characters put on the stencil, and runtime. It also reports the shot count when using VSB only for reference.

Table I shows that even for $K = 1$ (i.e., all characters must use the same projection region width), MSA can reduce the shot count by $29.328\times$, $1.767\times$ and $1.107\times$ over VSB only, E-BLOW and CASCO, respectively. By allowing each character to be printed by two shots, MSA-OS can further cut the shot count almost by half. MSA-OS ($K = 1$) has better shot count than MSA ($K = 1$) because overlapping shots is enabled. As a result, some characters that cannot be printed in MSA can be printed in MSA-OS. Also, a smaller projection region width may be used and hence more characters may be packed. The significant improvement over E-BLOW is partially because it performs only simple blank space sharing while MSA and MSA-OS perform flexible blank space sharing. Besides, E-BLOW does not attempt to find the optimal projection region width while our algorithms do.

Next, if we use two different sized shaping apertures (i.e., $K = 2$), a huge shot count reduction over using single shaping aperture can be obtained for both MSA and MSA-OS. MSA with $K = 2$ results in $3.02\times$ and $1.89\times$ shot count reduction compared to E-BLOW and CASCO, respectively. MSA-OS with $K = 2$ results in as much as $3.60\times$ and $2.25\times$ shot count reduction compared to E-BLOW and CASCO, respectively.

TABLE III
The values of the projection region widths determined by MSA and MSA-OS.

	w_{min}	w_{max}	MSA			MSA-OS		
	$+2S$	$+2S$	$K=1$	$K=2$	$K=3$	$K=1$	$K=2$	$K=3$
1D-1	13	37	37	35,37		35	15,37	
1D-2	15	39	39	29,39		32	25,39	
1D-3	15	41	41	27,41		34	19,33	
1D-4	15	43	43	29,43		25	19,33	
1D-1h	13	37	37	25,37	21,29,37	23	19,29	17,25,33
1D-2h	15	39	39	27,39	21,29,39	23	19,27	19,27,34
1D-3h	15	41	41	27,41	23,33,41	24	17,27	17,27,33
1D-4h	15	43	43	27,43	23,33,43	25	17,27	17,21,37

Note that for benchmark 1D-1, although MSA ($K = 2$), MSA-OS ($K = 1$) and MSA-OS ($K = 2$) use all 1000 characters, the shot count of MSA-OS ($K = 1$) is higher than those of the other two. The reason is that while MSA-OS ($K = 1$) can print all 1000 characters, 278 (which equals $10696 - 10418$) of the characters are printable only if overlapping shots (i.e., 2 shots) are used. Hence, the shot count is higher by 278.

For more testing, we generated some harder benchmarks (1D-1h to 1D-4h). We generated 200 extra character candidates into each of the original benchmarks while keeping the same stencil size. Table II reports the results of MSA and MSA-OS with $K = 1$, $K = 2$, and $K = 3$. It also reports the shot count when using VSB only for reference. As expected, the shot count reduction and the number of characters that can be put on the stencil by MSA increase with the value of K. And the greatest reduction occurs when K switches from 1 to 2. Besides, MSA-OS is significantly better than MSA. For $K = 1, 2$ and 3, MSA-OS reduces the shot count over MSA by $2.19\times$, $1.44\times$ and $1.20\times$, respectively. However, MSA-OS is much slower than MSA.

The values of the projection region widths determined by MSA and MSA-OS for all benchmarks are listed in Table III. We note in order not to miss out the widest characters, MSA always uses the largest projection region width values. However, MSA-OS often skips the larger projection region width values as widest characters can be covered using two shots. Hence, MSA-OS can pack more characters on the stencil and achieve higher shot saving.

We show in Table IV the memory requirement of MSA. The proposed character grouping technique in Section III.C can reduce the memory requirement of the dynamic program by roughly $40\times$. The resultant memory requirement after adopting the technique is less than 1 GB in each case. The memory requirement of MSA-OS is insignificant (far less than 1 GB).

Finally, we note that our algorithms also work for multi-beam direct write system [1], where multiple beam columns furnished with their own stencils write different regions of a wafer in parallel. As identical dies are typically manufactured on a wafer, the stencil design for all beam columns in a multi-beam system should be the same and is not different from a single beam system.

TABLE I
COMPARISON OF E-BLOW [11], CASCO [13], MSA AND MSA-OS.

	VSB only	E-BLOW [11]			CASCO [13]			MSA ($K=1$)			MSA ($K=2$)			MSA-OS ($K=1$)			MSA-OS ($K=2$)		
	#shots	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)
1D-1	770543	29536	934	1.91	14491	970	<0.005	12553	983	6.08	10418	1000	17.06	10696	1000	56.40	10418	1000	939.46
1D-2	770543	44544	863	1.74	27812	898	<0.005	24742	913	5.86	10734	997	16.06	13939	996	51.93	10516	999	882.14
1D-3	770543	78704	758	2.35	57698	791	<0.005	52902	807	5.83	27192	909	16.76	21219	973	54.27	18114	986	913.33
1D-4	770543	107460	699	2.96	79930	734	<0.005	75388	746	6.08	42651	846	18.78	31366	917	53.30	27750	938	974.37
Normalized	29.328	1.767	0.943	0.375	1.107	0.984	0.000	1.000	1.000	1.000	0.586	1.092	2.878	0.558	1.136	9.053	0.491	1.148	155.493

TABLE II
RESULTS BY OUR ALGORITHMS FOR DIFFERENT VALUES OF K ON HARDER BENCHMARKS.

	VSB only	MSA ($K=1$)			MSA ($K=2$)			MSA ($K=3$)			MSA-OS ($K=1$)			MSA-OS ($K=2$)			MSA-OS ($K=3$)		
	#shots	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)
1D-1h	922770	55416	992	7.26	25064	1123	22.58	17520	1164	42.65	20080	1191	76.78	17031	1195	1201.74	14940	1196	10666.56
1D-2h	922770	83363	913	6.96	46837	1028	21.29	36250	1076	40.04	34759	1114	73.03	31037	1133	1116.45	30073	1127	9900.95
1D-3h	922770	131776	808	7.00	85705	915	21.95	71875	954	55.39	66473	992	68.14	59868	1023	1158.40	59312	1019	11072.46
1D-4h	922770	164524	747	7.47	112184	858	24.41	96048	894	42.99	89169	924	67.35	80419	959	1263.46	79426	958	12826.80
Normalized	10.083	1.000	1.000	1.000	0.587	1.135	3.143	0.470	1.182	6.324	0.456	1.221	9.955	0.406	1.249	165.141	0.391	1.246	1547.666

TABLE IV
MEMORY USAGE (GB) OF MSA WITH AND WITHOUT USING THE MEMORY SAVING TECHNIQUE.

	MSA with memory saving			MSA without memory saving		
	$K=1$	$K=2$	$K=3$	$K=1$	$K=2$	$K=3$
1D-1	0.136	0.271	0.407	5.236	10.436	15.670
1D-2	0.131	0.262	0.392	5.044	10.087	15.092
1D-3	0.141	0.282	0.422	5.041	10.082	15.087
1D-4	0.156	0.311	0.467	5.205	10.377	15.582
1D-1h	0.136	0.271	0.407	6.282	12.564	18.800
1D-2h	0.131	0.262	0.392	6.051	12.102	18.154
1D-3h	0.141	0.282	0.422	6.048	12.096	18.144
1D-4h	0.156	0.311	0.467	6.245	12.490	18.696

REFERENCES

[1] T. Maruyama, Y. Machida, and S. Sugatani. CP based EBDW throughput enhancement for 22nm high volume manufacturing. In *Proceedings of SPIE 7637*, page 76371S, Feb. 2010.

[2] T. Maruyama, Y. Machida, S. Sugatani, H. Takita, H. Hoshino, T. Hino, M. Ito, A. Yamada, T. Iizuka, S. Komatsu, M. Ikeda, and K. Asada. CP element based design for 14nm node EBDW high volume manufacturing. In *Proceedings of SPIE 8323*, page 832314, April 2012.

[3] B.J. Lin. Future of multiple-e-beam direct-write systems. In *Proceedings of SPIE 8323*, March 2012.

[4] R. Inanami, S. Magoshi, S. Kousai, A. Ando, T. Nakasugi, I. Mori, K. Sugihara, and A. Miura. Maskless lithography: Estimation of the number of shots for each layer in a logic device with character projection-type low-energy electron-beam direct writing system. In *Proceedings of SPIE 5037*, pages 1043–1050, June 2003.

[5] H. Yasuda, A. Yamada, and M. Yamabe. Multi column cell (MCC) e-beam exposure system for mask writing. In *Proceedings of SPIE 7028*, page 70280B, 2008.

[6] H.C. Pfeiffer. Variable spot shaping for electron-beam lithography. *Journal of Vaccum Science and Technology*, 15(3):887–890, May 1978.

[7] M. Sugihara, T. Takata, K. Nakamura, R. Inanami, H. Hayashi, K. Kishimoto, T. Hasebe, Y. Kawano, Y. Matsunaga, K. Murakami, and K. Okumura. Cell library development methodology for throughput enhancement of electron beam direct-write lithography systems. In *Proceedings of International Symposium on System-on-Chip*, pages 137–140, Nov. 2005.

[8] A. Fujimura. Design for e-beam: Design insights for direct-write maskless lithography. In *Proceedings of SPIE 7823*, pages 137–140, Sept. 2010.

[9] K. Yoshida, T. Mitsuhashi, S. Matsushita, L.L. Chau, T.D. T. Nguyen, D. MacMillen, and A. Fujimur. Stencil design and method for improving character density for cell projection charged particle beam lithography. US Patent, Dec. 31 2009.

[10] K. Yuan, B. Yu, and D. Z. Pan. E-beam lithography stencil planning and optimization with overlapped characters. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 31(2):167–179, Feb. 2012.

[11] B. Yu, K. Yuan, J.-R. Gao, and D.Z. Pan. E-BLOW: e-beam lithography overlapping aware stencil planning for MCC system. In *Proc. of DAC*, 2013.

[12] J. Kuang and E.F.Y. Young. A highly-efficient row-structure stencil planning approach for e-beam lithography with overlapped characters. In *Proceedings of International Symposium on Phyiscal Design*, pages 109–116, 2014.

[13] W.K. Mak and C. Chu. E-beam lithography character and stencil co-optimization. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 33(5):741–751, May 2014.

[14] R. Inanami. Electron beam exposure apparatus, electron beam exposure method and method of manufacturing semiconductor device. US Patent, Nov. 11 2008.

[15] C. Chu and W.K. Mak. Flexible packed stencil design with multiple shaping apertures for e-beam lithography. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 137–142, 2014.

[16] G.S. Chua, W.L. Wang, B.I. Choi, Y. Zou, C. Tabery, T. Nguyen, and A. Fujimura. Optimization of mask shot count using MB-MDP and lithography simulation. In *Proceedings of SPIE 8166*, page 816632, 2011.

[17] Aki Fujimura, David Kim, Ingo Bork, and Christophe Pierrat. Writing 32nm-hp contacts with curvilinear assist features. In *Proceedings of SPIE 7823, Photomask Technology*, 2010.

Chris Chu received the B.S. degree in computer science from the University of Hong Kong, Hong Kong, in 1993. He received the M.S. degree and the Ph.D. degree in computer science from the University of Texas at Austin in 1994 and 1999, respectively.

Dr. Chu is a Professor in the Electrical and Computer Engineering Department at Iowa State University. His area of expertises include CAD of VLSI physical design, and design and analysis of algorithms.

Dr. Chu is currently an associate editor for IEEE TCAD. He has served on the technical program committees of several major conferences including DAC, ICCAD, ISPD, ISCAS, DATE, ASP-DAC, and SLIP.

Dr. Chu received the IEEE TCAD best paper award at 1999 for his work in performance-driven interconnect optimization. He received another IEEE TCAD best paper award at 2010 for his work in routing tree construction. He received the ISPD best paper award at 2004 for his work in efficient placement algorithm. He received another ISPD best paper award at 2012 for his work in floorplan block shaping algorithm. He received the ASPDAC best paper award at 2014 for his work in stencil design for electron-beam lithography. He received the Bert Kay Best Dissertation Award for 1998-1999 from the Department of Computer Sciences in the University of Texas at Austin. He is a Fellow of IEEE.

Wai-Kei Mak received the B.S. degree from the University of Hong Kong, Hong Kong, in 1993, and the M.S. and Ph.D. degrees from the University of Texas at Austin, U.S., in 1995 and 1998, respectively, all in computer science.

Dr. Mak is now a Professor with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. He was an Assistant Professor with the Department of Computer Science and Engineering, University of South Florida, U.S., from 1999 to 2003. His current research interests include VLSI physical design automation, and CAD for field-programmable technologies.

Dr. Mak is a recipient of the IEEE/ACM Asia and South Pacific Design Automation Conference 2014 Best Paper Award for his work in e-beam lithography throughput optimization. His lab won the first place at the FPT 2008 Logic Block Clustering Contest, the third place at the IEEE CEDA PATMOS 2011 Timing Analysis Contest, and the second place at the TAU 2013 Variation-Aware Timing Analysis Contest. He has served on the Program and/or the Organizing Committee of Asia South Pacific Design Automation Conference, the International Conference on Field Programmable Logic and Applications, and the International Conference on Field-Programmable Technology (FPT). He was the Technical Program Chair of FPT in 2006 and was the General Chair of the same conference in 2008. Since 2009, he has been a Steering Committee Member of the International Conference on Field-Programmable Technology.