E-Beam Lithography Stencil Planning and Optimization with Overlapped Characters

Kun Yuan, Bei Yu, and David Z. Pan, *Senior Member, IEEE*

***Abstract*—Electronic beam lithography (EBL) is one of the promising emerging technologies in the sub-22 nm regime. In EBL, the desired circuit patterns are directly shot into the wafer, which overcomes the diffraction limit of light in the current optical lithography system. However, the low throughput becomes its key technical hurdle. In the conventional EBL system, each rectangle in the layout will be projected by one electronic shot through a variable shape beam (VSB). This could be extremely slow. As an improved EBL technology, character projection (CP) shoots complex shapes, so-called characters, in one time, by putting them into a predesigned stencil. However, only a limited number of characters can be employed, due to the area constraint. Those patterns, not contained by any character, are still required to be written by VSB. A key problem is how to select an optimal set of characters and pack them on the CP stencil to minimize total processing time. In this paper, we investigate a problem of electronic beam lithography stencil design with overlapped characters. Different from previous works, besides selecting appropriate characters, their placements on the stencil are also optimized in our framework. Specifically, we propose a Hamilton-path-based iterative algorithm to handle 1-D stencil design problem, and an effective simulated annealing framework for the generalized 2-D case with an efficient look-ahead sequence pair evaluation technique. The experimental results show that, compared to conventional stencil design methodology without overlapped characters, we are able to reduce total projection time by 51%.**

***Index Terms*—Electronic beam lithography (EBL), overlapped characters, stencil design.**

I. Introduction

AS AGGRESSIVE scaling continues, the conventional 193 nm optical photolithography technology is facing the great challenge of printing sub-32 nm. For the near future, double/multiple patterning lithography has been developed as a temporary solution for 32 nm, 22 nm, even 16 nm, technology [1]–[6], but the manufacturing cost will be proportionally higher. This makes multiple patterning a relatively temporary solution for further scaling. In the long term, the semiconductor industries and researchers have been actively pushing on alternative emerging nanolithography to print finer feature size below 16 nm, such as electronic beam lithography (EBL), extreme ultraviolet (EUV), and nanoimprint.

EBL [7]–[9] is a maskless technology which shoots desired patterns directly into a silicon wafer, with a charged particle beam. The primary advantage is that it is one of the ways to beat the diffraction limit of light of current well-adopted optical lithography [10], which leads to 4× better resolution and lower cost compared to conventional optical lithography. However, EBL has one key limitation, low throughput.

The conventional type of the EBL system is variable shaped beam (VSB). In VSB, the layout is usually decomposed into a set of rectangles, and each one would be shot into resist by dose of electron sequentially. As Fig. 1(a) shows, the pattern of "EHE" is divided into 11 rectangles and needs total 11 shots. The whole processing time of this technique increases with number of beam shots. This makes its throughput very low for the modern complicated design, which is commonly composed of a significant number of small rectangles.

The character projection (CP) technology [7]–[9] has been invented for improving the throughput of VSB methods. The key idea is to print some complex shapes in one electronic beam shot, rather than writing multiple small rectangles. This reduces manufacturing time significantly. In detail, as the projection system of CP in Fig. 1(b) illustrates, a library of layout configurations, called *Character*s, or *Template*s, are prepared on a *stencil* first. During manufacturing, if any character exists in the targeted design, it will be chosen in the system and projected into the wafer. To print the example of Fig. 1(a), suppose two characters "E" and "H" are predesigned for the stencil. By adjusting and aligning the shaping aperture and stencial, we can print the patterns of "E," "H," "E" in a sequential manner, as Fig. 1(c)–(e) shows. In total, it only takes three shots.

Due to less beam shots for the same layout, the CP system is much faster than VSB. However, the number of characters is limited due to the area constraint of the stencil. As in the example of Fig. 1(f), there are only maximum $\lfloor W/w \rfloor \lfloor H/h \rfloor$ characters. For modern design, it is not practical to fully make use of CP, due to numerous distinct circuit patterns. Those patterns which do not match any character are still required to be written by VSB. Standard cell and some regular routing pattern, which have high area utilization of stencil because of their frequent appearances in the design, are good candidates for implementing characters. On the other side, the custom layout from memory IP, and most irregular wires/vias would most likely be processed by VSB.

Manuscript received June 13, 2011; revised August 12, 2011 and October 10, 2011; accepted November 7, 2011. Date of current version January 20, 2012. The preliminary conference version has been published at the International Symposium on Physical Design in 2011. This paper was recommended by Associate Editor J. Hu.

K. Yuan is with Cadence Design Systems, San Jose, CA 95134 USA (e-mail: kyuan@cerc.utexas.edu).

B. Yu and D. Z. Pan are with the Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78731 USA (e-mail: bei@cerc.utexas.edu; dpan@cerc.utexas.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2011.2179041

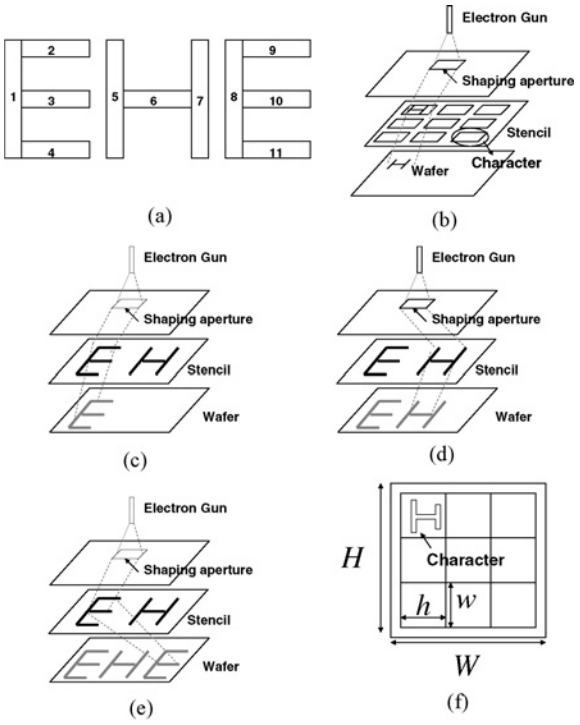


Fig. 1. Electron beam lithography. (a) VSB. (b) CP. (c) First shot of CP. (d) Second shot of CP. (e) Third shot of CP. (f) Typical Stencil for CP.

Several methodologies have been proposed to design and select group of circuit patterns as characters for minimizing total projection time of both CP and VSB. In [11], frequently-used standard cells are greedily chosen as characters, processed by CP technology. Sugihara *et al.* [12]–[15] employed integer linear programming to optimize the throughput, given a set of character candidates. Recently, EDA Vendor D2S, Inc. [7]–[9] proposed improving stencil design from a new point of view, but with no detailed algorithm presented. They showed that, in practice, when individual character/template is designed, blanking area is usually reserved around its boundaries. By sharing blanks between adjacent templates, more characters can be placed on the stencil than the regular design of Fig. 1(f), better improving the throughput.

The work of [7]–[9] implies that, due to possible overlapping, besides selecting appropriate characters as [11]–[15], their relative locations on the stencil should also be taken into account for minimizing the total projection time of EBL. In this paper, we will investigate this new problem of electronic beam lithography stencil design with overlapped characters. 1-D/2-D problem is researched separately, depending on whether the available overlapping space of characters is nonuniform in either horizontal or both directions. Moreover, we only consider the application of one stencil for this paper. The main contributions of our work are stated as follows.

1) We co-optimize the selection process of characters and their physical placements on stencil for effective EBL throughput improvement.
2) We propose a three-phase iterative refinement process to conduct 1-D stencil design optimization. A Hamilton-path based approach has been developed to solve single-row reordering efficiently and effectively.
3) We develop a sequence pair (SP) based simulated annealing framework to optimize general 2-D stencil design. We also propose an efficient look-ahead sequence pair evaluation algorithm to reduce runtime.

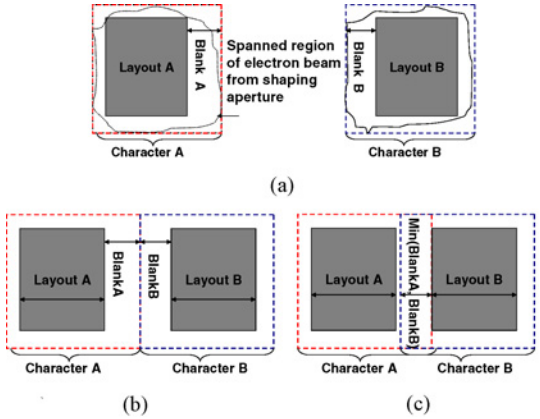


Fig. 2. Overlapped characters for improving the stencil densities. (a) Layout of characters. (b) Conventional stencil. (c) Overlapped characters.

II. Preliminaries and Problem Formulation

A. Overlapped Character

EBL is a maskless technique, which shoots desired patterns directly into a silicon wafer, and can potentially combat device parameter variations [16] and design congestion [17], [18]. Various investigations [12]–[15] have been conducted on the optimization of character selection for EPL technology, where no intersection is allowed between templates on the stencil, as shown by Fig. 1(f). Recently, the work of [7]–[9] has shown that the design of stencil can be further improved by overlapping adjacent characters, which allows more templates to be put and increases the throughput.

As pointed out by [7]–[9], when an individual character is designed, blanking space is usually reserved around its enclosed rectangular circuit pattern, shown by Fig. 2(a). The reason is that, when the electron beam is scattered from the shaping aperture of Fig. 1(b), it could span larger area on the stencil than the layout to be printed. In order to avoid projecting any unwanted image, the white space should be preserved. These blanking areas offer great opportunity for character sharing.

Suppose the required white space around layouts A and B are $BlankA$ and $BlankB$, respectively, in Fig. 2(a). If the characters are conventionally aligned by edge as Fig. 2(b), it results in a waste of area. The space between layout A and B is actually $BlankA + BlankB$, which is more than required for both patterns. By contrast, we would greatly reduce the total area of character A and B by sharing an amount of $min(BlankA, BlankB)$ space. In this case, $max(BlankA, BlankB)$ white width is still reserved between layout A and B, which is sufficient for ensuring correct printing image.

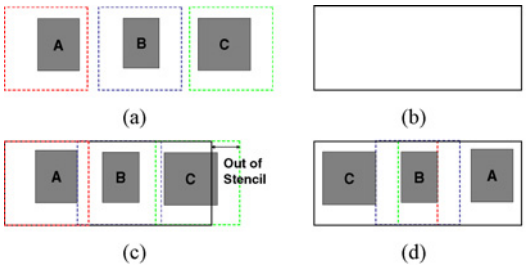


Fig. 3. Main difficulty of stencil design with overlapped characters.

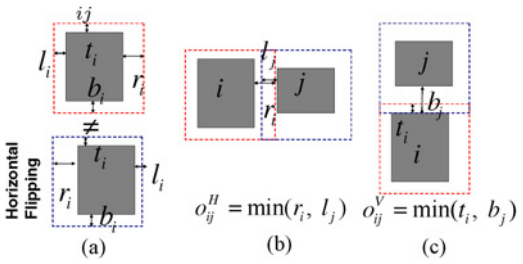


Fig. 4. Dimensional variable of character candidates.

B. Stencil Design Challenge

The main challenge of stencil design with overlapped characters comes from the fact that, for each character, the amount of required blanking space is not uniform, strongly depending on its enclosed layout patterns [9]. Because different shapes have different scatting and proximity effects under the projection of E-beam, to preserve the shape and fidelity of the written image, their minimum required shielding spacings vary and are not identical. The underlying reason is similar to that in the conventional optical lithography system; the spacing rules between metals are usually different, determined by geometrical dimension of associated design objects.

In consequence, for different placements of characters, the area reduction from template overlapping may vary a lot. Therefore, unlike the traditional design of Fig. 1(f), the number of maximum allowable characters in the stencil is not fixed. To achieve a high quality solution, the detailed physical placement information of all the characters must be taken into account. This makes the problem of stencil design with overlapped characters not only different from but also more difficult than the conventional nonoverlapping one addressed in [12]–[15].

As the example of Fig. 3(a) illustrates, suppose there are three character candidates A–C, and we would like to pack them into a simple stencil of Fig. 3(b) for minimum projection time. As easily seen, their blanking spaces are quite different. In conventional design where overlapping is not considered, at most two of them can fit. On the other side, when the blanking space is shared by adjacent characters, the result is correlated with the detailed physical implementation of stencil, and could be different from traditional design. If these three candidates are tried out by the order of A–B–C like Fig. 3(c), only A and B can be put in. Pattern C is out of bound and has to be processed by the VSB technique. This does not lead to higher throughput than the conventional nonoverlapped methodology. In contrast, if rearranged as C–B–A as Fig. 3(d), all of these three patterns can be used as CP characters. Obviously, it is a better stencil optimization.

C. Problem Formulation

Similar to previous work [12]–[15], we assume a set of character candidates have already been given. To model overlapping information, as Fig. 4(a) illustrates, assume the blanking spaces of each candidate c_i, from left, right, top and bottom boundaries, are l_i, r_i, t_i, and b_i, respectively. The orientation of these candidates is not allowed to be flipped, since it actually becomes a different template, as explained in [13]. When two candidates c_i and c_j are put adjacent to each other horizontally, their maximum allowed overlap is set as o_{ij}^H, which is $min(r_i, l_j)$ as shown by Fig. 4(b). Similarly, Fig. 4(c) defines the maximum vertical overlapping margin o_{ij}^V. o_{ij}^H and o_{ij}^V vary for different i and j.

Moreover, since the manufacturing time of EBL is dominantly determined by electronic beam shooting, in our work we make use of total number of shots as the measurement of projection time. Suppose each candidate c_i is referred r_i^c times in the chip. For each of its appearance, the candidate c_i will be projected by either CP or VSB method, with a number of shots n_i^{CP} and n_i^{VSB}. The total processing time (number of shots) of the entire circuit is computed by the following equation:

$$\sum_{c_i \in C^{\mathrm{CP}}} r_i^c n_i^{\mathrm{CP}} + \sum_{c_i \in (C^C \setminus C^{\mathrm{CP}})} r_i^c n_i^{\mathrm{VSB}}. \tag{1}$$

C^C is the set of all the character candidates. C^{CP} is the union of selected candidates processed by the CP method, which is a subset of C^C.

In our work, as discussed in the introduction, we only design and optimize one single stencil for a given chip. Our optimization problem can be stated as follows.

Problem formulation: given a design and its set of character candidate C^C, select a subset C^{CP} out of C^C as characters, and place them on the stencil S. The objective is to minimize the total projection time (number of shots) of this design expressed by (1), while the placement of C^{CP} is bounded by the outline of S. The width and height of stencil is W and H, respectively, and all the candidate has unique width w and height h. The maximum overlapping margin between adjacent characters is given by o_{ij}^H and o_{ij}^V.

D. Stencil Design Guidance

Multiple factors are required to be taken into account to select appropriate candidates onto stencil, which includes the times of references of each candidate in the design, the shot number of each candidate by VSB and CP respectively, the amount of blank margin and the size of each candidate, and the neighborhood relationships of characters.

Note that the overall projection time, Objective (1), can be rewritten as

$$\sum_{c_i \in C^C} r_i^c n_i^{\mathrm{VSB}} - \sum_{c_i \in C^{\mathrm{CP}}} r_i^c (n_i^{\mathrm{VSB}} - n_i^{\mathrm{CP}}) \tag{2}$$

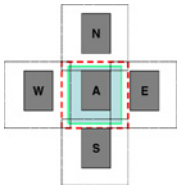


Fig. 5. Illustration of effective area. The figure is only drawn for the case that the blanking spaces of candidates are uniform.

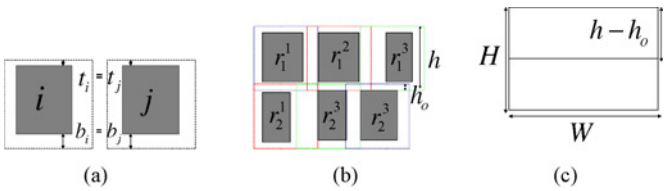


Fig. 6. 1-D stencil design.

subject to

$$\sum_{c_i \in C^{\mathrm{CP}}} area_i < Area(stencil) \tag{3}$$

where the first part is independent of stencil design. To reduce the processing time, $\sum_{c_i \in C^{\mathrm{CP}}} r_i^c(n_i^{\mathrm{VSB}} - n_i^{\mathrm{CP}})$ should be maximized under the area constraint. In other words, we need to maximize the shot number reduction per unit area of stencil, by putting characters onto the stencil. For the five factors we mentioned above, the shot number of each candidate by VSB and CP contributes to the shot number reduction. The amount of blank margin, the size of each candidate and their neighborhood relationship contribute to the optimization of area sharing from character placement. Guided by this observation, the criteria of our character selection is listed in the following paragraph.

For a candidate c_i, suppose it has a area of $area_i^o$ shared with other surrounding candidates, we only count half of $area_i^o$ as its *real* usage of stencil and compute the effective area $area_i^{eff}$ of c_i as $area_i - area_i^o/2$. As Fig. 5 shows, the red dash encloses $area_i$, and the light green region is $area_i^{eff}$. Therefore, to achieve maximum throughput improvement, we simply select the candidate with highest shot number reduction to effective area ratio, $(r_i^c(n_i^{\mathrm{VSB}} - n_i^{\mathrm{CP}}))/area_i^{eff}$, until no more character candidates can be fit in.

When the blanks are uniform for all the candidates, the above methodology is easily adoptable. In such a case, the sharing space of adjacent characters is identical for any two candidates, so $area_i^o$ is independent of character placement and can be precomputed. While overlapping margin varies, this problem becomes difficult, because $area_i^o$ is not a deterministic but effected by the detailed placement of characters. In this paper, we will focus this challenging problem with different combinations of overlapping blanking space.

We will first investigate into the special case of 1-D stencil design in Section III, where the amount of blanking spaces differs only in the horizontal direction. Vertical margins are identical for all the candidates. In our proposed optimization, we first put characters in the descending order of shot number reduction, i.e., $(r_i^c(n_i^{\mathrm{VSB}} - n_i^{\mathrm{CP}}))$, then perform optimization to maximize blanking sharing and area reduction. $\sum_{C_i \in C^{\mathrm{CP}}} area_i^o$.

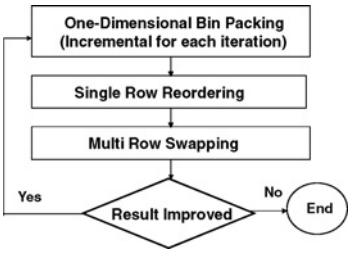


Fig. 7. Overview of 1-D stencil design with overlapped characters.

Next, in Section IV, the algorithm for a generalized 2-D problem will be developed. In this case, both vertical and horizontal blanking space are nonuniform for candidates. Our algorithm for the 2-D problem is a simulated annealing approach. In each iteration, we assign equal weight for an annealer to make two types of moves: the move that pushes in characters having higher shot number reduction $(r_i^c(n_i^{\mathrm{VSB}} - n_i^{\mathrm{CP}}))$, and the move which better increases the sharable space $area_i^o$.

III. 1-D Stencil Design

Normally, each template implements one standard cell. That is to say, the enclosed circuit patterns of all the characters have the same height, and their layouts near top and bottom boundary edges are mostly regular power rails. As a result, illustrated by Fig. 6(a), the required blanking spaces on the top t and bottom b are nearly identical for these candidates.

Therefore, in such a case, characters are usually placed on the stencil in a row-based manner, shown by Fig. 6(b). All rows have a unique height h. The overlapped blanking margin h_o between adjacent rows is also the same, which is $\min(t_i, b_i)$. In consequence, as Fig. 6(c) shows, the overlapping-aware stencil design becomes a 1-D problem. The number of character rows can be pre determined as $\lfloor (H - h_o)/(h - h_o) \rfloor$. The candidates would be packed into these rows with maximum width W.

Motivated by this fact, in this section, we will discuss a 1-D stencil design problem, with the assumption that all the candidates have the same vertical blanking space t_i and b_i. The overview of our algorithm for this special 1-D problem is given in Fig. 7.

In the very beginning, all the rows are empty. Next, character candidates will be pushed onto the stencil in 1-D bin packing, until no more can further fit in. Then steps 2 and 3 serve for the same suppose, tuning the position and selection of characters to create more space while not degrading the throughput. This allows adding more candidates into stencil in the next iteration. After step 3, 1-D bin packing will be re-performed incrementally based on the optimization result of the previous iteration. In other words, the existing characters on the stencil of step 3 will be fixed when we revoke the first phase of bin packing. However, in each iteration, steps 2 and 3 will be executed for all the existing characters, with no exceptions or fixed objects.

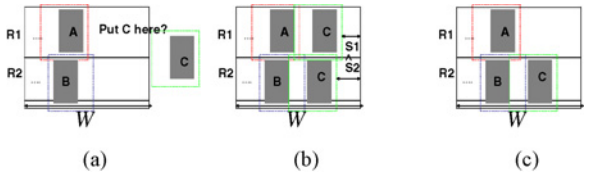


Fig. 8. This figure illustrates the procedure of best-fit bin packing with overlapping awareness.

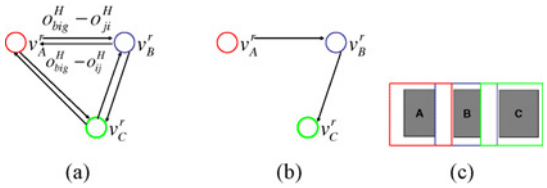


Fig. 9. This figure shows how to optimize the occupied width of each row as a min-cost Hamiltonian path problem.

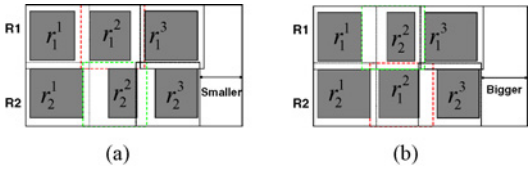


Fig. 10. This figure explains the motivation of multirow swapping.

A. Greedy 1-D Bin Packing

To construct a reasonable good starting point, we adopt a descending best-fit bin packing algorithm to push the character candidates into stencil, until there is no enough capacity.

Recall that the overall projection time (number of shots) of Objective (1) can also be represented as

$$\sum_{c_i \in C^C} r_i^c n_i^{\mathrm{VSB}} - \sum_{c_i \in C^{\mathrm{CP}}} r_i^c (n_i^{\mathrm{VSB}} - n_i^{\mathrm{CP}})$$

where the first part is independent of stencil design. To reduce the processing time, $\sum_{c_i \in C^{\mathrm{CP}}} r_i^c (n_i^{\mathrm{VSB}} - n_i^{\mathrm{CP}})$ should be made as large as possible, during greedy bin-packing.

Therefore, as preprocessing, we first assign each candidate c_i a profit value p_i, $r_i^c(n_i^{\mathrm{VSB}} - n_i^{\mathrm{CP}})$. The bigger p_i is, the larger amount of projection efforts can be saved by printing c_i using CP than VSB method. For getting a good greedy optimization result, the c_i with larger profit should be given higher priority to be placed on the stencil. Guided by this heuristic, in the second step, the character candidates, which have not been on the stencil yet, will be sorted decreasingly based on their profits and packed in a sequential manner.

Next, these sorted candidates will be pushed into stencil by a best-fit packing strategy. When c_i is to be packed, the row, which has the most amount of capacities left *after* accommodating c_i, will be picked. This is to consider the possible shared space between adjacent objects, when we are computing the remaining room in each row. As Fig. 8(a) illustrates, suppose only two rows are available and candidate C is to be packed next. It appears that row R1 has more capacity left. However, as Fig. 8(b) illustrates, when we try out C in both rows, it is R2 which has larger remaining room. As a result, candidate C is packed into R2, shown by Fig. 8(c).

B. Single Row Reordering

After greedy bin packing, there is no room left to accommodate more candidates. However, as motivated by Fig. 3, we can adjust the relative locations of already-placed characters in each row to shrink its occupied width and increase remaining capacity. This allows pushing in more candidates, which further reduces the overall projection time. Therefore, in this phase, our goal is to minimize the total width of its characters in each row for maximizing remaining capacity.

Suppose row r contains a set of c_0^r .... c_n^r characters from left to right, its total occupied width can be computed as $\sum_{i=0}^{n} w - \sum_{i=0}^{n-1} o_{i,i+1}^H$. It is not difficult to see that $\sum_{i=0}^{n} w$ is a constant as long as the number of characters is not changed. Therefore, to minimize the total occupied width, the overall overlapped blanking margin $\sum_{i=0}^{n-1} o_{i,i+1}^H$ should be maximized.

To compute optimal character permutation for maximum amount of shared blanking width, we formulate a minimum cost Hamiltonian path problem. First, a graph G is constructed as follows. Each c_i^r is represented by a vertex v_i^r. For each pair of v_i^r and v_j^r, we add two directed edges e_{ij} and e_{ji}. The associated costs are $o_{big}^H - o_{ij}^H$ and $o_{big}^H - o_{ji}^H$, respectively. o_{ij}^H / o_{ij}^H is the shared space when c_i is put left/right adjacent to c_j, and o_{big}^H is a constant value, bigger than any of o_{ij}^H. To maximize $\sum_{i=0}^{n-1} o_{i,i+1}^H$, it suffices to find a path visiting each node of G exactly once such that the total edge weights $(\sum_{e \in Path}(o_{big}^H - o_{ij}^H))$ along this path are minimized. As Fig. 9(a) illustrates, a graph for three character placement (A, B, C) is given. Suppose the minimum cost Hamiltonian path is found as Fig. 9(b), Fig. 9(c) shows its corresponding character placement.

Since the problem of minimum cost Hamiltonian path is NP-hard, it may be expensive to solve the whole row in one time. Our heuristic is to partition the row into multiple overlapped smaller segments, and solve each segment by the Hamiltonian path based method.

C. Multiple Row Swapping

After single row reordering, the character permutation within each row has been extensively optimized. However, it is still possible to increase their remaining capacities, by swapping characters from different rows. As Fig. 10 illustrates, by swapping r_1^2 and r_2^2, both characters find "better" neighbors with more overlapped blanking space. For rows R1 and R2, their remaining rooms are both increased.

The algorithm is briefly explained as follows. We test every pair of characters from different rows. Only when the remaining capacities of both rows are increased after swapping, it is considered as a *reasonable* swap. This ensures that the modified placement is definitely better than the original one. The reason is that, after swapping, if one row gains more room but another has less, it is possible that the following optimization is hurt by the row with shrunk capacity. After the *reasonable* swap pairs are found, they are sorted increasingly by capacity gains, and performed one pair by one pair. One thing to emphasis here is that the algorithm could be quite slow if we really

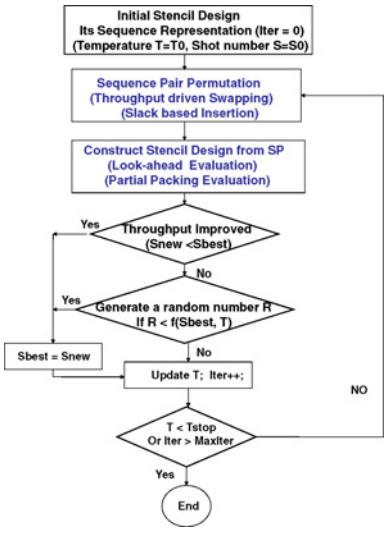


Fig. 11. Overview of 2-D stencil design with overlapped characters.

iterate through all the possible swapping, when the the total number of characters is large. To encounter this issue, we only randomly select a predefined number of swapping candidates.

IV. 2-D Stencil Design

In this section, we investigate the general case of EBL stencil design with overlapped characters. The blanking spaces of templates are nonuniform along both horizontal and vertical directions. Due to NP-completeness of this problem, we adopt a simulated-annealing based heuristic approach

The basic flow is shown in Fig. 11. Parquet [21] is adopted as our simulated annealing framework. Given an initial solution and starting temperature T0, each iteration we continuously make small permutation on sequence pair, and evaluate the resulting stencil design, as the blue steps indicated. The new SP/solution will definitely be adopted if reducing current best throughput (shot number Sbest) of Objective (1). While it is actually a worse character placement, this nonimproving result is accepted with a probability, computed from a function of current temperature T and Sbest. The function f(Sbest, T) decreases as Sbest and T decrease. At the end of each iteration, the temperature will be updated with decreasing trend. The algorithm terminates when the upper bound of iteration number or lower bound of temperature is reached. The detailed discussion for sequence pair evaluation and permutation can be found in Sections IV-B to IV-D, respectively.

A. Sequence Pair Representation

To represent a character placement solution, we make use of SP proposed in [19].

Given a set of character candidates C^C, its SP consists in two permutations $\overline{X}\&\overline{Y}$ of these templates $(c_0, c_1 ... c_n)$, which specifies their geometrical relationships as follows:

$$(\overline{X} :< .., c_i .. c_j .. >, \overline{Y} :< ... c_i ... c_j ... >) : c_i \text{ is left to } c_j \quad (4)$$

$$(\overline{X} :< .., c_j .. c_i .. >, \overline{Y} :< ... c_i ... c_j ... >) : c_i \text{ is below } c_j. \quad (5)$$

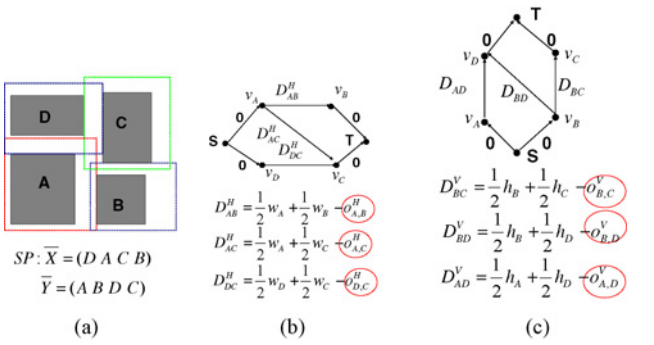


Fig. 12. This figure explains packing evaluation of [19] based on sequence pair. (b) H graph. (c) V graph.

Based on these constraints, we can map any SP into a solution of character placement as the following procedure.

Procedure 1.

Step 1: compute a packing solution of C^C, following the extensive methods of [19] and [20]. The details and speedups will be described in Sections IV-B and IV-C.

Step 2: assuming the left-bottom coordinates of packed candidate and stencil are the same, the candidates, which are located completely within the outline of stencil, are considered as selected characters.

Step 1 is the critical one in the above transformation. Due to specific properties of our problem, its implementation actually differs from the conventional approaches of [19] and [20], explained as follows.

B. Fast Look-Ahead Sequence Pair Evaluation

The key step of packing solution evaluation from SP is to determine the physical coordinates of each block. This problem has been well investigated, when overlapping is not considered between adjacent blocks. The original algorithm is proposed in [19], and improved by [20] with a new solution pruning technique. The work of [19] is easily extensible for our overlapping-enabled character placement problem, described in Section IV-B1, but it is well known that this type of method is very computationally-expensive. On the other side, in Section IV-B2 we show that the key speedup idea in [20] is not applicable directly or extensible trivially, although it is much faster. We then present an efficient look-ahead algorithm based on framework of [20] to solve this specific sequence pair computation problem.

1) *Extension for Longest Path Based Algorithm:* The method of [19] is based on a longest path algorithm, and starts from constraint graph construction. Given a SP, a H/V graph is built first to capture the horizontal/vertical relationship between different blocks. Assume there are totally C^C candidates. The H/V graph has $|C^C|$+2 vertexes, one v_i for each candidate c_i plus a source s and sink t. If c_j is (left adjacent to)/(below) c_k, a directed edge e_{jk} is added from v_j to v_k. The weight of e_{jk} is the minimum possible horizontal/vertical distance between the centers of c_j and c_k. Beside these, there is a zero-weight edge from source to every v_i, and a zero-weight edge from every v_i to sink. For the example of Fig. 12(a), Fig. 12(b) and (c) shows the resulting H and V constraint graphs, respectively.

TABLE I
NOTATION FOR THE ALGORITHM IN [20]

b	Any character candidate, b, is one of 1, ..., n
$X[i]$	Character candidate in ith position of the $\overline{X}$ sequence
$Y[i]$	Character candidate in the ith position of the $\overline{Y}$ sequence
$match[b].x$	The position of b in the $\overline{X}$ sequence
$match[b].y$	The position of b in the $\overline{Y}$ sequence
$pos[b]$	The left x coordinate of b in the stencil
$width[b]$	The width of character candidate b
$o[m, n]$	The overlapping when candidate m is left to n
$L[b]$	The auxiliary array to record the current x coordinate of candidate b

Algorithm 1 Calculate x Coordinates of Candidates Given a Sequence Pair

```
1: for i = 1 to n do
2:    match[X[i]].x = i; match[Y[i]].y = i; L[i] = 0;
3: end for
4: for i = 1 to n do
5:    b = X[i];p = match[b].y; pos[b] = L[p];t = pos[b] +
      width[b];
6:    for j = p to n do
7:       if t > L[j] then
8:          L[j]= t;
9:       else
10:         break;
11:      end if
12:   end for
13: end for
```

After that, the x/y coordinates of these candidates can be obtained by finding its weighted longest path algorithm from source. As easy to see, this methodology is also applicable for our problem, where overlapped space is allowed between adjacent vertexes. The only difference is that, when the weights of edge are assigned, the amount of shared blanking space must be considered, as highlighted by the red cycles in Fig. 12(b) and (c).

This type of longest path based algorithm is quite slow, as shown by [20]. In our work, we implement this extension as the baseline to show the effectiveness of our look-head pruning technique, which is proposed in Section IV-B2.

2) *Extension for Longest Common Subsequence Based Algorithm:* The work of [20] does not explicitly build the constraint graphs, but depends on the longest common subsequence computation. They evaluated the placement of character candidates much faster than [19]. In this subsection, we first show that the pruning algorithm in [20] is not applicable directly due to the breakdown of its assumption. Then we proposed a look-ahead technique to revive the core idea of [20] for fast sequence pair computation.

a) *Review of original pruning algorithm:* To better present our proposed pruning technique, first, we review the key steps of [20] in Algorithm 1, and its pitfall for overlapping-induced character floorplanning. The notations are listed in Table I. Their algorithm is based on longest common subsequence. Here, we only consider the situation where the x coordinate is computed.

Algorithm 2 Initial look-ahead algorithm for sequence pair evaluation

```
1: for i = 1 to n do
2:    match[X[i]].x = i; match[Y[i]].y = i; L[i] = 0; MinT[i] = 0
3: end for
4: for i = 1 to n do
5:    b = X[i];p = match[b].y; pos[b] = L[p];
6:    t = pos[b] + width[b]; MinT[n] = L[n]+o[b][Y[n];
7:    for j = n-1 to p, j = j - 1 do
8:       MinT[j] = min(MinT[j+1], L[j]+o[b][Y[j]]);
9:    end for
10:   for j = p to n do
11:      if t < MinT[j] then
12:         break;
13:      end if
14:      if t > L[j]+o[b][Y[j]] then
15:         L[j]= t-o[b][Y[j]];
16:      end if
17:   end for
18: end for
```

Lines 6–11 implement the essential pruning idea of [20]. In Line 8, L[j] will be updated if and only if the new required x coordinate t for candidate Y[j] is larger than current L[j]. Without considering the item of $o[b][Y[j]]$ in the nonoverlapping case, original algorithm can do effective pruning, lying on the foundation of the following ascending property of L[j]: if i is larger than j, then L[i] is guaranteed no less than L[j]. Once t is smaller than certain L[j] in Lines 7, we can directly break out of the loop Lines 6–11. Any element in array L after index j is definitely bigger than t, and there is no need to do further evaluation using Lines 6–11.

Unfortunately, while overlapping is considered, the original algorithm cannot be simply modified by updating Lines 7 and 8 using (6) and (7), respectively

$$t - o[b][Y[j]] > L[j] \tag{6}$$

$$L[j] = t - o[b][Y[j]]. \tag{7}$$

The reason is that the array of L does not have such an ascending property any more. This is due to the variations of o[b][Y[j]]. When we update array L by (7), L[m] may be larger than L[n] even m is smaller then n, because o[b][Y[m]] may be smaller than o[b][Y[n]].

b) *Initial look-ahead pruning algorithm:* Motivated by the above analysis, we first present our initial look-ahead sequence pair evaluation algorithm. The main direction is to find a good threshold, e.g., $MinT$, which maintains an ascending property and hence can replace the usage of L[i] for correct pruning. This is actually not difficult. As we may find out, $MinT[j] = min(L[j] + o[b][Y[q]])$, $j < q < n$, satisfies the requirement. The original algorithm can be then revised as Algorithm 2. Note that Line 11 is equivalent to checking the inequality specified by (6).

Lines 7–12 are the key steps here. If the condition of Line 11 is satisfied when j = p1, the Line 15 will not have chance to be executed for following iterations ($j >= p1$). We save computational operation here.

It works correctly, and seems helping reducing runtime with the pruning of Lines 11–12. However, if we take a closer look,

Algorithm 3 Efficient look-ahead algorithm to speed up sequence pair evaluation

```
1:  for i = 1 to n do
2:     match[X[i]].x = i; match[Y[i]].y = i; L[i] = 0; MinT[i]
       = 0
3:  end for
4:  for i = 1 to n do
5:     b = X[i];p = match[b].y;pos[b] = L[p];t = pos[b] +
       width[b];
6:     index = n;
7:     for j = p to n do
8:        if t < MinT[j] then
9:           index = j;
10:          break;
11:       end if
12:       if t > L[j]+o[b][Y[j]] then
13:          L[j]= t-o[b][Y[j]];
14:       end if
15:    end for
16:    for k = index-1 to p, k=k-1 do
17:       MinT[k] = min(MinT[k+1], L[k]);
18:    end for
19: end for
```

the step of calculating $MinT[j] = min(L[q] - o[b][Y[q]])$, $j < q < n$ is expensive. As in Lines 7–9 of Algorithm 2, because o[b][Y[q]] varies with the values of b, MinT[j] have to be fully updated in every iteration of loop Lines 4–17, which requires going through every element from p to n in Lines 7 and 8. This native look-ahead algorithm does not really benefit us in terms of runtime. Our next step is to think out a faster way to compute MinT[j].

c) *Efficient look-ahead pruning algorithm:* To resolve the issue of expensive updating in Algorithm 2, we come up with a little relaxed metric $MinT[j] = min(L[j])$, $j < q < n$ with a much faster computation strategy.

The implementation is described as Algorithm 3. By taking advantage of this new $MinT[k]$, we only need to perform partial updating shown in Lines 16–18. The correctness of this updating is easy to see. Any $MinT[k]$ can only be changed if some L[q], $k < q < n$, has been updated by Line 13, and the value of index-1 is the largest entry in L which has been modified during each iteration. Therefore, any MinT[k], $k > index$, would keep the same and there is no need to update them in Lines 16–18.

Theoretically, in the worst case, Algorithm 3 is as slow as Algorithm 2 because complete updating may still be needed every iteration in Lines 16–18, when the index is equal to n. However, in practice, Algorithm 3 would perform better. The significant speedup comes from the fact that in most of time, the index is much smaller than n. Although the relaxation of MinT[q], from $min(L[j] + o[b][Y[q]])$, $j < q < n$ to $min(L[j])$, $j < q < n$, makes the pruning step Lines 8–10 less frequent, the benefits from efficient $MinT[q]$ updating compensate this degradation.

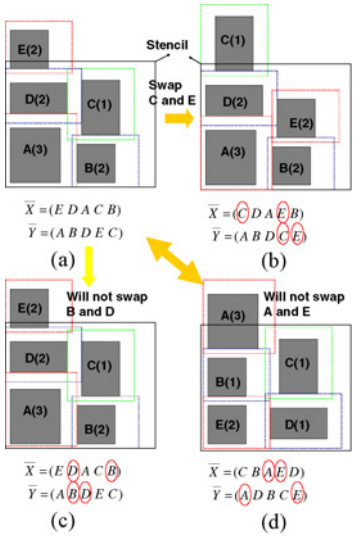


Fig. 13. This figure illustrates throughput-driven swapping.

C. Partial Packing Evaluation

After evaluating packing solution, in step 2 of Procedure 1, the candidates outside the outline of stencil will not be taken as characters. This implies that the detailed locations of these candidates are not important, and do not have to be computed in step 1. Great speedup can be achieved by making use of this property. In detail, in the implementation of SP-based minimum area packing, we stop placement evaluation as soon as the contour of already-packed character candidates is completely outside the outline of stencil by at least a margin of $o_{\max}$, given that $o_{\max}$ is the maximum value of o_{ij}^H and o_{ij}^V.

D. Sequence Pair Permutation

In this subsection, we present two effective SP perturbation methods for better local search toward shorter projection time: throughput-driven swapping and slack-based insertion.

1) *Throughput-Driven Swapping:* The first type of perturbation we perform is throughput-driven swapping. The basic idea is to try reducing overall projection time by swapping the positions of two candidates in the $\overline{X}\&\overline{Y}$ SP. This is equivalent to exchanging their relative locations in the packing solution.

Fig. 13 illustrates a motivational example, which has five blocks A–E to be packed. The required number of shots, to project any of these candidates once, are assumed as 1 and 10 for CP (n_i^{CP})and VSB (n_i^{VSB}) methods, respectively. The digit in the parentheses denotes how many times r_i of each component will be used and printed in the design.

Fig. 13(a) gives a SP representation and its corresponding stencil design, based on the Procedure 1 in Section IV-A. Following the definition of Objective 1, the total processing time (number of shots) is $3 + 2 + 1 + 2 + 10 \times 2 = 28$, since A–D are selected as characters while E is not. If swapping the locations of C and E in SP as Fig. 13(b), we would end up with a better stencil design with less amount processing time. It only takes a number of 19 shots, which is computed as $3 + 2 + 10 + 2 + 2 = 19$ in this case.

In the detailed implementation, we enforce two heuristic swapping constraints to enable efficient and effective shot number reduction.

First, given a SP, out of the pair of elements to be changed, we require that one candidate c_s should have been selected as characters by its corresponding stencil packing, while the other one, c_o, is not. For the example of Fig. 13(a), we only allow the exchange of the positions between E and any of A-D. The swapping among any two of A-D is not enabled. The reason is that if the two candidates to be swapped are both in or out of stencil already, most likely the new SP generates a stencil solution with the same set of selected characters and just different geometrical ordering. As an example, if we swap candidates B and D which are both already in the stencil, like from Fig. 13(a) to (c), the resulting packing result also selects A–D as characters, still requiring 28 shots in total.

Second, after randomly picking in-stencil candidate c_s and out-of-stencil one c_o for swapping, we will compute the difference of their profits $p_o - p_s$, to decide whether this swapping would be tried on. The profit p_o/p_s is defined as same as $r_i(n_i^{\text{VSB}} - n_i^{\text{CP}})$ in Section III-A, which reflects the reduction of the shoot number by printing this candidate with CP rather than VSB. If we swap the locations of c_s and c_o, it is highly likely that c_s will be pushed out of stencil but the c_o would be selected as a character in turn. Assuming all the other candidates stay either in or outside the stencil, as the state before the swapping, the total shot reduction by this exchange can be approximated as $p_o - p_s$. Therefore, if the difference $p_o - p_s$ is smaller than zero, it is in high possibility that the swapping under consideration will not lead to a better packing result. For the example of Fig. 13(a), suppose c_s and c_o are A and E, respectively, and it turns out $p_o - p_s$ is -9. In this case, the corresponding stencil design indeed becomes worse, taking 35 shots as Fig. 13(d) shows.

2) *Slack-Based Insertion:* Given a SP and its corresponding character solution, our purpose of slack-based insertion is to add-in a new candidate, which is currently not serving as a character, into the stencil. To ensure robust throughput improvement, we would like to find a good strategy to insert such an extra candidate so that all the previously already-placed characters are still kept on the stencil in most trials. This equals to increase the number of usable templates. In this subsection, we make use of the concept of slack, applied in [21] and [22], to search such a good insertion location.

Given a character c_s on the stencil, its x/y slack is defined as the allowed movement range of x/y coordinates of c_s, under the constraint that none of all the other already-placed characters would be pushed outside the stencil after such a move. Fig. 14(a) and (b) illustrates a simple example, with four characters A–D. Their leftmost and rightmost packing solutions are shown by Fig. 14(a) and (b), respectively. Based on these two extreme cases, the x slack of C, for example, can be computed as $X_c^{\text{right}} - X_c^{\text{left}}$.

Once slacks are known, we randomly pick a *base* character c_b, which has large slacks in both x and y directions, and insert a new candidate c_{new} before it. The reason is that the location of such *base* can be moved in relatively big amount to make space for additional characters. In terms of SP operation,

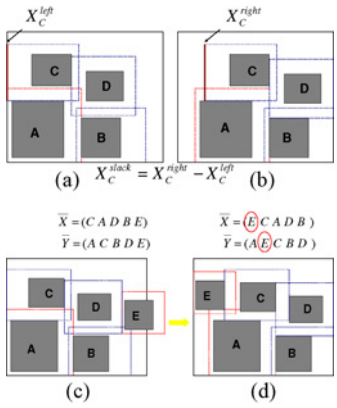


Fig. 14. Simple example of slack-based insertion.

this can be done by simply changing the position of c_{new} right before c_b in $\overline{X}$ and $\overline{Y}$ permutations. As illustrated by Fig. 14(c), suppose the c_b and c_{new} are candidate C and E, respectively. The resulting new SP is obtained by inserting E right in front of the position of C, as shown by Fig. 14(d).

V. Experimental Results

We implement our algorithm in C++ and test on an Intel Core 3.0 GHz Linux machine with 32 G RAM. LKH [23] is chosen as the solver for min-cost Hamilton path, where the maximum problem size is set as 50. In other words, if the number of characters in a row exceeds 50, we will chop it into smaller segments, each less than 50, and perform LKH individually. Moreover, Parquet [21] is adopted as our simulated annealing framework.

To test the efficiency of proposed methods, we randomly generate eight benchmarks. The size of stencil is set as $100\,\mu\text{m} \times 100\,\mu\text{m}$, and a total number of 1000 character candidates with unique size are generated. The sharable blanking area within each candidate is randomly decided, uniformly distributed between 0% and 50% character width. For the special case of 1-D problem, the blanking space along the vertical direction is set as a constant value. Moreover, for each candidate c_i, we randomly assign a triple of value (r_i, n_i^{VSB}, n_i^{CP}) as its referred time in chip, and respective number of shots by VSB and CP. n_i^{VSB} is made five to ten times larger than n_i^{CP}.

The detailed statistical data for individual testcase is shown in Table II. The first column denotes the name of benchmarks, where "1D-x" and "2D-x" are applied for one and 2-D problem, respectively. "csize" is the size of each character candidate, formatted by "$\mu\text{m} \times \mu\text{m}$." The units of all the other columns are "$1e^4\,\mu\text{m}^2$." "Total area" shows the total area of all the character candidates, and "total blanks" is the summation of their sharable blanking space. "Optimal area" is computed as "total area" minus "total blanks," typically larger than the area of given stencil. This matches the fact that even under best possible case of stencil design, where all the blanking areas are indeed shared by adjacent characters, the entire set of the candidates can not be fully pushed into the stencil.

For comparative reasons, we implement two different stencil design approaches. The first one, *No-Overlap*, is based on the

TABLE II
STATISTICS ON TESTCASES

ckts	csize	Total Area	Total Blanks	Optimal Area
1D-1	3.8 × 3.8	1.444	0.416	1.028
1D-2	4.0 × 4.0	1.6	0.479	1.121
1D-3	4.2 × 4.2	1.764	0.514	1.25
1D-4	4.4 × 4.4	1.936	0.569	1.367
2D-1	3.8 × 3.8	1.444	0.414	1.03
2D-2	4.0 × 4.0	1.6	0.529	1.071
2D-3	4.2 × 4.2	1.764	0.662	1.102
2D-4	4.4 × 4.4	1.936	0.774	1.162

work of [15], where no overlapped characters are allowed. A little difference is that, in its implementation, only one stencil with unique character size is considered. For our problem, their algorithm is somewhat degenerated into a method of selecting the most profitable candidates, which profit is judged by $r_i(n_i^{\mathrm{CP}} - n_i^{\mathrm{VSB}})$. In the second comparative approach, *Greedy*, possible sharing is taken into account, but a greedy methodology is applied to choose character candidates. In the 1-D problem, only the first phase of heuristic descending best-fit (DBF) packing in Section III-A is performed. For 2-D problem, 2-D DBF packing is conducted.

A. 1-D Stencil Design

Table III lists the comparison of stencil design in the 1-D case. "#shot" shows the total processing time (number of shots) of the circuit by using corresponding stencil design methodologies, which is computed by the equation of Objective 1. "#char" is the number of characters that fits into stencils, and "#*CPU*" tells the runtime of these stencil optimization methods in terms of seconds.

As we can see, compared to *No-overlap*, we are able to put on average 42% more characters on the stencil, and reduce the total projection time (number of shots) by 51%. With respect to *Greedy* algorithm, our approach still achieves on average 14% more projection time reduction, by allowing 7% more characters placed. The CPU time of our approach is relatively large but its absolute value is only around 20s. These results show the effectiveness and efficiency of our proposed three-phase iterative refinement algorithm.

For this special one-dimension problem, *Greedy* looks also quite useful. The reason is that the vertical blanking spaces of these candidates are uniform in this case, and have been fully shared during the stencil design.

Next, we investigate the benefits achieved by sharing horizontal and vertical spacing, respectively. Note that we can enable or disable vertical space margin by setting appropriate number of available rows. In Fig. 15, light blue column "No-overlap" illustrates the number of characters put on the stencil when no space overlapping is allowed. Dark blue column "H-overlap" lists the results, when horizontal blank sharing is enabled but vertical is forbidden. Green column "H+V overlap" shows the character number when both inside-row and inter-row sharing are enabled which is our default setting. The only difference between "H-overlap" and "H+V overlap" is that, in "H+V overlap," the overlapping between rows plays roles.

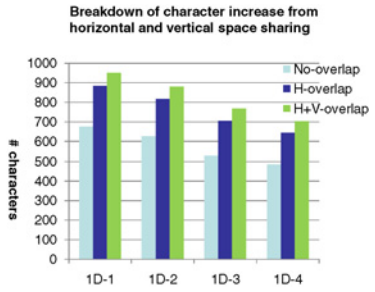


Fig. 15. Investigation of the throughput improvement from horizontal and vertical overlapping, respectively.

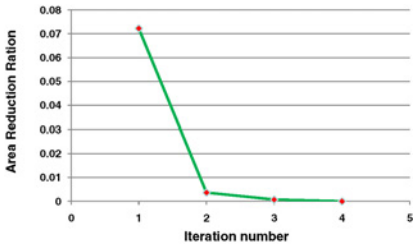


Fig. 16. Trend of area reduction ratio with respect to the number of iterations (for 1D-1).

As the result shows, the improvement of throughput is mainly from sharing horizontal spacing within each row. This alone increases the number of used characters by 32%. The vertical margin optimization contributes another 10% character number increase. This demonstrates our three-step iterative refinement method, designed for horizontal overlapping optimization, is effective and contributes major portion of throughput improvement.

To illustrate how our algorithm performs over iterations, we also plot the trend of area reduction for testcase 1-D in Fig. 16. The y-axis shows the amount of reduced area after we perform Hamilton-based compaction and multirow swapping in each iteration, with normal to the stencil area. It can be concluded that the first iteration plays the key role in our optimizations.

Further, we break down and study the statistics of different steps of proposed algorithm. In Fig. 17, [1]–[3] correspond to the stages of 1-D bin-packing, single row reordering, multirow, and inter-stencil tuning, respectively. Here, we only focus on one testcase 1D-1. Fig. 17(a) illustrates the CPU time distribution of these steps. On the other hand, Fig. 17(b) reflects their individual contributions for throughput improvement. In our three-step iterative refinement flow, only the first step 1-D bin-packing directly increases the number of characters. Steps 2 and 3 are all used for compacting currently-existing characters on the stencil, which creates more spacing for pushing more candidates in bin-packing the next iteration. Therefore, we only list the reduced packing area by steps 2 and 3 in Fig. 17(b). Seen from Fig. 17, the major CPU bottleneck comes from single row reordering and inter-stencil tuning. Step 2 of with-in row optimization contributes most for packing area reduction, which is the key step for further occupation of additional characters.

To evaluate how far *Greedy* and our proposed solution are close to the optimal one, we also implemented an ILP

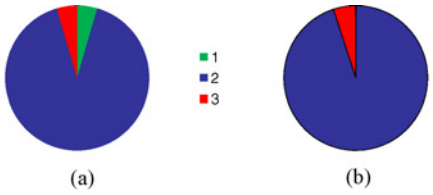


Fig. 17. Distribution of CPU time and area saving of distinct steps in our proposed 1-D flow (for testcase 1D-1 only). (a) Breakdown of CPU time. (b) Breakdown of area saving.

TABLE III
RESULT COMPARISON FOR 1-D PROBLEM

	NO-OVERLAP			GREEDY			Our Approach		
ckts	#shot	#char	$CPU(s)$	#shot	#char	CPU	#shot	#char	CPU
1D-1	28 654	676	1.2	13 528	901	2.2	10 083	951	22.3
1D-2	41 727	625	1.1	17 929	836	2.1	14 921	880	21.8
1D-3	38 460	529	0.9	25 155	727	1.9	22 503	768	20.6
1D-4	41 260	484	0.8	29 462	665	1.8	26 756	702	20.1
Total	150 101	2314	4	86 074	3129	8	74 263	3301	84.8
ratio	2.0	1	0.05	1.16	1.35	0.10	1	1.42	1

TABLE IV
RESULT COMPARISON WITH RESPECT TO ILP-DERIVED OPTIMAL SOLUTION

ckts	#char	#shot	CPU (s)
NO-OVERLAP	16	1096	0.00
GREEDY	20	846	0.02
Our Approach	22	704	0.1
ILP	24	515	>3600

formulation, and considered its solution as "optimal." Since the complexity of ILP is forbiddingly high, we pick up one small design. The sizes of candidates and stencil are set as $4\,\mu\text{m} \times 4\,\mu\text{m}$ and $16\,\mu\text{m} \times 16\,\mu\text{m}$, respectively. The number of candidates is 30. The results are shown in Table IV.

In terms of shot number, our approach is 56% and 14% better than *Nonoverlap* and *Greedy* respectively, which shows the effectiveness of proposed flow. When compared to ILP, it is 27% worse. This indicates possible improvement space of our methodology, which we are going to work on in future. On the other side, as we can see, ILP is very slow even for such a small testcase, taking more than 1 h. All the other three methods finish in seconds. It is easily seen that ILP definitely cannot handle our real case of $1e^3$ candidates.

For *Greedy* algorithm, it is 41% worse than ILP but only 14% inferior than our approach, judged by shot number. This again supports our previous conclusion that the greedy algorithm in 1-D is not that bad as expected.

B. 2-D Stencil Design

In our 2-D stencil design, we reused the same parameters in Parquet for our simulated annealing framework, including initial and stopping temperature, temperature degrading factors, and so on. There are three types of different SP perturbation methods, throughput driven swapping and slack based insertion (proposed in Section IV-D), plus random permutation. Random permutation is adopted here just for avoiding getting stuck into local space. In our implementation, we rely on a pseudo number generator to determine which type of perturbation to apply in each iteration.

TABLE V
RESULT COMPARISON FOR 2-D PROBLEM

	NO-OVERLAP			GREEDY			Our Approach			
ckts	#shot	#char	CPU	#shot	#char	CPU	#shot	#char	CPU	$CPU(eval)$
2D-1	23 319	676	1.3	26 832	625	2.3	16 877	803	242	171
2D-2	29 368	576	1	25 977	642	2.6	20 141	750	235	170
2D-3	32 399	526	0.9	30 411	558	2.5	23 850	688	221	155
2D-4	35 410	474	0.8	31 930	531	2.7	25 278	660	214	161
Total	120 496	2252	4	115 150	2356	10.1	86 146	2901	912	657
ratio	1.40	1	0.004	1.33	1.05	0.011	1	1.30	1	0.72

Table V lists the comparison of stencil design in the general 2-D case. Column $CPU(eval)$ shows the runtime for SP-evaluation phase. The meaning of labels is the same as Table III. Compared to *No-overlap* and *Greedy* methods, in average, our proposed SP-based algorithm places 28% and 24% more characters on stencil, which reduces the projection time (number of shots) by 31% and 25%, respectively. The *Greedy* algorithm does not work that well in this 2-D problem, because the blanking area varies in both horizontal and vertical directions and the native first-bin-best-fit packing very easily gets stuck in local optima. Moreover, from Column $CPU(eval)$, we observe that the major portion time of our algorithm is spent on SP-evaluation (around 73%), even with the applicant of our proposed look-ahead pruning algorithm.

Due to 2-D optimization, the runtime of our approach is much longer than 1-D problem comparatively. It takes a few hundred seconds, but is still satisfactory. The design of stencil is only a one-time process before projecting large volume of chips by EBL. Several minutes preprocessing time is relatively very tiny in the whole manufacturing procedure.

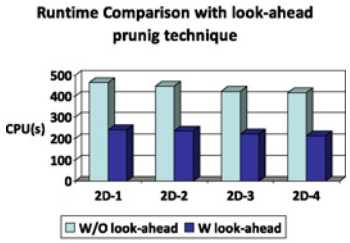
Next, we will show the effectiveness of our proposed look-ahead sequence pair evaluation technique in Fig. 18. "Without look ahead" applies the extension of longest path based evaluation method in Section IV-B1, while "With look ahead" adopts our proposed look-ahead pruning technique based on longest subcommon sequence computation. For these two versions, we achieve the same solution quality, the throughput. In terms of runtime, our approach can achieve about 48% reduction. When judged by the step of SP evaluation only, the CPU time decreases by 66%. That also demonstrates our statement in Section IV-B2) that although the theoretical timing complexity of Algorithm 3 is as much as naive Algorithm. 2, it performs much better in practice.

Similar to the 1-D problem, we also study the individual contributions of different SP perturbation methods applied in our simulated annealing flow. As shown in Fig. 19, the proposed throughput driven and slack base methods perform effectively, which contribute around 80% of total projection time reduction. Their runtimes are comparable, not showing outstanding outlier.

As the last experiment, we apply the proposed 2-D simulated annealing based framework to solve 1-D testcases. The result is listed in Table. VI. We added two new columns CP and VSB to show their shot numbers respectively. As we can show, 2-D algorithm generates 24% more shot number and

TABLE VI
RESULT COMPARISON WHEN APPLYING 2-D METHODS TO 1-D PROBLEM

	Our Approach of 1-D					1-D Greedy					Our Approach of 2-D				
ckts	#shot	#CP	#VSB	#char	$CPU(s)$	#shot	#CP	#VSB	#char	$CPU(s)$	#shot	#CP	#VSB	#char	CPU
1D-1	10 083	6105	3978	951	22.3	13 528	5800	7728	901	2.2	16 345	5449	10 896	794	257
1D-2	14 921	6569	8352	880	21.8	17 929	6262	11 667	836	2.1	20 986	4796	16 190	692	245
1D-3	22 503	5334	17 169	768	20.6	25 155	5043	20 112	727	1.9	25 906	4504	21 402	648	240
1D-4	26 756	4864	21 892	702	20.1	29 462	4611	24 851	665	1.8	29 157	4372	24 785	589	228
Total	74 263	22 872	51 391	3301	84.8	86 074	21 716	64 358	3129	8	92 394	19 121	73 273	2723	970
ratio	1	1	1	1	1	1.16	0.96	1.25	0.95	0.09	1.24	0.83	1.43	0.83	11.43



(a)

Fig. 18. Total CPU time with and without look-ahead sequence pair evaluation.

(a) (b)

Fig. 19. Distribution of CPU time and projection time reduction of different types of SP permutations in our proposed simulated annealing framework (for testcase 2D-1 only). (a) CPU time. (b) Projection time reduction.

runs much slower than our proposed 1-D three-step iterative method. Its solution quality is also 7% worse than greedy 1-D DBF packing algorithm. The reasons are two-fold. First, the 2-D algorithm is not aware of the fact that the vertical blank space of 1-D cases is uniform and can be utilized optimally. Moreover, 2-D algorithm is rather heuristic, and not able to handle horizontal overlapping as effectively as Hamilton cycle based compacting proposed in 1-D three phase refinement flow, or DBF packing algorithm in 1-D greedy approach.

VI. CONCLUSION

In the future, to further improve the algorithm runtime, we will try to substitute the annealing-based 2-D stencil design by some nonstochastic packing algorithm, e.g., the work in [24]. We also plan to work on multiple-stencil optimization with massive parallel electronic beam projection as well as electronic beam lithography friendly standard cell design, placement and routing algorithm. In addition, we are interested in expanding our investigation on research opportunity between physical design (e.g., [25], [26]) and other emerging lithography, such as triple patterning, nanoimprint, and EUV.

ACKNOWLEDGMENT

The authors would like to thank Dr. G.-J. Nam at IBM Austin Research, Austin, TX, for helpful discussions on this problem.

REFERENCES

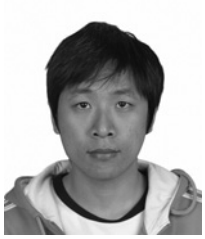
[1] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition for double patterning lithography," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2008, pp. 465–472.
[2] K. Yuan, J.-S. Yang, and D. Z. Pan, "Double patterning layout decomposition for simultaneous conflict and stitch minimization," in *Proc. Int. Symp. Phys. Des.*, Mar. 2009, pp. 107–114.
[3] K. Yuan, K. Lu, and D. Z. Pan, "Double patterning lithography friendly detailed routing with redundant via consideration," in *Proc. Des. Autom. Conf.*, Jun. 2009, pp. 63–64.
[4] K. Yuan and D. Z. Pan, "WISDOM: Wire spreading enhanced decomposition of masks in double patterning lithography," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2010, pp. 32–38.
[5] J.-S. Yang, K. Lu, M. Cho, K. Yuan, and D. Z. Pan, "A new graph-theoretic, multi-objective layout decomposition framework for double patterning lithography," in *Proc. Asia South Pac. Des. Autom. Conf.*, Jan. 2010, pp. 637–644.
[6] B. Yu, K. Yuan, and D. Z. Pan, "Layout decomposition for triple patterning lithography," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2011.
[7] A. Fujimur, "Beyond light: The growing importance of E-beam," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2009.
[8] A. Fujimur, "Design for E-beam: Getting the best wafers without the exploding mask costs," in *Proc. Int. Symp. Quality Electron. Des.*, Mar. 2010.
[9] A. Fujimura, T. Mitsuhashi, K. Yoshida, S. Matsushita, L. L. Chau, T. D. T. Nguyen, and D. MacMillen, "Stencil design and method for improving character density for cell projection charged particle beam lithography," U.S. Patent 20090325085, Jan. 2010.
[10] H. C. Pfeiffer, "New prospects for electron beams as tools for semiconductor lithography," *Proc. SPIE*, vol. 7378, p. 737802, May 2009.
[11] T. Fujino, Y. Kajiya, and M. Yoshikawa, "Character-build standard-cell layout technique for high-throughput character-projection EB lithography," *Proc. SPIE*, vol. 5853, p. 160, Jul. 2005.
[12] M. Sugihara, T. Takata, K. Nakamura, Y. Matsunaga, and K. Murakami, "A CP mask development methodology for MCC systems," *Proc. SPIE*, vol. 6283, p. 62833J, May 2006.
[13] M. Sugihara, K. Nakamura, Y. Matsunaga, and K. Murakami, "CP mask optimization for enhancing the throughput of MCC systems," *Proc. SPIE*, vol. 6349, p. 63494B, Oct. 2006.
[14] Y. Matsunaga, M. Sugihara, and K. Murakami, "Technology mapping technique for enhancing throughput of multi-column-cell systems," *Proc. SPIE*, vol. 6517, p. 65170Z, Mar. 2007.
[15] M. Sugihara, "Optimal character-size exploration for increasing throughput of MCC lithographic systems," *Proc. SPIE*, vol. 7271, p. 72710L, Feb. 2009.
[16] Y. Zhang and C. Chu, "RegularRoute: An efficient detailed router with regular routing patterns," in *Proc. Int. Symp. Phys. Des.*, Mar. 2011, pp. 45–52.
[17] Y. Zhang and C. Chu, "Fastroute 3.0: A fast and high quality global router based on virtual capacity," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2008, pp. 344–345.
[18] Y. Zhang and C. Chu, "CROP: Fast and effective congestion refinement of placement," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2009, pp. 344–350.

[19] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 15, no. 12, pp. 1518–1524, Dec. 1996.
[20] X. Tang, R. Tian, and M. Wong, "Fast evaluation of sequence pair in block placement by longest common subsequence computation," in *Proc. Des. Autom. Test Eur.*, Mar. 2000, pp. 106–111.
[21] S. H. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.
[22] J. Z. Yan and C. Chu, "Optimal slack-driven block shaping algorithm in fixed-outline floorplanning," in *Proc. Int. Symp. Phys. Design*, Mar. 2012.
[23] [Online]. Available: http://www.akira.ruc.dk/∼keld/research/LKH
[24] J. Z. Yan and C. Chu, "DeFer: Deferred decision making enabled fixed-outline floorplanning algorithm," *IEEE Trans. Comput.-Aided Des.*, vol. 29, no. 3, pp. 367–381, Mar. 2010.
[25] J. Z. Yan, N. Viswanathan, and C. Chu, "Handling complexities in modern large-scale mixed-size placement," in *Proc. IEEE/ACM Des. Autom. Conf.*, Jul. 2009, pp. 436–441.
[26] J. Z. Yan and C. Chu, "Handling complexities in modern large-scale mixed-size placement," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 30, no. 7, pp. 1020–1033, Jul. 2011.

Kun Yuan received the B.S. degree in electronic engineering information science from the University of Science and Technology of China, Hefei, China, in 2004, and the Ph.D. degree in electrical and computer engineering from the University of Texas, Austin, in 2010.

He was with TeraRoute, Austin, TX, in 2007 and with NVIDIA, Santa Clara, CA, in 2009. He is currently a Senior Member of Technical Staff with Cadence Design Systems, San Jose, CA. His current research interests include physical design automation for manufacturability and numerical optimization.

Dr. Yuan received the ISPD Routing Contest Award in 2007, and three Best Paper Awards (ASPDAC 2010, ISPD 2011, and the IBM Research 2010 Pat Goldberg Memorial Best Paper Award in CS/EE/Math).



Bei Yu received the M.S. degree in computer science from Tsinghua University, Beijing, China, in 2010. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Texas, Austin.

His main research interests include physical design, design for manufacturability, and optimization algorithms with applications in very large scale integration computer-aided design.

David Z. Pan (S'97--M'00--SM'06) received the B.S. degree from Peking University, Beijing, China, and the M.S. and Ph.D. degrees from the University of California, Los Angeles (UCLA).

From 2000 to 2003, he was a Research Staff Member with the IBM T. J. Watson Research Center, Yorktown Heights, NY. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Texas, Austin. He has published over 160 papers in international conferences and journals, and holds eight U.S. patents. His current research interests include nanometer physical design, design for manufacturability and reliability, vertical integration design and technology, and design/computer-aided design (CAD) for emerging technologies.

Dr. Pan has served as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS PART I and PART II, the *Journal of Computer Science and Technology*, and the IEEE CIRCUITS AND SYSTEMS SOCIETY NEWSLETTER. He has served as the Chair of the IEEE CANDE Committee and the ACM/SIGDA Physical Design Technical Committee. He is in the Design Technology Working Group of the International Technology Roadmap for Semiconductor. He has served in the technical program committees of major VLSI/CAD conferences, including ASPDAC (Subcommittee Chair), DAC (Subcommittee Chair), DATE, ICCAD (Subcommittee Chair), ISPD (Program Chair), ISQED (Topic Chair), ISCAS (CAD Track Chair), SLIP (Publication Chair), GLSVLSI, ACISC (Program Co-Chair), ICICDT (Award Chair), and VLSI-DAT (EDA Track Chair). He was the General Chair of ISPD 2008, the General Chair of ACISC 2009, and the Steering Committee Chair of ISPD 2009. He received a number of awards for his research contributions and professional services, including the ACM/SIGDA Outstanding New Faculty Award in 2005, the NSF CAREER Award in 2007, the SRC Inventor Recognition Award thrice in 2000 and 2008, the IBM Faculty Award four times in 2004–2006, 2010, the UCLA Engineering Distinguished Young Alumnus Award in 2009, seven Best Paper Awards (SRC Techcon 1998 and 2007, DATE 2009, ICICDT 2009, ASPDAC 2010, ISPD 2011, and the IBM Research 2010 Pat Goldberg Memorial Best Paper Award in CS/EE/Math) and many other Best Paper Award nominations from DAC/ICCAD/ASPDAC/ISPD, ISPD Routing Contest Awards in 2007, the eASIC Placement Contest Grand Prize in 2009, the IBM Research Bravo Award in 2003, the Dimitris Chorafas Foundation Research Award in 2000, and the ACM Recognition of Service Award in 2007 and 2008. He was an IEEE CAS Society Distinguished Lecturer from 2008 to 2009.