INTEGRATION the VLSI journal 58 (2017) 47–54

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

# Stitch aware detailed placement for multiple E-beam lithography

Yibo Lin[a,*], Bei Yu[b], Yi Zou[a,c], Zhuo Li[d], Charles J. Alpert[d], David Z. Pan[a]

[a] ECE Department, University of Texas at Austin, Austin, TX, USA
[b] CSE Department, The Chinese University of Hong Kong, NT, Hong Kong
[c] College of Engineering and Applied Sciences, Nanjing University, Nanjing, China
[d] Cadence Design Systems, Inc., Austin, TX, USA

ARTICLE INFO

*Keywords:*
Multiple electron beam lithography
Stitch error
Detailed placement
Dynamic programming

ABSTRACT

In multiple electron beam lithography (MEBL), a layout is split into stripes and the layout patterns are cut by stripe boundaries, then all the stripes are printed in parallel. If a via pattern or a vertical long wire is overlapping with a stitch, it may suffer from poor printing quality due to the so called *stitch error*; then the circuit performance may be degraded. In this paper, we propose a comprehensive study on the stitch aware detailed placement to simultaneously minimize the stitch error and optimize traditional objectives, e.g., wirelength and density. Experimental results show that our algorithms are very effective on modified ICCAD 2014 benchmarks that zero stitch error is guaranteed while the scaled half-perimeter wirelength is very comparable to a state-of-the-art detailed placer. In addition, our technique is very generic that it is applicable to many other placement targets, such as local congestion optimization, which is also demonstrated in the experimental results.

## 1. Introductions

Due to the capability of accurate pattern generation, e-beam lithography (EBL) is a promising candidate for next generation lithography technologies for sub-14 nm nodes, along with other techniques such as extreme ultra violet (EUV) and directed self-assembly (DSA) [1–3]. However, low throughput is still the bottleneck of an EBL system. Recently, an extended EBL technique, multiple e-beam lithography (MEBL), is proposed to improve manufacturing throughput using parallel beam printing [4]. MEBL system utilizes thousands of parallel beams to write multiple layout patterns simultaneously. Industry has already explored different MEBL implementations and has demonstrated promising performance in terms of both lithography accuracy and throughput [5,6].

In MEBL manufacturing process, a layout is split into stripes, and the boundary between two touching stripes is defined as a stitch line. Each stripe has width of 50~200 μm, and different stripes are printed simultaneously through different electron beams. Although the parallel writing scheme can dramatically improve the system throughput, it also introduces serious printability issues. That is, each stitch can introduce so called **stitch error**, in an area with width around 15 nm [5]. If a pattern is overlapping with a stitch, it may suffer from poor printing quality due to the stitch error. Therefore, if not carefully designed, due to the shape distortion, an MEBL system may confront yield issue or even functional error.

We observe very significant shape distortions on via patterns and long vertical wires. Fig. 1 shows two SEM images of shape distortion on via layer and metal layer, respectively. In Fig. 1(a) we can see that all vias are very regular inside the beam stripes. However, at the stripe boundaries, the vias suffer from obvious distortions and irregular shrinking. In Fig. 1(b) we can see that the vertical wires are malformed in the stitch regions. Similar observations were also reported by Fang et al. [7] that the vertical wires are more susceptible to stitch errors than the horizontal wires.

There are several methods to minimize the impacts of stitch errors from lithography perspective, e.g., avoiding dividing a critical pattern into adjacent sub-fields [8], using different field sizes [9], or reducing the field size [10]. Recently, Fang et al. [7] considered the stitch error during detailed routing stage. However, detailed routing is a very late stage in physical design flow, thus there may exist some stitch errors difficult to be removed. For instance, stitch errors from vias dropped on pins of a standard cell cannot be optimized during routing stage. There are various detailed placement algorithms to address other emerging issues in advanced technology nodes, such as multiple patterning lithography [11–13], N10 design rules [14], multiple-row height cells [15–17], etc., which are summarized in [18].

In this work we propose a comprehensive study to consider the stitch error removal in detailed placement. We can directly optimize the positions of both vias and intra-cell vertical wires. In addition, we consider local congestion, thus a router (e.g. [7]) has more routing options to effectively remove stitch errors in higher metal layers. Fig. 2 shows a placement example with three gates, where the density of

* Corresponding author.

http://dx.doi.org/10.1016/j.vlsi.2017.02.004

Available online 20 February 2017

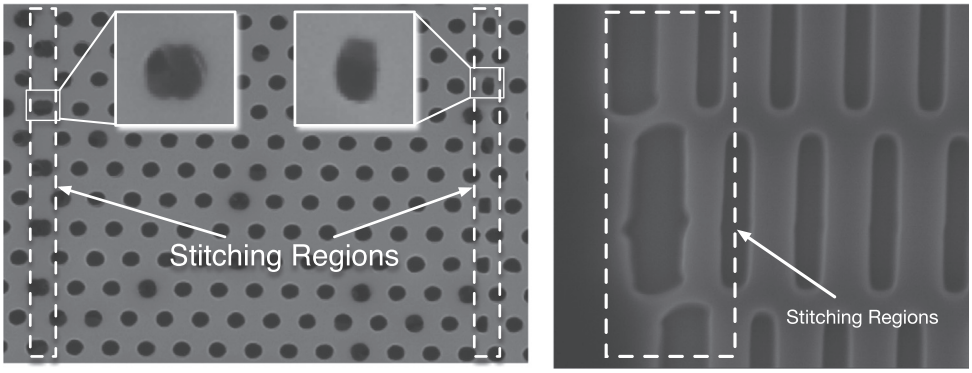


**Fig. 1.** SEM images of stitch error for (a) via layer and (b) metal layer vertical wires.

vertical metal1 segments varies from cell to cell. Some cells are more susceptible to stitch errors as they have vertical wire segments distributed at every site, while other cells have more space to avoid stitch errors. The comparison between Fig. 2(a) and Fig. 2(b) shows that it is possible to smartly avoid stitch errors with small cell movement.

To the best of our knowledge, this is the first work taking stitch errors into consideration in placement stage. Our contributions are summarized as follows.

- We propose a comprehensive detailed placement study to simultaneously minimize the stitch error and optimize traditional objectives, e.g., wirelength and density.
- We develop a swap-based detailed placement engine with an optimal stitch aware single row placement.
- We present an $O(nM)$ pruning technique to speed-up the single row problem, where $n$ and $M$ are number of cells in a row and maximum displacement respectively.
- Our pruning technique is very generic that it is applicable to conventional placement and other applications. We show that the pruning technique is also adjustable for local congestion optimization.

The rest of this paper is as follows. Section 2 introduces the stitch constraints and the problem formulation. Section 3 explains the optimization algorithms in detail. Section 4 lists the experimental results, followed by conclusion in Section 5.

## 2. Preliminaries and problem formulation

In an MEBL system, stitch lines repeat periodically with equal intervals. If a standard cell is not carefully placed and overlaps with one stitch line, it may suffer from stitch error. In this work we consider three kinds of possible stitch errors, as follows. (1) **Stitch over via**: if a via is cut by a stitch line, it can lead to potential disconnection. (2) **Vertical routing**: a vertical routing segment suffers more from stitch lines than horizontal lines. (3) **Short polygon**: short horizontal routing segment with vias may also result in problem.

To accurately capture a stitch error, we partition each cell into sites with width equal to the poly pitch. Since some intra-cell segments or vias are very susceptible to stitches, we note those sites covered by these segments/vias as *dangerous sites*. For example, Fig. 3 shows the dangerous sites of cell BUF_X8. Note that for simplicity, here only intra-cell segments are illustrated. A stitch error happens if one dangerous site overlaps with an MEBL system stitch line.

This work adopts scaled half-perimeter wirelength (sHPWL) from ICCAD 2013 placement contest, defined as follows.

$$sHPWL = HPWL \times (1 + \alpha \times P_{ABU}), \quad (1)$$

where $\alpha$ is set to 1, and HPWL denotes half-perimeter wirelength. $P_{ABU}$ represents *ABU penalty* to evaluate the placement congestion. Please refer to [19] for more details regarding the $P_{ABU}$ calculation.

**Problem 1** (*Stitch Aware Detailed Placement*). Given an initial detailed placement with the information of dangerous sites for each standard cell, we seek a legal placement to minimize the stitch errors and the sHPWL, simultaneously.

After solving Problem 1, we further perform a local congestion refinement step to improve local routability and pin access without introducing any additional stitch error and demonstrate the flexibility of the proposed algorithm.

**Problem 2** (*Stitch Aware Local Congestion Refinement*). Given a detailed placement solution without stitch errors, refine local congestion while minimizing displacement without introducing any stitch error.

It should be noted that Problem 2 aims at smoothing congested regions, as shown in Fig. 4 where cells are shown with dangerous sites and pins are as cross marks. The region in Fig. 4(a) is very congested due to large number of pins in the region. We can insert whitespaces to relieve congestion in Fig. 4(b) while it is necessary to avoid stitch errors at the same time. It is suitable to solve it by minimization of a maximum cost, which helps relieve congestion without large perturba-

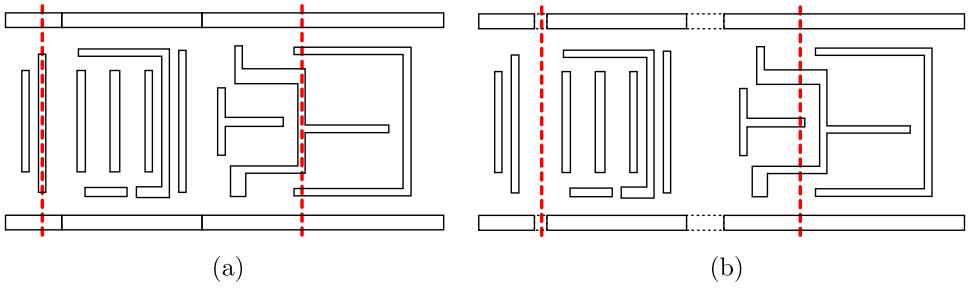


**Fig. 2.** An example of (a) stitch errors in placement and (b) e-beam friendly placement.

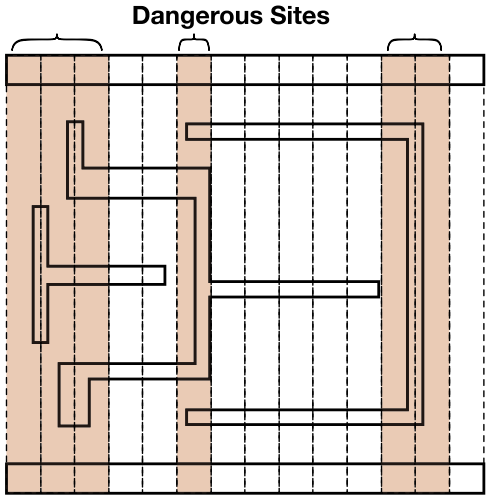


**Fig. 3.** An example of cell BUF_X8, where dangerous sites are labeled as red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

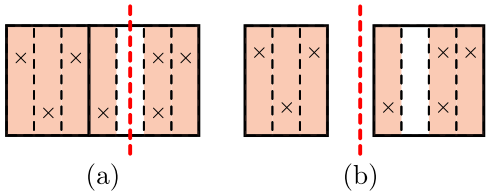


**Fig. 4.** Example of local cells and pins and (a) local congested region and (b) congestion improved by inserting whitespaces.

tion to the layout. Further details will be discussed in Section 3.3.

## 3. Detailed placement algorithms

In this section we describe the details of our placement algorithms. As shown in Fig. 5, our framework mainly consists of two stages. In the first stage, single row based approach is applied to optimize wirelength and stitch errors optimally. If all stitch errors are removed successfully by this stage, we directly output placement solutions. Otherwise, in the second stage, cell swapping and movement are introduced to improve both wirelength and congestion. Note that the stitch error is considered through the whole flow.

### 3.1. Single row placement

As a powerful approach in detailed placement, single row based placement is widely studied in both conventional placement [20–22] and lithography aware application, such as triple patterning lithography (TPL) compliance [23–26,13]. If there are fixed macros in the layout, conventional single row algorithms (e.g. Abacus [22]) divide a row into several sub-rows. However, this strategy is not suitable for MEBL application, as the stitch lines are soft constraints rather than hard constraints. In TPL compliance, the main challenge lies in the distance between abutting cells, while the stitch errors in MEBL are not related to neighboring cells. In addition, in the single row algorithm proposed by [23], a graph based approach is applied to find optimal solution in $O(mnK)$. Here $m$ is the site number in the row, $n$ is the cell number, and $K$ is the number of pre-coloring solutions for each cell. Usually $m$ is a very large number, thus this algorithm may suffer from runtime for large size circuits.

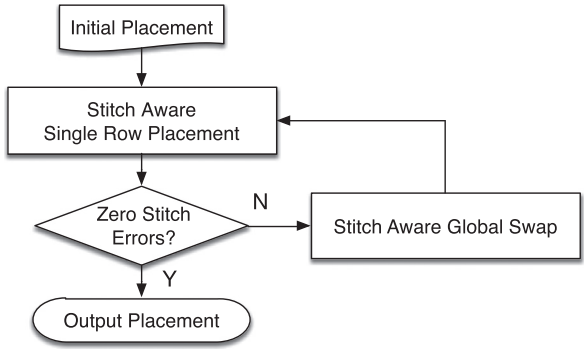


**Fig. 5.** Overall flow of our stitch aware detailed placement.

In this paper we adapt a dynamic programming based algorithm [27] to solve single row detailed placement. Different from other techniques (e.g. [22]), it can naturally handle both hard constraints (fixed macros) and soft constraints (stitch errors). Each cell is associated with a movable range, which is usually a finite site candidates. The dynamic programming scheme is able to achieve optimal solution for combined cost functions, such as movement, wirelength and stitch errors. Note that comparing with [27], we significantly improve the runtime complexity while still maintaining the optimality.

For convenience, Table 1 lists some definitions used in the single row placement. The algorithms are described with the concept of displacement values of a cell. For example, if cell $c_i$ is originally at position $p^0{}_i$ and it is then moved to position $p^1{}_i$, its displacement value is $p_i^1 - p_i^0$. We also describe positions with displacement values of a cell; e.g., the position with displacement value $-1$ of cell $c_i$ denotes the position $p_i^0 - 1$. The algorithm for single row placement is explained with a graph in Fig. 6. All candidate displacement values of a cell is listed as a column of nodes. There is an edge between two nodes if they can reach their displacement values without overlap. For example, the maximum displacement for cell $c_i$ is $M$, so the displacement range for $c_i$ is from $-M$ to $M$, the value of which is marked in the node. Each edge also contains a cost according to Eq. (2). Two additional nodes, $s$ and $t$, are inserted to the graph. The problem is stated as finding the path with lowest cost from node $s$ to node $t$, which can be solved with dynamic programming.

The $cost_i(p_i)$ function in the experiments is as follows,

$$cost_i(p_i) = \tau \cdot WL(p_i) + \phi \cdot MOV(p_i) + \nu \cdot SP(p_i), \tag{2a}$$

$$SP(p_i) = \begin{cases} 0, & \text{no stitch,} \\ \text{large number,} & \text{generate a stitch error,} \end{cases} \tag{2b}$$

where $WL$ denotes wirelength cost, $MOV$ denotes movement, and $SP$ denotes stitch error penalty. $SP$ is set to a very large number when a stitch error is generated, e.g., half-perimeter of the layout. In our experiments, $\tau$, $\phi$, and $\nu$ are set to 10, 1, and 1. In legalization step, we simply set $\tau$ and $\nu$ to zero.

Given an ordered sequence of cells $S$, to calculate wirelength cost for cell $c_i$, we need to fix the positions for all other cells. The wirelength

**Table 1**
Notations used in single row placement.

| | |
|---|---|
| $M$ | Maximum displacement for a cell. |
| $p_i^0$ | Initial position of Cell $c_i$. |
| $p_i$ | The position of Cell $c_i$, $p_i^0 - M \le p_i \le p_i^0 + M$. |
| $L_i$ | Ordered set of positions for cell $i$ and $pi \in L_i$ |
| $p_i^*$ | Final position of Cell $i$ after optimization |
| $\alpha_i(p_i)$ | solution of $c_1$ to $c_i$ in which $c_i$ is placed at $p_i$. |
| $t_i(p_i)$ | The cost of best placement solution from $c_1$ to $c_i$ in which $c_i$ is placed at $p_i$. |
| $\gamma_i(p_i)$ | The position of $c_{i-1}$ in the optimal solution of $c_1$ to $c_{i-1}$ in which $c_i$ is at $p_i$. |
| $r_i(p_i)$ | Whether the solution corresponding to $t_i(p_i)$ is inferior or not. |
| $cost_i(p_i)$ | The cost of $c_i$ when it is placed to $p_i$. |
| $w_i$ | Width of Cell $c_i$. |

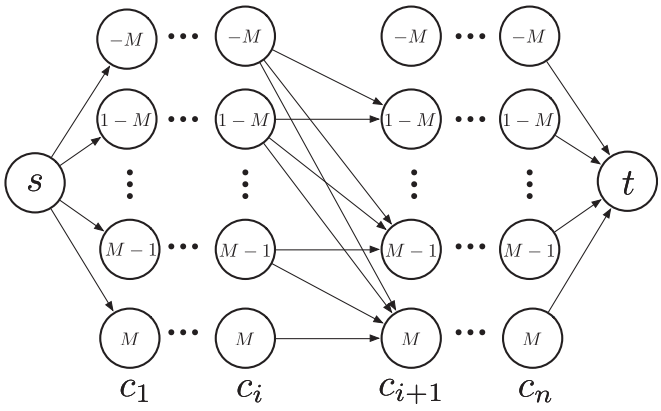


**Fig. 6.** Single row placement algorithm.

cost is determined by the bounding boxes of nets. But if cell $c_i$ has connection to any cell in $S$, the wirelength cost for cell $c_i$ cannot be determined since cells in $S$ are not fixed. To handle this, we introduce the wirelength model in [28] which ensures the wirelength cost for cell $c_i$ is independent to other cells in $S$, while the optimality of the solutions are maintained. If cell $c_i$ is connecting to any cell $c_j$ in the left of cell $c_i$ in the same placement row, we regard the position of $c_j$ as the left boundary of the row when computing the wirelength cost for $c_i$. Similarly, if cell $c_i$ is connecting to any cell $c_j$ in the right of cell $c_i$ in the same placement row, we regard the position of $c_j$ as the right boundary of the row. This model is widely used in ordered single row placement, which turns out to be equivalent to HPWL [29,28].

We can see that function $cost_i(p_i)$ is quite flexible, since we can include movement, wirelength and stitch errors. For hard constraints like fixed macros, we only need to set its maximum displacement $M$ to zero. For soft constraints like stitch lines, additional cost is applied if a cell has overlap with them.

**Lemma 1.** *Algorithm shown in Fig. 6 is optimal for cost function in* Eq. (2).

The proof is similar to that in [27], and is omitted here for brevity. The basic idea is that the optimal placement solution can be found through a shortest path from $s$ to $t$, and all the positions of cells can be derived from the displacement values of corresponding nodes. Since the constructed graph is a directed acyclic graph, the shortest path can be calculated using topological traversal in $O(nM^2)$ steps, where $n$ is the cell number in the row, and $M$ is the maximum displacement for each cell.

### 3.2. An $O(nM)$ pruning algorithm

The runtime complexity of the above single row placement is $O(nM^2)$. When $M$ is very large, the runtime becomes unacceptable. Here we propose a set of pruning techniques to achieve further speedup, while still keeping the optimality. In addition, we can theoretically prove that the runtime complexity can be improved from $O(nM^2)$ to $O(nM)$.

**Algorithm 1.** Single row placement with pruning

**Require:** A set of ordered cells $c_1$ to $c_n$ of a row.
**Ensure:** All the cells in the set are placed subjecting to optimal objective function.

1: $L_i \leftarrow [p_i^0 - M, p_i^0 + M], \forall\ i = 1\ to\ n$;
2: $t_1(p_1) \leftarrow cost_1(p_1),\ p_1 \in L_1$;
3: $t_i(p_i) \leftarrow \infty, i \leftarrow 2\ to\ n,\ p_i \in L_i$;
4: **for** each $c_i$, $i \leftarrow 2\ to\ n$ **do**
5: $N \leftarrow p_{i-1}^0 - M$;
6: **for** each $p_i \in L_i$ **do**
7: **for** each $p_{i-1} \in L_{i-1}$ where $p_{i-1} \geq N$ **do**
8: $cost \leftarrow t_{i-1}(p_{i-1}) + cost_i(p_i)$;
9: **if** $cost < t_i(p_i)$ **then**
10: $t_i(p_i) \leftarrow cost$;
11: $\gamma_i(p_i) \leftarrow p_{i-1}$;
12: $N \leftarrow p_{i-1}$;
13: **else**
14: break;
15: **end if**
16: **end for**
17: **end for**
18: Check inferior solutions and remove them from $L_i$;
19: **end for**
20: $cost_{min} \leftarrow \infty$;
21: **for** $p_n \in [p_n^0 - M, p_n^0 + M]$ **do**
22: **if** $t_n(p_n) < cost_{min}$ **then**
23: $cost_{min} \leftarrow t_n(p_n)$;
24: $p_n^* \leftarrow p_n$;
25: **end if**
26: **end for**
27: **for** $i \leftarrow n$ down to 2 **do**
28: $p_{i-1}^* \leftarrow \gamma_i(p_i)$;
29: **end for**

The details of our $O(nM)$ implementation are shown in Algorithm 1. The main difference between the problems in [23,27] and our problem lies in the cost function. That is, the cost functions for a cell in the former problems depend on other cells, such as the distance or coloring cost between two abutting cells, while the cost defined in Eq. (2) is only related to the cell itself; i.e., it is independent to any other cell. Due to the independence in the cost function, we can minimize the total cost with $O(nM)$ time complexity. Our speedup technique is **generic** that it can also be applied into conventional detailed placement and legalization with an objective like wirelength or movement.

**Lemma 2.** *Comparing two solutions $\alpha_i(p_i)$ and $\alpha_i(q_i)$, if $t_i(p_i) \geq t_i(q_i)$ and $p_i \geq q_i$, then $\alpha_i(p_i)$ is inferior to $\alpha_i(q_i)$.*

**Proof.** Suppose cell $c_i$ has two candidate positions $p_i$ and $q_i$, where $p_i \geq q_i$ and $t_i(p_i) \geq t_i(q_i)$. Now consider any candidate position $p_{i+1}$ for cell $c_{i+1}$. If cell $c_i$ can be placed at $p_i$ without overlapping with cell $c_{i+1}$, then $q_i$ is also a legal position for cell $c_i$. We can always move cell $c_i$ from $p_i$ to $q_i$ for better cost, because the total cost at cell $c_{i+1}$ is the minimum value of $t_i(p_i) + cost_{i+1}(p_{i+1})$. Therefore, solution $\alpha_i(q_i)$ is better than $\alpha_i(p_i)$.□.

Lemma 2 corresponds to line 18 in Algorithm 1 where all inferior solutions are checked and skipped in the for loop. It implies that $t_i(p_i) < t_i(q_i)$ when $p_i > q_i$. Since the inferior solutions are removed from the set $L_i$, we can assume there is no inferior solution in the follow-up analysis.

If $p_{i-1}$ introduces overlaps between cell $c_{i-1}$ and $c_i$, the cost is assigned to infinity. After skipping all inferior solutions, one should also note that in line 9, the condition $cost < t_i(p_i)$ is always satisfied when no overlapping occurs. The reason lies in that $t_{i-1}(p_{i-1})$ is decreasing w.r.t $p_{i-1}$ according to Lemma 2 and $cost_i(p_i)$ does not change in the for loop from line 7–16. So the *else* condition in line 13 only happens when $p_{i-1}$ results in overlaps, and we can break the loop under such a condition.

**Lemma 3.** *Let $p_{i-1}^*$ be the optimal position of cell $c_{i-1}$ when cell $c_i$ is placed at $p_i$, and $q_{i-1}^*$ be the optimal position of cell $c_{i-1}$ when cell $c_i$ is placed at $q_i$. If $q_i \geq p_i$, then $q_{i-1}^* \geq p_{i-1}^*$.*

**Proof.** For a legal position $p_i$ of cell $c_i$, to minimize $t_i(p_i)$, we need to find the smallest $t_{i-1}(p_{i-1})$ for all possible $p_{i-1}$, because $cost_i(p_i)$ has been determined by $p_i$. Let $P_{i-1}$ be the set of all legal values of $p_{i-1}$ and

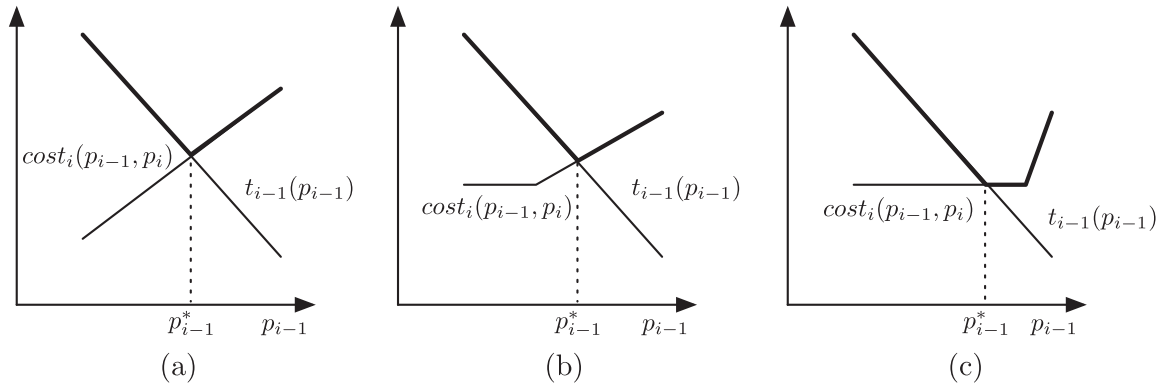


**Fig. 7.** Examples of maximization operation between a decreasing and a non-decreasing functions.

$Q_{i-1}$ denote all possible values of $q_{i-1}$. Suppose $p^*_{i-1}$ is the best position for cell $c_{i-1}$ when $c_i$ is placed at $p_i$ and $q^*_{i-1}$ is the best position for cell $c_{i-1}$ when $c_i$ is placed at $q_i$. $P_{i-1}$ and $Q_{i-1}$ should share the same left boundary $l$. Let $P^r_{i-1}$ be the right boundary of set $P_{i-1}$ and $Q^r_{i-1}$ be the right boundary of set $Q_{i-1}$. $P^r_{i-1}$ is no larger than $Q^r_{i-1}$, as $q_i \geq p_i$. In other words, we have $P_{i-1} \subseteq Q_{i-1}$. The relationship can be rewritten as,

$$P_{i-1} = \{p_{i-1} | \, l \leq p_{i-1} \leq P^r_{i-1}, p_{i-1} \in Z\}, \; Q_{i-1} = \{q_{i-1} | \, l \leq q_{i-1} \leq Q^r_{i-1}, q_{i-1} \in Z\}, \; P^r_{i-1} \leq Q^r_{i-1}.$$

If $q^*_{i-1}$ lies in the range between $l$ and $P^r_{i-1}$, it must be equal to $p^*_{i-1}$; otherwise, it is equal to some value between $P^r_{i-1}$ and $Q^r_{i-1}$. Hence, $q^*_{i-1} \geq p^*_{i-1}$.□

Lemma 3 corresponds to lines 5, 7 and 12 in Algorithm 1. After computing the optimal solution for cell $c_i$ at position $p_i = q$, We can start $p_{i-1}$ from $N$ (line 7 of Algorithm 1) instead of $p^0_{i-1} - M$ to find an optimal solution for a value $p_i > q$. In line 7 and 12, for easier explanation, we use $N$ as a position, but in implementation we can store the index of position in $L_{i-1}$ to $N$ such that the set $L_{i-1}$ is accessed in constant time.

The above analysis guarantees the optimality of Algorithm 1. Compared with previous $O(M^2)$ algorithms in [27,23], Algorithm 1 changes the complexity of the local search (from line 5 to line 18) to $O(M)$. Line 18 also takes $O(M)$ time to check all inferior solutions and remove them. The runtime complexity of Algorithm 1 is $O(nM)$.

Now we explain why the complexity of the local search (from line 5 to line 18) is $O(M)$ in Algorithm 1 with Fig. 6. The nature of the local search is to compute the best cost of cell $c_1$ to $c_i$ for each position of cell $c_i$. Let $N_i(p_i)$ denote the $N$ value in line 7 to line 16 for position $p_i$ of cell $c_i$; in other words, $N = N_i(p_i)$ when we enter line 7. We also assume there is no inferior solution for the computation of complexity for the worst case (inferior solutions are skipped anyway). Then the algorithm only repeats the part of lines 8–15 by $N_i(p_i + 1) - N_i(p_i) + 1$ times for position $p_i$ of cell $c_i$ due to the update of $N$ and early exit in line 14. Consider all the positions of cell $c_i$ from $p^0_i - M$ to $p^0_i + M$. It is necessary to repeat the part of lines 8–15 by

$$\sum_{p_i = p^0_i - M}^{p^0_i + M} N_i(p_i + 1) - N_i(p_i) + 1 = N_i(p^0_i + M + 1) - N_i(p^0_i - M) + 2M + 1, = 2M + 2M + 1, = O(M) \text{ times}, \tag{3}$$

where we introduce $N_i(p^0_i - M) = p^0_i - M$ and $N_i(p^0_i + M + 1) = p^0_i + M$ for boundary conditions.

### 3.3. Flexibility of pruning techniques

In this section, we solve Problem 2 with an extension to the single row algorithm with pruning technique. It should be noted that our pruning algorithm is flexible to any cost function $cost_i(p_i)$ as long as it only depends on $p_i$ itself. That is, it can be applied to speed-up the conventional single row detailed placement problems [20–22].

Furthermore, the cost function can also be extended from summation to maximization in line 8 of Algorithm 1. The application comes from the minimization of maximum displacement of cells when optimizing local congestion, where the cost function of each cell is adjusted from Eq. (2) to the following,

$$cost_i(p_{i-1}, p_i) = \phi \cdot MOV(p_i) + \nu \cdot SP(p_i) + \tau \cdot GAP(p_{i-1}, p_i), \tag{4a}$$

$$GAP(p_{i-1}, p_i) = \max(0, \; UB - (p_i - p_{i-1} - size_{i-1})), \tag{4b}$$

where $GAP(p_{i-1}, p_i)$ is the spacing cost between two neighboring cells and $UB$ is a user-defined upper bound for the spacing cost. The spacing cost can be more complicated such as that in [27] as long as the cost is non-increasing with the increase of spacing, while we use a simple version for illustration. We switch the symbol of cost function from $cost_i(p_i)$ to $cost_i(p_{i-1}, p_i)$ because the cost in Eq. (4) depends on positions of both cell $c_{i-1}$ and $c_i$.

In Algorithm 1, line 8 is adjusted to,

$$cost \leftarrow \max(t_{i-1}(p_{i-1}), \; cost_i(p_{i-1}, p_i)). \tag{5}$$

It should be noted that enabling minimization of the maximum cost ensures small spacing and displacement cost of the worst case, which facilitates to solve Problem 2.

With the extension of cost function, it is not hard to see that Lemma 2 still holds, which means we can still prune inferior solutions in lines 18, but the correctness of early exit in line 14 needs to be explained. Due to the pruning of inferior solutions, $t_{i-1}(p_{i-1})$ is decreasing, while $cost_i(p_{i-1}, p_i)$ is non-decreasing (increasing) w.r.t $p_{i-1}$. The maximization operation between a decreasing function and a non-decreasing function results in the fact that, given $p^*_{i-1}$ as the best position, in the region of $p_{i-1} \leq p^*_{i-1}$, $\max(t_{i-1}(p_{i-1}), cost_i(p_{i-1}, p_i))$ is non-increasing, while in the region of $p_{i-1} \geq p^*_{i-1}$, the cost is non-decreasing, shown as Fig. 7. Therefore, it does not affect the results when exit early in line 14 because $p^*_{i-1}$ has been found.

Lemma 3 needs additional proof under the new cost function. Let $p^*_{i-1}$ be the best solution for $t_i(p_i)$. Then for any $p_{i-1} < p^*_{i-1}$,

$$\max(t_{i-1}(p_{i-1}), cost_i(p_{i-1}, p_i)) = t_{i-1}(p_{i-1}) > \max(t_{i-1}(p^*_{i-1}), cost_i(p^*_{i-1}, p_i)), \tag{6}$$

where the equality comes from the discussion in Fig. 7. For any $q_i > p_i$ and $p_{i-1} < p^*_{i-1}$, we have following inequalities,

$$\max(t_{i-1}(p_{i-1}), cost_i(p_{i-1}, q_i)) \geq t_{i-1}(p_{i-1}) > \max(t_{i-1}(p^*_{i-1}), cost_i(p^*_{i-1}, p_i)) = t_i(p_i), \tag{7}$$

which indicates that current solution of $p_{i-1}$ and $q_i$ is inferior to $\alpha_i(p_i)$ with $p^*_{i-1}$ and $q_i$. Therefore, we can directly start from $p^*_{i-1}$ when searching for the best solution of $t_i(q_i)$. Although the condition of overlap is not mentioned, it can be integrated to the spacing cost, which still leads to non-decreasing cost function w.r.t $p_{i-1}$. Hence the proof holds for the new cost function with maximization operation and spacing cost.

### 3.4. Stitch aware global swap

In this step, the main objective is to optimize regions that contain cells involved in stitch errors. After the optimization of single row placement, most stitch errors have been resolved. The remaining ones usually appear in highly congested placement bins. Therefore, we only try to move cells in such bins to alleviate the congestion and meanwhile reduce wirelength.

Due to the congestion of these regions, it is difficult to resolve them with local perturbation such as reordering or sliding window. Thus global swap [28,30] is adopted where cells are allowed to move anywhere within the displacement constraints. Generalized swap not only enables swapping with cells but also white spaces, which integrates both swapping and moving strategies. The basic procedure for cell swap is iteratively repeating the following three steps: (1) Select a source cell to swap; (2) Identify optimal region for source cell; (3) Find the best cell or white space to swap with the source cell in the optimal region.

In our implementation, we set the score function for swap as follows,

$$score(c_i, c_j) = \Delta sHPWL - \lambda \cdot P_{ds} - \mu \cdot P_{ov}, \tag{8}$$

where $\Delta sHPWL$ indicates sHPWL improvement, $P_{ds}$ indicates the penalty for density increase of dangerous sites, and $P_{ov}$ is overlap penalty. Suppose cell $c_i$ is in bin $B_i$ and cell $c_j$ belongs to bin $B_j$. The area of both bins is $A_b$. We define the *density of dangerous sites* as the number of dangerous sites over total amount of available sites in a bin. If a bin has overlap with any stitch line, we account only 70% of its total sites as available. Let $D_{ds}(i)$ denote the density of dangerous sites in bin $B_i$ before swap and $D'_{ds}(i)$ denote the density of dangerous sites in bin $B_i$ after swap. Then we can define $P_{ds}$ with the following equation:

$$P_{ds} = \max(0, |D'_{ds}(i) - D'_{ds}(j)| - |D_{ds}(i) - D_{ds}(j)|) \cdot A_b. \tag{9}$$

The overlap penalty is the area difference between the source cell and target cell or white space. If the target white space is larger than the source cell, overlap penalty is zero. To achieve an equivalent numeric scale to wirelength cost, $P_{ds}$ and $P_{ov}$ are divided by site half-perimeter in the implementation. In this way, all the costs have the same unit as distance. $\lambda$ and $\mu$ are set to 100. Only swapping attempt with best positive scores is accepted.

The scoring scheme proposed in Eq. (8) aims for balancing the density of cells and dangerous sites while improving wirelength. Although the penalty from ABU density is able to handle global density distribution, local control is necessary to avoid extremely dense regions. Furthermore, it is easier for a congested region with very few dangerous sites to find a stitch-error-free solution than that with a lot of dangerous sites. Thus we introduce $P_{ds}$ as the additional penalty for such kind of regions. Since row-based legalization engine is applied, the height of bins for $P_{ds}$ is set to row height.

Overlap penalty is introduced to control the efforts during legalization. High legalization efforts will incur large displacement for some cells and thereby large wirelength degradation. Hence, after every 5000 swaps, legalization algorithm will be performed to remove overlaps. Legalization algorithm is based on single-row placement (Section 3.1) with minimum movement as an objective.

We observe that the runtime for global swap is highly related to the complexity of score function. Considering that wirelength is included in the calculation, it will be very slow to query the bounding box of large nets. Thus we develop a data structure in which pins of a net are stored as an ordered sequence according to pin positions. Cells in a row is kept in a linked-list [30] for fast cell swap and movement.

Usually a cell is connected to limited number of nets, thus its degree can be treated as constant. Using the data structures above, it only takes constant time to query the bounding box and $O(\log e)$ to update cell position in a net with $e$ pins. Since score calculation happens much more frequent than actual cell swap or movement, faster score calculation helps to reduce overall runtime. Let $k$ be the number of swapping candidates for a cell $c_i$, we can achieve $O(k)$ time complexity for score calculation and $O(\log e_{max})$ for cell position update if a swap or movement is accepted, where $e_{max}$ is the maximum $e$ of nets connected to cell $c_i$.

## 4. Experimental results

Our algorithms were implemented in C++ and tested on a 3.40 GHz Linux machine with 32 GB memory. Since traditional academic placement benchmark suites has no intra-cell wire information, we integrated the NanGate 15 nm standard cell library [31] into ICCAD 2014 placement benchmarks [19]. ICCAD 2014 placement contest defines two maximum displacement values for each benchmark, and we choose the smaller ones for less perturbation to the original placements. We applied a state-of-the-art detailed placer, RippleDp [32], to generate the initial placement solutions. We scaled the bin dimensions for ABU density analysis from the ICCAD 2014 benchmarks, so most generated test cases match to the number of bins in the original ones. We pre-computed dangerous sites for all standard cells in the library, which was served as input to our placer. We set the stripe width of each single beam to 50 μm.

The metrics of the new benchmarks are shown in Table 2, where columns "#cells" and "#nets" list the total cell number and net number, respectively. Besides, columns "#blk", "$d_t$" and "Disp." represent the blockage (fixed macro) number, the target density, and the maximum displacement in *um*. Target density $d_t$ is necessary for computing ABU penalty. Column "Util." denotes the area utilizations of benchmarks. Note that test cases `mgc_edit_dist`, `mgc_matrix_mult` and `netcard` contain mixed-sized cells.

Table 3 lists the performance of our placer at different optimization stages. The initial placement solutions (column "**Init.**") are generated by a traditional detailed placer, RippleDp [32], which aims at minimizing wirelength. As the state-of-the-art detailed placer, RippleDp can produce very high quality placement solutions in terms of both HPWL and sHPWL. Here we set displacement constraint to be a very large number so that RippleDp can produce converged results. Column "**SR**" stands for single row placement, while column "**Full Flow**" denotes the whole flow combining global swap and single row placement. To evaluate the effectiveness of our algorithms, following metrics are introduced. **HPWL** stands for half perimeter wirelength which is used as a metric for wirelength. **ST#** represents the number of cells that contains stitch errors. It is measured by how many dangerous sites are covered by the beam boundaries. Placement solutions with high congestion are not desired, so we introduce **sHPWL** as discussed in Section 2. When measuring **Runtime**, which is the CPU run time in seconds, single thread is applied for consistency of results.

From Table 3 we can see that, with certain displacement constraints, the proposed single row placement can achieve very good efficacy in stitch error cancellation. That is, 99.9% of the initial stitch errors are removed. Meanwhile, an average of 0.19% HPWL improvement and slight sHPWL increase are observed. However, for some corner cases, such as `leon2` and `netcard`, the single row placement is not powerful enough due to the movement constraints from blockages

**Table 2**
Benchmarks for Stitch Aware Placement.

| Design | #cells | #nets | #blk | Util. | $d_t$ | Disp. |
|---|---|---|---|---|---|---|
| `vga_lcd` | 165 K | 165 K | 0 | 68.94% | 70% | 10 |
| `b19` | 219 K | 219 K | 0 | 44.85% | 70% | 20 |
| `leon3mp` | 649 K | 649 K | 0 | 72.02% | 75% | 30 |
| `leon2` | 794 K | 795 K | 0 | 84.19% | 90% | 40 |
| `mgc_edit_dist` (`med`) | 131 K | 133 K | 13 | 67.26% | 70% | 30 |
| `mgc_matrix_mult` (`mmm`) | 155 K | 159 K | 16 | 59.31% | 65% | 30 |
| `netcard` | 959 K | 961 K | 12 | 66.29% | 70% | 50 |

**Table 3**
Result comparison among different approaches.

| Design | Init. | | | SR | | | | Full Flow [33] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | HPWL ($\times 10^6$) | sHPWL ($\times 10^6$) | ST # | ΔHPWL (%) | ΔsHPWL (%) | ST # | Runtime (s) | ΔHPWL (%) | ΔsHPWL (%) | ST # | Runtime (s) |
| vga_lcd | 1.42 | 1.87 | 1266 | −0.28 | +0.36 | 0 | 7.70 | −0.28 | +0.36 | 0 | 7.74 |
| b19 | 0.97 | 1.14 | 1435 | −0.25 | −0.00 | 0 | 10.54 | −0.25 | −0.00 | 0 | 10.74 |
| leon3mp | 5.34 | 6.84 | 6474 | −0.32 | −0.14 | 0 | 33.36 | −0.32 | −0.14 | 0 | 33.59 |
| leon2 | 13.09 | 14.49 | 8172 | −0.09 | +0.12 | 1 | 42.24 | −0.10 | +0.11 | 0 | 49.90 |
| med | 1.52 | 1.88 | 864 | −0.14 | +0.18 | 0 | 6.05 | −0.14 | +0.18 | 0 | 6.11 |
| mmm | 0.91 | 1.13 | 1117 | −0.17 | −0.13 | 0 | 7.22 | −0.17 | −0.13 | 0 | 7.28 |
| netcard | 14.57 | 20.19 | 7789 | −0.11 | +0.06 | 21 | 44.68 | −0.10 | +0.08 | 0 | 50.02 |
| avg. | 5.40 | 6.79 | 3873 | −0.19 | +0.06 | 3 | 21.68 | −0.19 | +0.07 | 0 | 23.62 |
| ratio | – | – | **1** | – | – | **0.001** | 1.00 | – | – | **0** | 1.09 |

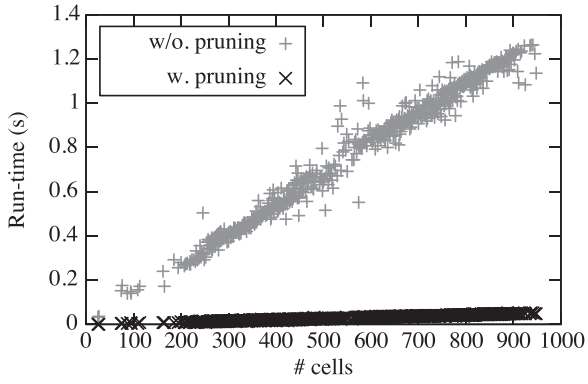


**Fig. 8.** Comparison on algorithm scalability.

or congestions. Therefore, global swap is introduced as a follow-up optimization step, and the corresponding results are shown in the last column. We can see that swapping cells between rows improves congestion in dense regions and optimize wirelength. By applying global swap together with single row algorithm, we are able to achieve zero stitch errors for all test cases. As only small number of bins are considered for global swap, the runtime overhead can be neglected. Small changes in HPWL and sHPWL also indicate that the algorithm produces little perturbation to initial placement.

It should be noted that the runtime of single row placement in Table 3 for case `netcard` is very close to that of `leon2`, while the former has much larger cell number. The reason lies in those blockages in `netcard`. That is, the runtime of single row placement is not only related to the number of cells, but also the amount of maximum displacement. Blockages have zero maximum displacement. So during the propagation of candidate solutions in the dynamic programming process, many infeasible solutions are automatically pruned. Therefore, the solution space has been significantly reduced and as a consequence, the best solution is found in shorter time.

Fig. 8 compares the runtime difference for variant amounts of cells in a row between whether applying pruning techniques or not. The data is directly collected from benchmarks in Table 2 and the runtime values of rows with the same number of cells are averaged. We can see the runtime grows linearly with the problem size and the difference in the slopes shows that pruning techniques effectively drop runtime. We compare the solutions from whether the pruning techniques are enabled or not and average the runtime in Fig. 8. On average, the $O(nM)$ pruning technique can provide around 30×speedup without any loss of optimality.

We also evaluate the of local congestion optimization discussed in Problem 2 of Section 3.3 as a post refinement step. Table 4 gives the result comparison between the full flow in [33] and that with our congestion refinement. Since the objective of the refinement is to increase the gap between pairs of cells while minimizing maximum displacement, we introduce a metric called “pair spacing ratio (**PSR**)” for each pair of horizontally neighboring cells to evaluate the performance,

$$\mathrm{PSR}(c_i, c_j) = \frac{size_i + size_j}{p_j + size_j - p_i}, \tag{10}$$

where $size_i$ and $size_j$ denote the width of cell $c_i$ (left) and $c_j$ (right), respectively. The lower left corners of cells $c_i$ and $c_j$ are denoted by $p_i$ and $p_j$, respectively. In other words, PSR $(c_i, c_j)$ is defined as the total width of two cells divided by their total spanning width including the spacing between them. The overall PSR cost is evaluated with the average PSR of all horizontally neighboring cell pairs. From the table,

**Table 4**
Result comparison for congestion refinement.

| Design | [33] | | | | | [33] + Congestion Refinement | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | ΔHPWL (%) | ΔsHPWL (%) | ST # | Runtime (s) | PSR (%) | ΔHPWL (%) | ΔsHPWL (%) | ST # | Runtime (s) | PSR (%) |
| vga_lcd | −0.28 | +0.36 | 0 | 7.74 | 93.71 | +0.81 | −1.25 | 0 | 8.23 | 89.84 |
| b19 | −0.25 | −0.00 | 0 | 10.74 | 88.99 | +2.27 | −0.56 | 0 | 11.37 | 84.46 |
| leon3mp | −0.32 | −0.14 | 0 | 33.59 | 93.83 | +0.88 | −0.62 | 0 | 36.00 | 90.78 |
| leon2 | −0.10 | +0.11 | 0 | 49.90 | 97.12 | +0.22 | +0.24 | 0 | 52.81 | 95.75 |
| med | −0.14 | +0.18 | 0 | 6.11 | 89.21 | +0.66 | −2.82 | 0 | 6.42 | 84.67 |
| mmm | −0.17 | −0.13 | 0 | 7.28 | 89.03 | +1.83 | −1.43 | 0 | 7.72 | 83.68 |
| netcard | −0.10 | +0.08 | 0 | 50.02 | 96.44 | +0.37 | −0.75 | 0 | 52.94 | 94.04 |
| avg. | −0.19 | +0.07 | 0 | 23.62 | 92.62 | +1.01 | −1.03 | 0 | 25.07 | 89.03 |
| ratio | – | – | 0 | 1.09 | **1.00** | – | – | 0 | 1.16 | **0.96** |

we can see that the congestion refinement is not only effective in removing local congestion, but also smoothing the density, because both PSR and sHPWL are improved by 4% and 1.1% compared with the flow in [33], while no stitch errors occur. Although there is degradation in HPWL, better density and local congestion are more important for routability and final routed wirelength, considering the improvement in sHPWL. In the refinement, we set the maximum displacement $M$ to 10 to avoid large perturbation to the layout, which also speeds up the algorithm. As a consequence, there is only 7% runtime overhead. The weight for spacing cost is set to 10 in the experiment and $UB$ is set to the width of smaller cells in the cell pairs.

## 5. Conclusion

This work develops the first placement framework considering e-beam stitch errors during detailed placement stage. A linear-time single row placement algorithm is proposed with highly-adaptable objective functions. Experimental results show its effectiveness in stitch cancellation while maintaining wirelength and congestion. With the collaboration of stitch aware post-placement optimization such as [7], better manufactorability can be achieved. In addition, our high performance pruning technique can be naturally embedded into existing physical design flow with different metrics (e.g., wirelength, routability, or congestion).

## Acknowledgment

This work is supported in part by NSF (Project CCF-1218906), SRC (Project 2414.001) and CUHK Direct Grant for Research. Thanks to William Chow and Prof. F. Y. Young from CUHK for the updated version of RippleDp [32].

## References

[1] D.Z. Pan, B. Yu, J.-R. Gao, Design for manufacturing with emerging nanolithography, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (TCAD) 32 (2013) 1453–1472.
[2] D.Z. Pan, L. Liebmann, B. Yu, X. Xu, Y. Lin, Pushing multiple patterning in sub-10 nm: are we ready?, in: ACM/IEEE Design Automation Conference (DAC), 2015, pp. 197:1–197:6.
[3] B. Yu, X. Xu, S. Roy, Y. Lin, J. Ou, D.Z. Pan, Design for manufacturability and reliability in extreme-scaling VLSI, Sci. China Inf. Sci. (2016) 1–23.
[4] B.J. Lin, Future of multiple-e-beam direct-write systems, J. Micro/Nanolithogr. MEMS MOEMS (JM3) 11 (2012) (033011–1).
[5] C. Van den Berg, G. De Boer, S. Boschker, E. Hakkennes, G. Holgate, M. Hoving, R. Jager, J. Koning, V. Kuiper, Y. Ma, et al., Scanning exposures with a MAPPER multibeam system, in: Proceedings of SPIE, volume 7970, 2011.
[6] M.A. McCord, P. Petric, U. Ummethala, A. Carroll, S. Kojima, L. Grella, S. Shriyan, C.T. Rettner, C.F. Bevis, REBL: design progress toward 16 nm half-pitch maskless projection electron beam lithography, in: Proceedings of SPIE, volume 8323, 2012.
[7] S.-Y. Fang, I.-J. Liu, Y.-W. Chang, Stitch-aware routing for multiple e-beam lithography, in: ACM/IEEE Design Automation Conference (DAC), 2013, pp. 25:1–25:6.
[8] K. Suzuki, T. Fujiwara, K. Hada, N. Hirayanagi, S. Kawata, K. Morita, K. Okamoto, T. Okino, S. Shimizu, T. Yahiro, Nikon EB stepper: its system concept and countermeasures for critical issues, in: Proceedings of SPIE, volume 3997, 2000.
[9] D. Dougherty, R. Muller, P. Maker, S. Forouhar, Stitching-error reduction in gratings by shot-shifted electron-beam lithography, J. Lightw. Technol. 19 (2001) 1527–1531.
[10] J. Albert, S. Theriault, F. Bilodeau, D. Johnson, K. Hill, P. Sixt, M. Rooks, Minimization of phase errors in long fiber bragg grating phase masks made using electron beam lithography, IEEE Photonics Technol. Lett. 8 (1996) 1334–1336.
[11] B. Yu, X. Xu, J.-R. Gao, Y. Lin, Z. Li, C. Alpert, D.Z. Pan, Methodology for standard cell compliance and detailed placement for triple patterning lithography, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (TCAD) 34 (2015) 726–739.
[12] H.-A. Chien, Y.-H. Chen, S.-Y. Han, H.-Y. Lai, T.-C. Wang, On refining row-based detailed placement for triple patterning lithography, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (TCAD) 34 (2015) 778–793.
[13] Y. Lin, B. Yu, B. Xu, D.Z. Pan, Triple patterning aware detailed placement toward zero cross-row middle-of-line conflict, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (TCAD) (2017).
[14] K. Han, A.B. Kahng, H. Lee, Scalable detailed placement legalization for complex sub-14nm constraints, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2015, pp. 867–873.
[15] G. Wu, C. Chu, Detailed placement algorithm for VLSI design with double-row height standard cells, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (TCAD) (2015).
[16] W.-K. Chow, C.-W. Pui, E.F.Y. Young, Legalization algorithm for multiple-row height standard cell design, in: ACM/IEEE Design Automation Conference (DAC), 2016, pp. 83:1–83:6.
[17] Y. Lin, B. Yu, X. Xu, J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, C.J. Alpert, D.Z. Pan, MrDP: Multiple-row detailed placement of heterogeneous-sized cells for advanced nodes, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2016a, pp. 7:1–7:8.
[18] Y. Lin, B. Yu, D. Z. Pan, Detailed placement in advanced technology nodes: a survey, in: IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), Hangzhou, China, 2016b, pp. 25–28.
[19] M.-C. Kim, J. Hu, N. Viswanathan, ICCAD-2014 CAD contest in incremental timing-driven placement and benchmark suite, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2014, pp. 361–366.
[20] U. Brenner, J. Vygen, Faster optimal single-row placement with fixed ordering, in: IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE), 2000, pp. 117–121.
[21] A.B. Kahng, I.L. Markov, S. Reda, On legalization of row-based placements, in: ACM Great Lakes Symposium on VLSI (GLSVLSI), 2004, pp. 214–219.
[22] P. Spindler, U. Schlichtmann, F.M. Johannes, Abacus: fast legalization of standard cell circuits with minimal movement, in: ACM International Symposium on Physical Design (ISPD), 2008, pp. 47–53.
[23] B. Yu, X. Xu, J.-R. Gao, D.Z. Pan, Methodology for standard cell compliance and detailed placement for triple patterning lithography, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2013, pp. 349–356.
[24] J. Kuang, W.-K. Chow, E.F.Y. Young, Triple patterning lithography aware optimization for standard cell based design, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2014, pp. 108–115.
[25] T. Lin, C. Chu, TPL-aware displacement-driven detailed placement refinement with coloring constraints, in: ACM International Symposium on Physical Design (ISPD), 2015, pp. 75–80.
[26] Y. Lin, B. Yu, B. Xu, D.Z. Pan, Triple patterning aware detailed placement toward zero cross-row middle-of-line conflict, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2015, pp. 396–403.
[27] T. Taghavi, C. Alpert, A. Huber, Z. Li, G.-J. Nam, S. Ramji, New placement prediction and mitigation techniques for local routing congestion, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2010, pp. 621–624.
[28] M. Pan, N. Viswanathan, C. Chu, An efficient and effective detailed placement algorithm, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2005, pp. 48–55.
[29] A.B. Kahng, P. Tucker, A. Zelikovsky, Optimization of linear placements for wirelength minimization with free sites, in: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), 1999, pp. 241–244.
[30] S. Popovych, H.-H. Lai, C.-M. Wang, Y.-L. Li, W.-H. Liu, T.-C. Wang, Density-aware detailed placement with instant legalization, in: ACM/IEEE Design Automation Conference (DAC), 2014, pp. 122:1–122:6.
[31] NanGate FreePDK15 Open Cell Library, ⟨http://www.nangate.com/?Page_id=2328⟩, 2015.
[32] W.-K. Chow, J. Kuang, X. He, W. Cai, E.F.Y. Young, Cell density-driven detailed placement with displacement constraint, in: ACM International Symposium on Physical Design (ISPD), 2014, pp. 3–10.
[33] Y. Lin, B. Yu, Y. Zou, Z. Li, C.J. Alpert, D.Z. Pan, Stitch aware detailed placement for multiple e-beam lithography, in: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), 2016, pp. 186–191.