A Heuristic Standard Cell Placement Algorithm Using Constrained Multistage Graph Model

HWAN GUE CHO AND C. M. KYUNG, MEMBER, IEEE

Abstract—We present a fast heuristic algorithm for the constructive placement of standard cells based on the so-called constrained multistage graph (CMSG) model which has a linear time complexity in the number of cells. The first step of this algorithm performs the row assignment of each cell by converting the circuit connectivity into the CMSG, where each stage of the CMSG corresponds to a cell-row in the final layout. In the second step called line sweep method, the position of each cell within the row is determined one by one so that the local channel density is minimized. Experimental results on benchmark circuits have shown that the proposed algorithm yields very competitive results in terms of the number of feedthrough cells and channel density, which was verified by the comparison of the proposed algorithm with the simulated annealing (SA) and other iterative methods such as pairwise interchange, or generalized force-directed relaxation. A theoretical time complexity analysis was performed to show that the time complexity of the proposed algorithm is $O(n)$, where $n$ denotes the number of cells, which was substantiated with the experimental results.

I. INTRODUCTION

UNTIL NOW, an enormous amount of effort has been made toward devising various placement algorithms for the standard cell layout and building block layout [1]–[3]. Many recent papers [4]–[6] report on diverse applications of simulated annealing (SA) principle to various VLSI layout problems. However, SA is a very time-consuming process and much effort is being spent to implement it on parallel computing architectures [6]–[10]. On the other hand, it was reported in the so-called simulated sintering algorithm [4], that good initial placement significantly reduces the overall computation time. Therefore, with all the powerful computers and time-consuming iterative approaches which ultimately guarantee optimal results with a sufficient number of iterations, it is still desirable to have a fast algorithm with near-optimal performance.

Such a shortcut solution approach is valuable either as an initial placement for more sophisticated iterative improvement procedures or as a final layout solution for medium performance applications. In this paper, we propose an extremely fast heuristic algorithm for the placement problems in standard cell layout with a very competitive performance obtained by exploiting the special property of standard cell layout that each cell is to be placed in one of multiple rows and the cost of connecting cells in the same row or in the neighboring rows should be relatively cheap.

Placement algorithms for standard cells can be basically categorized into two types, i.e., constructive placement and iterative improvement [12]. In most of the practical implementations, a constructive placement algorithm is used for the initial placement and the result is refined later using various iterative improvement algorithms. For the initial placement, there are several strategies such as graph clustering [14], graph partitioning based on min-cut scheme [16], [17], and linear ordering/folding [18], [19]. Since the goal of initial placement is to obtain a moderate performance placement in a small amount of CPU time, it is generally desirable that the algorithm be simple and fast. Min-cut strategy for circuit partitioning generally yields good and stable placement results by employing the concept of terminal propagation [16]. However, the terminal propagation method requires an overhead for computing the rectilinear Steiner tree which is an NP-complete problem, although some heuristics exist for 2- and 3-point Steiner tree problems [24].

On the other hand, most of the constructive algorithms consist of three parts: seed selection, circuit clustering, and ordering of the clustered circuits in rows or columns [12], [13], [18], [19], where the clustering and ordering are done independent of each other, so that further iterations are generally required for reasonably good results.

It is intuitively clear that, to preserve maximally the topology of the inter-cluster connection pattern during the placement procedure, it is desirable to consider the clustering procedure and ordering procedure simultaneously.

In this paper, we propose a fast, noniterative, $O(n)$-time algorithm for standard cell placement, which results in a very competitive layout, and is significantly faster than such time-consuming iterative improvement approaches as pairwise cell exchange, force-directed relaxation methods, and SA [11], [14]. In the first step of our procedure, explained in Section II, each cell is first assigned its row based on the so-called constrained multistage graph (CMSG) which is a graph model representing the standard cell layout. The specific position of each cell within each row is determined in the second step called line sweep method explained in Section III, where the ob-
jective function to be minimized reflects the local channel
density at the position of the hairline cursor which sweeps
the chip plane from the left end of the row to the right.

A straightforward software implementation of the
CMG algorithm called CMSGBasic and the line sweep
algorithm called LSBasic has easily resulted in a time
complexity of $O(n^{15})$, which also has shown that this
algorithm is at least 100 times faster than other iterative
algorithms such as pairwise interchange and force-di-
rected relaxation, while achieving comparable perfor-
mance. In Section IV, we showed the final version of the
CMG algorithm, called CMSGLinear coupled with
LSLinear whose time complexity was proven to be $O(n)$
through both theoretical analysis based on several practi-
cal assumptions and experimental results. The CPU time
and the performance of the placement result, in terms of
the channel density and the number of feedthrough cells,
of the proposed algorithm were compared with those of
typical constructive placement [13] using clustering and
ordering, SA, pairwise interchange, and generalized
force-directed relaxation techniques.

II. CONSTRAINED MULTI-STAGE GRAPH (CMG)
FORMATION

A. Historical Background

In the channel routing phase of the standard cell layout,
it is obvious that the routing cost can be reduced if i) the
cells belonging to a signal net are located in rows such
that the number of feedthroughs is minimal, and/or ii) the
cell positions within each row is adjusted such that the
channel density is minimal. In this paper, two different
algorithms are described to meet each of these two crite-
ria. This is, the CMG model for row assignment ex-
plained in this section is to satisfy the condition i) while
the line sweep method to be explained in Section III is to
satisfy the condition ii).

In this section, we describe an algorithm called con-
strained multistage graph (CMG) formation for the row
assignment in standard cell layout. In previous standard
cell placement algorithms [3], [12], [13], assignment of
cells to each row is performed in a two-step sequence,
the cell formation and assignment of clusters to rows.
These steps occur independently of each other in the pre-
vious work, i.e., clusters are formed such that closely
connected cells are encouraged to reside in a cluster while
the inter-cluster (interrow) connections are suppressed.
In standard cell layout, however, it is not generally desirable
to let each cluster formed by min-cut-like procedures be-
come each row, since the inter-row connections as well
as the intra-row connections have to be fully utilized to
reduce the overall routing length. The only thing to be
avoided in the row assignment is excessive generation of
feedthrough cells. Also the $K$-way graph partition algo-


Fig. 1. (a) A graph and (b) its corresponding multistage graph.

inequality constraint holds for every stage, i.e.,

$$\left| W_{\text{avg}} - \sum_{v \in V_i} w(v) \right| \leq \frac{1}{2} \max_{v \in V} \{ w(v) \}, \quad \text{for all } i$$

where, $W_{\text{avg}}$ denoting average row width is defined as

$$W_{\text{avg}} \triangleq \frac{\text{sum of width of all cells}}{\text{the number of rows}} = \frac{\sum_{v \in V_i} w(v)}{R}$$

and $V_i$, denoting the total vertex set, is defined as

$$V = \bigcup_{i=1}^{R} V_i, \quad (V_i \text{ is the set of vertices in the } i^{th} \text{ stage}).$$

In constructing a CMSG having a constant number of stages from a given circuit, we insert a dummy vertex corresponding to each feedthrough cell having the width of (1 unit = 10 μm corresponds to the wiring pitch, i.e., polysilicon wire width plus the inter-wire spacing) in the actual layout into the set $V_i$ to maintain the property of the multi-stage graph, whenever edges crossing over the $j^{th}$ stage exist. One objective in the procedure of transforming a given circuit graph into the CMSG is to minimize the total number of feedthrough cells to be inserted to maintain the property of multi-stage graph. However, the problem known as optimal linear arrangement problem for obtaining the linear ordering of $n$ cells onto $n$ stages such that the number of feedthroughs is minimal has been proven to be NP-complete [24]. The CMSG problem which is to find the arrangement of $n$ cells onto $R$ rows ($R \leq n$) such that the number of feedthroughs is minimized is also NP-complete, since it is a generalization of the optimal linear arrangement problem, and therefore, we have resorted to finding the heuristic algorithm proposed in this paper.

C. Seed Net Selection

The process of forming a CMSG from the circuit graph begins by forming two base rows in the middle of the chip which are the stage $[R/2]$ and stage $[R/2] + 1$, where $R$ is the total number of stages specified, as shown in Fig. 2(a) and (b). Since the selection of a seed cell, at the start of clustering, is very sensitive to the quality of the final layout and there is no stable guideline for the seed cell selection, we chose to start with seed nets instead of seed cell which was used in our earlier work, called CMSG linear1 having the time complexity of approximately $O(n^{1.5})$ [21]. Basically, there are two major advantages in employing the seed net instead of seed cell for forming the base row(s). Firstly, the number of feedthrough cells was significantly reduced by forming the CMSG from the two base rows in the middle and expanding both upward and downward, compared to starting from one base row at the bottom and expanding upward as in the earlier seed-cell approach [21]. Secondly, quality of the resultant placement, in terms of the number of feedthrough cells and channel density, was much less sensitive to the selection of seed nets than the selection of a seed cell. In order to select a set of seed nets, the whole circuit is simply partitioned into two parts using min-cut algorithm [20] such that the sum of weights of cells in each part is nearly equal to each other. The set of seed nets is selected as the set of cut-nets in the bipartitioned circuit, where the cut-net denotes a net having modules on both parts.

D. Row Assignment Using CMSG Model

After the set of seed nets is chosen, the assignment of cells onto the two base rows, i.e., stage $[R/2]$ and $[R/2] + 1$ has to be performed as shown in Fig. 2(b). For simplicity, we consider only the upper half, i.e., from stage $[R/2] + 1$ to stage $R$ in Fig. 2(b), since the lower half group can be similarly treated. To describe the row assignment process, we divide the whole cells into two groups. One group called "staged" group consists of cells which were assigned their respective rows, and the other group called "unstaged" group consists of cells which were not yet assigned their row. After the cell assignment in the base row is completed, the assignment of remaining cells in the subsequent rows is performed. For the purpose of convenience in explaining the procedure of forming CMSG, we assume that all the stages from the base to the $i^{th}$ stage are fully occupied with cells and we will describe how to construct the $(i + 1)^{th}$ stage from there (cell assignment onto the base row at the beginning can be done using the similar procedure as will be mentioned).

Let $\text{Adj\_row}(i)$ be the set of unstaged cells which is connected to one or more cells in the $i^{th}$ stage. $\text{Adj\_row}(i)$ then becomes a pool of cells which are can-
didates to be positioned in the \((i + 1)\)th stage. If the sum of the widths of cells in \(\text{Adj-row}(i)\) is greater than \(W_{\text{avg}}\), then some cells in \(\text{Adj-row}(i)\) are to be removed. When we remove a cell from \(\text{Adj-row}(i)\), the cell whose removal causes the generation of minimal number of feed-through cells is removed first. We define a “discarding” function, \(F_d(x)\) to select a cell to be removed from \(\text{Adj-row}(i)\):

\[
F_d(x) = |S_1(x)| - |S_2(x)|
\]

where \(S_1(x)\) is the set of nets where each net has both cell \(x\) and some cell(s) in the \(i\)th stage, and \(S_2(x)\) is a subset of \(S_1(x)\), such that for each net in \(S_2(x)\) at least one cell other than \(x\) in \(\text{Adj-row}(i)\) connects to that net. It is then obvious that \(F_d(x)\) denotes the number of feedthrough cells to be generated in the \((i + 1)\)th stage if the cell \(x\) is removed from \(\text{Adj-row}(i)\). This removal process continues until all the remaining cells in \(\text{Adj-row}(i)\) can fit in the \((i + 1)\)th row. If tie occurs in the \(F_d(x)\)-value among more than two cells, we simply choose the cell of largest width. In an example shown in Fig. 3(a), the cells staged in the \(i\)th stage and the cells in \(\text{Adj-row}(i)\) are illustrated. Based on this removal strategy, we choose to remove the cell \(a\) and \(b\) since the removal of cell \(a\) or \(b\) does not incur the generation of feed-through cells, i.e., \(F_d(a) = 2 - 2 = 0\) (\(\because S_1(a) = S_2(a) = \{N_5, N_7\}\), \(F_d(b) = 1 - 1 = 0\) (\(\because S_1(b) = S_2(b) = \{N_9\}\)), while removing cell \(c\), creates a feedthrough cell in the \((i + 1)\)th stage i.e., \(F_d(c) = 2 - 1 = 1\), as shown in Fig. 3(b) (\(\because S_1(c) = \{N_6, N_8\}\), \(S_2(c) = \{N_9\}\)).

On the other hand, if the sum of the widths of the cells in \(\text{Adj-row}(i)\) is less than \(W_{\text{avg}}\), additional cells are to be recruited into the \(\text{Adj-row}(i)\) to fill in the empty slots in \(\text{Adj-row}(i)\). We select a cell from the unstaged group which maximizes the following objective function called \(F_r(x)\) “recruiting function” and insert the selected cell into \(\text{Adj-row}(i)\):

\[
F_r(x) = N_1(x) - N_2(x)
\]

where \(x\) is a candidate cell among the unstaged group. \(N_1(x)\) denotes the number of signal nets to which both cell \(x\) and at least one cell in the staged group belong, and \(N_2(x)\) denotes the number of signal nets to which both cell \(x\) and at least one cell in the unstaged group belong. The assignment of cells to the base row(s) can be similarly handled by replacing the \(\text{Adj-row}(i)\) in the above with the set of cells belonging to the seed net and following the same procedure. The overall procedure explained so far is called CMSG_basic algorithm and is shown below in pseudo code.

**Algorithm CMSG_basic**

**Constrained Multi-Stage Graph Algorithm for Row Assignment**

**Input**: a graph denoting the electrical circuit

**Output**: a constrained multistage graph (row assignment result)

1. Compute the average row width, \(W_{\text{avg}}\) from the number of rows, \(R\).
2. Choose seed nets through min-cut partition.
3. Make the base rows i.e., row \([R/2]\) and \([R/2] + 1\) and let \(i = 1\).
4. Get \(\text{Adj-row}(i)\).
5. If the total width of \(\text{Adj-row}(i)\) > \(W_{\text{avg}}\) then while (the total width of \(\text{Adj-row}(i)\) > \(W_{\text{avg}}\)) do begin delete a cell, \(c\) from \(\text{Adj-row}(i)\) minimizing the discarding function \(F_d(c)\);
if the deletion generates feedthrough cells
then insert the feed-through cells into
$\text{Adj}_\text{row}(i)$;
end;
else
while (the total width of $\text{Adj}_\text{row}(i) < W_{avg}$) do
choose a cell, $c$ out of unplaced group maximizing the recruiting function $F_i(c)$ and
insert it into $\text{Adj}_\text{row}(i)$;
end
(6) Assign the cells in $\text{Adj}_\text{row}(i)$ to the $(i + 1)$th stage.
(7) if the last row is constructed, stop here.
else let $i := i + 1$ and go to (4).

Algorithm 1 : Pseudo code of CMSG basic, an algorithm for the row assignment through converting the given circuit graph into the constrained multi-stage graph (CMSG).

III. LINE SWEEP METHOD FOR INTRA-ROW CELL PLACEMENT

After each cell is assigned its row to be placed in, the second step called the line sweep method is responsible for determining the exact physical location of each cell within the row. Initially, we calculate the left end and right end position of each row within the chip assuming that all rows are centered symmetrically at a position as shown in Fig. 4, where $B_i$ and $E_i$ denote the left and right end positions of the $i$th row, respectively.

To describe the line sweep method for the cell positioning within a row, we define the $i$th point $P_i$ as the $x$-coordinate of the vertical interface line which is the boundary between the placed cells and the unplaced cells in the $i$th row as shown by shaded cells and unshaded cells, respectively in Fig. 5(a). Fig. 5(b) is a snapshot taken during the cell positioning process using line sweep method, where placed cells are those which already found their final positions within the row, while unplaced cells are those still remaining in the waiting pool of the relevant row in the CMSG. Current position of the sweep line shown as a dotted vertical line at $x = x_i$ is hopping rightward from its current $x$-position ($P_2$ in Fig. 5(a)) to its next position determined as the nearest $P_i$ from $x = x_i$. That is, $P_2$ becomes the next position of the sweep line since $x(P_3) = 9$ is the smallest among $x(P_i)$’s. Suppose we have placed several cells in multiple rows as in Fig. 5(b); we then find the $m$th row which satisfies $P_m = \text{min}\{P_i\}$. The next cell to be placed has to be chosen from the group of unplaced cells in the $m$th row. This is shown in Fig. 5(b) where the placed cells are shown in their respective rows, while the unplaced cells are still in the CMSG in the right side. In Fig. 5(b), a cell $A$ (denoted as shaded) is selected from the stage 3 of CMSG, which is joining the row 3, when the sweep line is at $x = x_i$. An objective function $F_p(x)$ called placing function, which computes the number of nets connected to the cell that cross over the current sweep line, $P_m$, is used for selecting the cell $x$ to be placed as defined below:

$$F_p(x) = |N_c(x) \cap N_d(x)|$$

where $N_c(x)$ is the set of nets attached to the cell $x$, and $N_d(x)$ is the set of nets crossing over the current sweep line. In Fig. 5(a), $P_4$ denotes the $x$-coordinate value of the vertical cells, in the 4th row. Sweep line is seen as a dotted vertical line at $x = x_s$ to its right side. In Fig. 5(b), a cell $A$ (denoted as crosshatched) which was just picked up among cells in the second stage of the CMSG to be placed in the second row. The next position for sweep line is at $P_1$ and the cell (shaded) having maximal $F_p$-value, i.e., maximal number of nets crossing over $x = X_s$ is being selected from the third stage of CMSG and placed in the third row.
The cell which maximizes \( F_p(x) \) is selected and this process continues until all the cells are placed within the corresponding row. If tie occurs in \( F_p(x) \)-value among many cells, the cell of the smallest width is chosen first. Therefore, the feedthrough cells are usually given the highest priority by this tie-breaking strategy, since its width is usually the smallest. The intra-row placement procedure called LS_basic is briefly shown as a pseudo code in the Algorithm 2 shown below.

[ Algorithm LS_basic ]

Line Sweep Algorithm for Intra-Row Cell Placement
Input : CMSG
Output : final standard cell placement layout.

(1) Compute \( B_i \) and \( E_i \) for all rows.
(2) Let \( P_i := B_i \) for all rows.
(3) Find the \( m \)th row such that \( P_m = \min \{ P_i \} \).
(4) Choose a cell \( c \) among the unplaced cells in \( m \)th row maximizing the placing function, \( F_p(c) \).
(5) Advance the position of \( P_m \) such that \( P_m := P_m + ( \text{the width of the cell } c ) \).
(6) If there are still unplaced cells, then go to step (3).
Otherwise stop here.

Algorithm 2 : Pseudo code of the procedure called LS_basic, an algorithm for converting the CMSG into the intra-row cell placement through line sweeping.

IV. ALGORITHM IMPLEMENTATION AND TIME COMPLEXITY ANALYSIS

In this section, we will describe the details of data structures and procedures in implementing the aforementioned Algorithm 1 and Algorithm 2 such that the overall time complexity is \( O(n) \), i.e., proportional to the number of cells in the circuit.

We will also analyze the computational complexity of the algorithms proposed. In the following analysis, we denote the number of standard cells as \( n \). There are many practical assumptions made in our analysis as follows:

Assumption 1) The number of rows in a chip, \( p \) is proportional to \( \sqrt{n} \).

Assumption 2) The number of cells in a row, \( \delta \) is proportional to \( \sqrt{n} \).

Assumption 3) The maximum number of signal nets per cell, \( \alpha \) is a constant.

Assumption 4) The maximum number of cells per signal net (except power, ground, and clocks), \( \beta \) is a constant.

Assumption 5) The maximal number of signal nets in a channel crossed by a vertical sweep line, \( \phi \) is assumed to be constant, although it was empirically shown to be a very slowly increasing function of \( n \).

Phase-1: Constrained Multi-Stage Graph Construction
Algorithm 3 called CMSG_linear2 is a refinement of the Algorithm 1, as shown by a pseudo code below:

[ Algorithm CMSG_linear2 ]

Linear Time Constrained Multi-Stage Graph Construction Algorithm
Input : a graph denoting the electrical circuit.
Output : constrained multi-stage graph (row assignment result)

( 1 ) Compute the number of rows and average row width, \( W_{avg} \).
( 2 ) Find the seed nets using the linear time min-cut algorithm.

\/* In the following we only consider the upper stages since the low stages are similarly handled */

Let \( i := \lfloor R/2 \rfloor + 1 \).
( 3 ) if \( i = \lfloor R/2 \rfloor + 1 \) then include all cells linked to the seed nets into \( Adj_row(i) \).
else include all cells linked to stage \( i \) into \( Adj_row(i) \).
( 4 ) if \( \text{total width of } Adj_row(i) > W_{avg} \) then
( 5 ) while( total width of \( Adj_row(i) > W_{avg} \) ) do
\/* \( \alpha \) is the maximum number of signal nets per cell, which was assumed to be constant */
( 6 ) for \( f = 0, 1, 2, \cdots, \alpha \) do
( 7 ) for each cell \( x \in Adj_row(i) \) do
\/* All cells in \( Adj_row(i) \) were bucket-sorted in their values of \( F_s(x) \), which denotes the number of feedthrough cells generated by removing cell \( x \) from \( Adj_row(i) \) */
\( E \left( F_s(x) \leq f \right) \) then
( 8 ) \( Adj_row(i) := Adj_row(i) - \{ x \} \)
( 9 ) endif
(10) endfor
(11) endfor
(12) endwhile
(13) endif
else
(14) Compute \( k \), which denotes the maximum number of signal nets of a cell connecting both at least one cell in \( Adj_row(i) \) and another cell in unstaged group.
(15) while( the total width of \( Adj_row(i) < W_{avg} \) ) do
(16) for \( f = k, k - 1, k - 2, \cdots, 1 \) do
(17) for each cell \( c \in Adj_row(i) \) do
\/* All cells in \( Adj_row(i) \) were bucket-sorted in their values of \( F_s(x) \), which denotes the number of signal nets shared between unstaged cell \( x \) and cells in \( Adj_row(i) \) */
\( F_s(x) > f \) then
(18) \( Adj_row(i) := Adj_row(i) + \{ x \} \)
(19) endif
(20) endfor
(21) endfor
(22) endfor
(23) endwhile
The core of the algorithm CMSG_linear2 is in adjusting the cell members in Adj_row(i) so that the total sum of widths of cells in Adj_row(i) is nearly equal to \( W_{avg} \), average row width, by removing some cells from Adj_row(i) (in steps (4)-(14)) or by recruiting some cells into Adj_row(i) (in steps (15)-(28)). In doing that, all cells within the Adj_row(i) were sorted in their values of \( F_d(x) \), discarding function and \( F_I(x) \), recruiting function with bucket sorting utilizing the fact that the value of \( W_{avg} \) and unstaged cells can be upper bounded, whereby the time complexity of the overall procedure was reduced to \( O(n) \), i.e., proportional to \( n \), the number of cells in the circuit.

Let us analyze the time complexity of the algorithm CMSG_linear2. Steps (1) and (2) are initializing steps. In step (2), we use the linear time min-cut algorithm \([20]\) to get seed nets and gather cells linked to the seed nets in linear time easily to form two base rows of upper stages (see Fig. 6). Steps (3)-(8) are procedures for removing cells from the set Adj_row(i), while step (15)-(25) are those for recruiting cells into Adj_row(i). It is easy to see that steps (1) and (2) can be computed in \( O(n) \). Gathering cells to be included in Adj_row(i) in step (3) can be done within \( O(a\beta\delta) \) time, since there are at most \( \delta \) cells per row and each cell in the ith stage (row) can be placed in \( O(a\beta) \) time, since there are at most \( a \) cells per row and each cell in the ith stage (row) can be placed in \( O(a\beta) \) time. On the other hand, for loop consisting of steps (6)-(12) is repeated as many as \( a \) times, where \( a \) is the maximum number of signal nets per cell which was assumed to be a constant. Therefore, the total time complexity between step (5) and step (13) is \( O(k\alpha^2\beta^2\delta) \).

Since there are at most \( a\beta\delta \) cells in Adj_row(i) and \( O(a\beta) \) unstaged cells are connected to at least one cell in Adj_row(i), computing the maximum connecting value, \( k \), between cells in Adj_row(i) and unstaged cells can be done by \( O(a\alpha^2\beta^2\delta) \) comparisons. So the time complexity of step (16) is \( O(a\alpha^2\beta^2\delta) \). The loop of steps (20)-(24) requires cell recruiting tests as many as \( a\beta \) times for each cells in Adj_row(i) which makes the time complexity of the for loop of step (19)-(25) to be \( O(\alpha^2\beta^2\delta) \). Therefore, the time complexity between step (18) and step (26) is \( O(k\alpha^2\beta^2\delta) \), which is \( O(\alpha^2\beta^2\delta) \), since \( k \leq \alpha \) and overall time complexity between step (4) and step (28) can be performed in \( O(\alpha^2\beta^2 \delta) \), \( O(\delta) = O(\sqrt{n}) \). Since the loop between steps (4) and (28) should be iterated exactly \( \rho - 1 = O(\sqrt{n}) \) times for upper stages and lower stages, the total time complexity of the CMSG_linear2 procedure is \( O(\sqrt{n}) + O(\sqrt{n}) = O(n) \).

**Phase-2: Line Sweep Method for Intra-Row Cell Placement**

Algorithm 2 called LS_basic was refined to reduce the overall time complexity down to \( O(n) \) and the resultant pseudo code is shown as Algorithm 4 called LS_linear shown below. We introduced the concept of buffered cells to reduce the amount of search in selecting a cell among unplaced cells, which is shown in Fig. 6. Steps (1) and (2) of Algorithm 4 need \( \rho = O(\sqrt{n}) \) time, where \( \rho \) is the number of rows in the chip. Using the Ulrich’s time-wheel concept used in logic simulation \([22]\), step (3) can be computed in constant time without the sorting procedure if we prepare a time-wheel with the number of slots as large as the width of the largest cell divided by the basic length units. Since Buffer(m) has at most \( O(\phi\beta) \) cells, step (4) requires \( O(\phi\beta) \). Steps (6)-(8) requires \( O(a\beta) \) operations. Therefore, the time complexity of steps from (3)-(8) is \( O(\phi\beta + a\beta) \) = constant. Since the steps from (3)-(8) have to be repeated as many as \( n \) times, which is the number of updating \( P_i \)'s in step (3), the whole time complexity is \( O(n) \).

*Algorithm LS_linear*

**Linear-Time Line Sweep Algorithm for Intra-Row Cell Placement**

**Input**: CMSG

**Output**: final standard cell placement.

(1) Compute \( B_i \) and \( E_i \) for all rows.

(2) Let \( P_i := B_i \) for all rows.

(3) Find \( m \) such that \( P_m = \min \{ P_i \} \).

/* Buffer(m) is the set of cells connected to at least one placed cell in the mth row. */
(4) Find the cell $x$ which maximizes $F_p(x)$ among cells in $\text{Buffer}(m)$ and place it in the $m$th row.
(5) $P_m := P_m + (\text{width of the cell } x)$. */ Advance the position of $P_m$ */
(6) for each unplaced cell, $r$ linked to $x$
(7) Insert cell $r$ into its corresponding $\text{Buffer}$
(8) endfor
(9) If there are still unplaced cells, then go to step (3) else stop here.

Algorithm 4 : Pseudo-code of LS_linear, which is a linear time refinement of the LS_basic utilizing the concept of buffer.

V. EXPERIMENTAL RESULTS AND DISCUSSION

We implemented the algorithms mentioned as C-language programs running on IBM PC/AT. The procedure implemented is a combination of one of the CMSG formation algorithms (CMSG_basic, CMSG_linear1 and CMSG_linear2), and one of the line sweep algorithms (LS_basic and LS_linear) explained before. Two different sets of experiments were performed for test circuit group A and B, respectively. First, we compared the CPU time and performance of two procedures, CMSG_basic + LS_basic and CMSG_linear1 + LS_linear with two other classic procedures, i.e., one based on linear ordering and folding (LOF) plus pairwise interchange (PI), and the other based on LOF plus generalized force-directed relaxation (GFDR), using test circuit group A consisting of ALU, COM and CAL. Table I shows the characteristics of the example circuits in group A and Table II shows the result of comparison among the four different procedures (LOF + PI, LOF + GFDR, CMSG_basic + LS_basic, and CMSG_linear1 + LS_linear) in terms of CPU time, channel density, and the number of feedthroughs. From Table II we can easily see that the CPU time of the last two CMSG_based procedures has been reduced by a factor of greater than 100 compared to the other two iterative algorithms, LOF + PI and LOF + GFDR. For all these algorithms, the channel density values estimated using the greedy channel router [23] were almost the same except that the result of CAL with LOF + PI was relatively poorer than others. On the other hand, the number of feedthroughs was drastically reduced for all examples by using CMSG-based procedures.

In the second set of experiment using another test circuit group B, we compared the two CMSG-based algorithms, CMSG-linear1 and CMSG-linear2, with SA. In CMSG_linear1, the CMSG formation process starts from a seed cell and is propagated in one direction, while in the procedure called CMSG_linear2, the CMSG formation process propagates in both directions from the two base rows in the middle selected using the seed nets. Table III shows the characteristics of the benchmark circuits, where CIRCUITX is a medium-sized peripheral interface chip, DECIN is a highly combinational logic, and REGFILE is a highly regular, sequential logic circuit [25]. The result of comparison among three procedures, SA, CMSG_linear1, and CMSG_linear2 are shown in Table IV, where we can note two things.

Firstly, CMSG_linear1 + LS_linear is worse than CMSG_linear2 + LS_linear in terms of the number of feedthrough cells and the total channel density. We also observed that the CMSG_linear1 procedure was very sensitive to the selection of seed cell, while the CMSG_linear2 procedure was much more sensitive to the selection of seed nets. For the two CIRCUITX examples having 833 cells, the (CMSG_linear1 + LS_linear) pro-

<table>
<thead>
<tr>
<th>TABLE I</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statistical Data for Test Circuit Group A (Cell Height = 150 µ)</strong></td>
</tr>
<tr>
<td>Circuit (Group A)</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>ALU</td>
</tr>
<tr>
<td>COM</td>
</tr>
<tr>
<td>CAL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE II</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU Time and Performance Comparison Among LOF + PI, LOF + GFDR, CMSG_basic + LS_basic, and CMSG_linear1 + LS_linear for Test Circuit Group A.</strong></td>
</tr>
<tr>
<td>Circuit (Group A)</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>CPU time(Sec.)</td>
</tr>
<tr>
<td>LOF + GFDR</td>
</tr>
<tr>
<td>CMSG_basic + LS_basic</td>
</tr>
<tr>
<td>CMSG_linear1 + LS_linear</td>
</tr>
<tr>
<td>Total number</td>
</tr>
<tr>
<td>Channel density</td>
</tr>
<tr>
<td>CMSG_basic + LS_basic</td>
</tr>
<tr>
<td>CMSG_linear1 + LS_linear</td>
</tr>
<tr>
<td>Number of feed-through cells</td>
</tr>
<tr>
<td>CMSG_basic + LS_basic</td>
</tr>
<tr>
<td>CMSG_linear1 + LS_linear</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE III</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statistical Data for Test Circuit Group B (Cell Height = 150 µ)</strong></td>
</tr>
<tr>
<td>Circuit (Group B)</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>DECIN</td>
</tr>
<tr>
<td>REGFILE</td>
</tr>
<tr>
<td>CIRCUITX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE IV</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU Time and Performance Comparison Among S.A., CMSG_linear1 + LS_linear, CMSG_linear2 + LS_linear, and for Test Circuit Group B.</strong></td>
</tr>
<tr>
<td>Circuit (Group B)</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>CPU time(Sec.)</td>
</tr>
<tr>
<td>S.A.</td>
</tr>
<tr>
<td>IBM PC/AT</td>
</tr>
<tr>
<td>CMSG_linear2 + LS_linear</td>
</tr>
<tr>
<td>Total</td>
</tr>
<tr>
<td>Channel density</td>
</tr>
<tr>
<td>CMSG_linear1 + LS_linear</td>
</tr>
<tr>
<td>CMSG_linear2 + LS_linear</td>
</tr>
<tr>
<td>Number of feed-through cells</td>
</tr>
<tr>
<td>CMSG_linear1 + LS_linear</td>
</tr>
<tr>
<td>CMSG_linear2 + LS_linear</td>
</tr>
</tbody>
</table>

"NA" mean: that data were not available due to either excessive CPU time or storage required; an asterisk (*) denotes that the storage capacity for feedthrough cells, which was set to 900, was exceeded. The numbers, 217 and 223 with the double asterisk (**) denote the total channel density data obtained by using rowwise S.A. instead of LS_linear after CMSG_linear2 procedure.
procedure failed to complete the row assignment due to the overflow of the capacity (=900) of the buffer for the storage of feedthrough cells.

Secondly, we like to compare the (CMSG_linear2 + LS_linear) procedure to the SA. We were not able to obtain the stable SA result for the CIRCUITX examples due to excessive CPU time (about 6 h on 2 MIPS machine). For the DECIN and REGFILE examples, the CPU time of (CMSG_linear2 + LS_linear) is about two orders of magnitude smaller than that of SA, while the number of feedthrough cells was significantly larger in the SA result. Surprisingly, the total channel density obtained using (CMSG_linear2 + LS_linear) procedure was even less than that of SA, i.e., 39 versus 42, which is believed to be due to the nature of the connectivity of the REGFILE circuit having no local clusters, but the wiring pattern is bus-type, global, and extremely regular.

Fig. 7 shows the result of placement of the DECIN circuit using procedure CMSG_linear2 + LS_linear followed by the channel routing using greedy algorithm.

Finally, we will analyze the time complexity of the proposed algorithms from the experimental results. As was mentioned, procedure CMSG_linear1 + LS_linear is the result of modifying the procedure CMSG_basic + LS_basic, such that the time complexity is nearly linear in terms of the number of cells, \( n \), i.e., \( O(n) \) while CMSG_linear2 is different from CMSG_linear1 only in that CMSG_linear2 starts the CMSG formation process from the two based rows in the middle selected using the seed nets instead of the seed cell used in the CMSG_linear1. Fig. 8 shows the result of plotting CPU time versus \( n \), the number of cells in the test circuit for various sizes of circuits, i.e., from \( n = 67, 144, 270, 481, 625, 751, 895, \) and 1100. The curve "CMSG_linear1 + LS_linear" and "CMSG_linear2 + LS_linear" is CPU time on IBM PC/AT for procedure CMSG_linear1 + LS_linear and CMSG_linear2 + LS_linear, respectively, while the curve "CMSG_basic + LS_basic" is CPU time on IBM PC/AT for procedure CMSG_basic + LS_basic. Two curves are shown for the (CMSG_linear2 + LS_linear) procedure, i.e., including the generated feedthrough cell in the cell count or not. Slopes of the curves "CMSG_linear1 + LS_linear" and "CMSG_linear2 + LS_linear" substantiates our assertion that the time complexity of procedure CMSG_linear1 + LS_linear and CMSG_linear2 + LS_linear is CPU time on IBM PC/AT for procedure CMSG_basic + LS_basic.

We proposed a fast, one-pass, \( O(n) \)-time complexity algorithm called CMSG_linear2 coupled with LS_linear for the standard cell placement, which was experimentally proven to be at least 100 times faster than other nearly exhaustive iterative procedures such as simulated annealing (SA), pairwise interchange and generalized force-directed relaxation while the performance of the CMSG_based algorithm was either superior or at least comparable to those methods. We have also shown various techniques for speeding up the algorithm such as time wheel concept [22], buffered cell group, and by setting upper bounds on the value of \( F \), \( F_p \), and selecting candidate cells in terms of the discretized values of \( F \), and
$F_p$ in the bucket-sorted list. A salient feature of the proposed CMSG-based algorithm is that the row assignment, i.e., the clustering of cells in the row, is performed with in-situ consideration of the final ordering topology among the clusters, compared to other approaches [13] where the row assignment is performed after the clusters are formed. By doing so, we were able to avoid the time-consuming iteration steps, and reduce the channel density and the feedthrough cells by exploiting the two-dimensional connection pattern in the row assignment stage. Finally, the algorithm called CMSG_linear2 was shown to be very stable owing to the use of the concept of seed nets to form the base rows instead of seed cells in CMSG_linear1.

ACKNOWLEDGMENT

The first author thanks Prof. Chua for helpful discussions.

REFERENCES


Hwan Gye Cho received the B.S. degree in computer science and statistics from Seoul National University, Seoul, Korea in 1984, and the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology, in 1986. Currently, he is a teaching assistant and Ph.D. student in the Department of Computer Science, Korea Advanced Institute of Science and Technology.

His research interests include applied graph theory, design automation, design and analysis of computer algorithms, and computational geometry.

C. M. Kyung (S'76-M'81) received the B.S. degree from the Department of Electrical Engineering of Seoul National University in 1975 and the M.S. and Ph.D. degrees from the Department of Electrical Engineering of KAIST (Korea Advanced Institute of Science and Technology) in 1977 and 1981, respectively.

After graduation from KAIST, he worked at AT&T Bell Laboratories, Murray Hill, NJ, from April 1981 to January 1983 in the area of semiconductor device and process simulation. In February 1983, he joined the Department of Electrical Engineering at KAIST as an Assistant Professor, and was promoted to Associate Professor in 1986. His current research interest includes physical layout CAD algorithms for VLSI design, computer graphics, and neural nets.