An Efficient Algorithm for Some Multirow Layout Problems

Jack A. Feldman, Israel A. Wagner, and Shmuel Wimer

Abstract—**Automatic generation of standard CMOS logic cells has been studied intensively during the last decade. Some maturity has been achieved, and several commercial tools are available. The continuous progress in VLSI technology presents new challenges in developing efficient algorithms for the layout of standard CMOS logic cells and in combining them within functional macros.**

In this paper, three multirow layout problems are presented: transistor orientation, contact positioning and symbolic-to-shape translation. It is shown that these multirow problems have a common property, which we call *quantitative dependency*. Using this property, an optimization technique is presented, which is based on a penalty-delay strategy. It is proved that the penalty-delay strategy assures optimality, and that the optimal solution can be obtained in linear time.

The algorithmic approach is based on the observation that optimal layout decisions in any region within a cell or a macro depend only on quantitative measures of the decisions in other regions, rather than on their details. This suggests to depart from the traditional approach of handling the different regions separately and combining them afterward into a single unit, an approach that may degrade the quality of the final layout. Instead, the entire macro can be processed at once, taking into account the mutual quantitative dependency between distinguished regions.

I. Introduction

THE AUTOMATIC generation of standard CMOS logic cells has been addressed in many papers, and several commercial layout systems offer such generators in their toolkit. The automatic layout generation of a cell is usually divided into two steps. First, a symbolic layout is generated, and then it is translated into mask shapes. Symbolic layout is divided further into placement and routing, which are usually carried out separately, whereas translation to mask shapes is performed either by employing a compactor or by a direct shape embedding. When a direct mapping into shapes is attempted, ground rules consideration should not be deferred to the final stage. Placement and routing algorithms capable of handling *symbolic spacing rules* are essential for high-quality layout.

The optimal transistor placement problem was studied extensively. Several heuristics for general CMOS logic cells have been proposed, and for some specific circuit structures optimal algorithms exist [9]. In general, however, even when restricted to a linear array of transistors [15], the problem is NP-hard [4] and thus solved in two steps: find the linear order of P-N transistor pairs, and then find for every transistor its optimal orientation ("flip") within the array [6]. In [1] the two problems have been addressed simultaneously. For CMOS cells that consist of several transistor rows, the problem of optimal transistor flip was presented in [11] and was solved by an optimal algorithm for the specific problem.

Several papers (cf., [3], [5], [6]) have addressed the problem of transistor placement in CMOS cells in two separate steps. In the first step, a linear ordering of the transistors in P and N rows that takes into account wiring density and total wire length is sought. This step also attempts to maximize the likelihood of diffusion abutments occurring between consecutive transistors. The proper orientation of the transistors is ignored in this step. Then, a second step consists of finding the optimal orientation (flip) for each transistor, under the linear ordering imposed by the first step. The optimal orientation step aims at minimizing the extra space caused by diffusion gaps between adjacent transistors. The minimum wire length obtained in the model of [3] and [5] is not harmed by the optimal orientation step. Practically, transistor flip may increase the wiring density by at most one, while the total wire length can only be reduced.

The optimal flip problem has been studied in several papers. In [6] the problem was solved for a single row in a branch-and-bound procedure, in time complexity exponential in the number of transistors. In [11] the problem was solved by a dynamic programming approach in time complexity linear in the number of transistors, regardless of the number of rows. In [1] the dynamic programming solution for a single row was embedded in the linear ordering step of transistors.

Most routing algorithms for CMOS logic cells look for a feasible solution, but do not consider symbolic spacing rules. Symbolic spacing rules capture the technology ground rules in terms of symbolic configurations and associated penalties. Like placement, optimal routing of CMOS cells is a difficult problem and, therefore, is solved in several steps. Usually, the cell area is divided into several regions, the interconnecting nets are first assigned to these regions, and then each region is routed separately [17], [7].

Manuscript received March 5, 1992; revised January 8, 1993. This paper was recommended by Editor Margaret Marek-Sadowska.

The authors are with the IBM Israel Scientific Center, Technion City, Haifa, Israel.

IEEE Log Number 9207795.

0278-0070/93$03.00 © 1993 IEEE

The outcome of this process is a symbolic layout in which wire segments and contacts are assigned to wiring tracks and grid points. Still, a postprocess of the resulting symbolic routing that takes into account all the different regions simultaneously may yield a significant area reduction. This can be accomplished by transformations of the symbolic layout that pose better initial conditions for the shape mapping process that follows.

Such a problem was solved in [1] for the case of a single row cell, as a part of a dynamic programming routing algorithm for the entire cell. Its complexity was linear in the number of transistors, but grew exponentially with the number of transistor rows and wiring tracks. Therefore, when many rows of transistors and/or many wiring tracks within each row are being involved, breaking the problem into two steps is suggested. In the first, a simple routing algorithm (e.g., interval graph coloring) is employed. Then, a second step finds the most feasible track location for each contact.

The translation from symbolic layout to mask shapes is usually carried out by a compactor [10], [13]. A common practice for CMOS cell layout is to align transistors and wires to grid locations whose pitch is dictated by the technology ground rules and design methodology. This allows a convenient composition of smaller cells into large macros [12], [14]. Still, local displacement of contacts around the grid points are tolerated. This degree of freedom can be utilized to yield a smaller layout when the spacing rules between adjacent contacts dictate the cell size.

The preceding problems, which arise in the various phases of the cell generation, can be represented by a unified mathematical model. The optimization problem resulting from this unified model can be solved efficiently in linear time, employing a greedy-like approach based on a "penalty-delay" strategy.

The rest of the paper is organized as follows. The next section formally presents the layout problems and shows how they all are mapped into the same optimization problem. The application to optimal transistor flip is discussed first. Then, an application for local symbolic routing changes is shown, followed by an application for optimal contact displacement in the final mask layout. Section III describes an optimal algorithm to solve the problem, and proves its correctness. Section IV concludes and presents some problems for further research.

II. Three Layout Problems

In this section, three layout problems are defined that reflect different aspects of VLSI layout optimization.

A. Placement: Optimal Transistor Flip

The following problem was presented and solved in [11]: Let T be an arrangement of $m \times n$ transistors in a two-dimensional array, where

$$T_{i,j} = (S_{i,j}, D_{i,j}).$$

$S_{i,j}$ and $D_{i,j}$ are the nets connected to the source and drain diffusion terminals of the transistor, respectively. If the right net of the transistor in position (i, j) is the same as the left net of the transistor in position $(i, j + 1)$, the corresponding terminals abut. Otherwise, a space must be inserted between the two terminals, thus increasing the layout width. When a space is inserted in some row of column j, this penalty is paid throughout the entire column, since all the transistors must be aligned in both rows and columns. Consequently, one is interested in orienting the transistors in such a way that the number of space insertions is minimized. See Fig. 1 for an example. Formally, let $x_{i,j}$ be the orientation of the transistor $T_{i,j}$, defined by

$$x_{i,j} = \begin{cases} 1, & \text{if } T_{i,j} \text{ is oriented } (S_{i,j}, D_{i,j}) \\ 2, & \text{if } T_{i,y} \text{ is oriented } (D_{i,j}, S_{i,j}). \end{cases}$$

In this case, all x values are taken from the set $\{1, 2\}$. Let X be an $m \times n$ matrix of the preceding orientation variables. The actual left and right diffusions of a transistor, for a given configuration X, will be defined as

$$Left(X, i, j) = \begin{cases} S_{i,j}, & \text{if } x_{i,j} = 1 \\ D_{i,j}, & \text{if } x_{i,j} = 2 \end{cases}$$

$$Right(X, i, j) = \begin{cases} D_{i,j}, & \text{if } x_{i,j} = 1 \\ S_{i,j}, & \text{if } x_{i,j} = 2. \end{cases}$$

The penalty of two consecutive orientations is given by

$$f_{i,j}(x_{i,j-1}, x_{i,j}) = \begin{cases} 0, & \text{if } Right(X, i, j-1) = Left(X, i, j) \\ 1, & \text{otherwise.} \end{cases}$$

Therefore, the objective function that measures the total space insertions is given by

$$F(X) = \sum_{j=2}^{n} \max_{i=1}^{m} \{f_{i,j}(x_{i,j-1}, x_{i,j})\}.$$

The optimal transistor flip problem is to find X that minimizes $F(X)$. The maximum taken for each pair of consecutive columns in the preceding equations reflects the fact that once a space is inserted in one row, the penalty is paid in all the remaining rows, regardless of whether a space insertion is needed in the other rows.

B. Symbolic Layout Optimization

Another problem in layout generation occurs during symbolic-to-geometric mapping. Here, the layout is described by using symbols and their relative locations (symbolic layout). The next step is to map the symbolic layout into a set of real shapes. This process determines the absolute location of the shapes, and may result in extra spaces due to design rule constraints [16].

The two problems described subsequently deal with the issue of symbolic layout modifications during the sym-

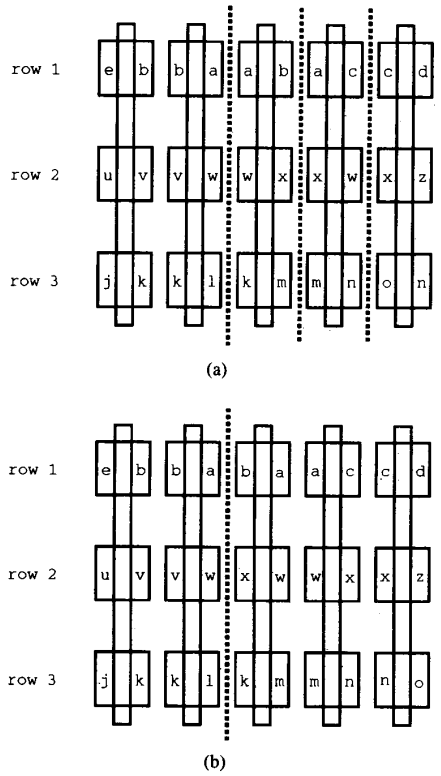


Fig. 1. Two possible solutions for a multirow transistor flip problem: (a) nonoptimal solution (three gaps); (b) optimal solution (one gap).

bolic-to-shape conversion stage. The modifications we deal with are local relocations of symbolic objects to avoid design rule violations. This results in space reduction. For some technologies, gate-contact to gate-contact spacing rules may be larger than gate-to-gate spacing rules. As a result, a uniform grid for all physical layers is too restrictive. The minimum distance between two adjacent contacts on gates forbids them to reside on two horizontally adjacent grid points. In the sequel, we discuss two ways to solve this problem.

1) Optimal Vertical Contact Relocation: One way to solve this design rule problem is to shift a contact vertically to an upper or lower vacant track, if one is available. Fig. 2(a) shows the wiring part of a symbolic layout. There, devices are aligned vertically, and this alignment must be preserved to enable vertical connections between devices. Wires and contacts can be assigned only to predefined wiring tracks within each row. We restrict our discussion to those wires whose endpoint is connected to a transistor gate by means of a metal-to-polysilicon contact. Then, if two contacts reside on the same track in two consecutive columns, shifting one of them to another track (if possible) will resolve the conflict, and thus will avoid an extra spacing. This requires an extra orthogonal piece of metal to bridge the new contact track and the original wire track, as shown in Fig. 2(b) and (d). See Fig. 2(c) and (d) for nonoptimal and optimal solutions, respectively. Note that the contact locations are bounded to feasible values only; that is, a contact cannot be located on a track that will cause a short-cut with another net [see Fig. 2(b)].

Formally, let C be the poly-to-metal contact matrix in a symbolic form:

$$c_{i,j} = \begin{cases} 1, & \text{if there is a contact in row } i \text{ and column } j. \\ 0, & \text{otherwise.} \end{cases}$$

Assume that every row i, $1 \le i \le m$, contains k wiring tracks. Let the wire segment to which the contact $c_{i,j}$ has to be connected reside on track t within row i. Let $\alpha_{i,j}$ and $\beta_{i,j}$ be the numbers of the lowest and highest vacant tracks, respectively, on which $c_{i,j}$ can be legally relocated. In other words, tracks $\alpha_{i,j}$, $\alpha_{i,j}+1, \cdots, t-1$, $t, \cdots, \beta_{i,j}-1, \beta_{i,j}$, are vacant at column j. Then, the set $s_{i,j}$ of legal positions for each contact will be defined as $s_{i,j} = \{\alpha_{i,j}, \cdots, \beta_{i,j}\}$, and the decision variables will be represented by an $m \times n$ matrix X:

$$x_{i,j} = \begin{cases} t \in s_{i,j}, & \text{if } c_{i,j} = 1. \\ 0, & \text{otherwise.} \end{cases}$$

Let f be a penalty function defined on two consecutive columns as follows:

$$f(x_{i,j-1}, x_{i,j}) = \begin{cases} 1, & \text{if } (c_{i,j-1} = c_{i,j} = 1) \text{ and } (x_{i,j-1} = x_{i,j}). \\ 0, & \text{otherwise.} \end{cases}$$

Then the objective function that measures contact-to-contact conflicts is given by

$$F(X) = \sum_{j=2}^{n} \max_{i=1}^{m} \{f(x_{i,j-1}, x_{i,j})\}.$$

The optimal vertical contact relocation problem is to minimize $F(X)$.

2) Optimal Horizontal Contact Offsetting: Another way to handle the contact adjacency problem is by moving the contacts off-grid. In some technologies, offsetting the left contact to the left and the right one to the right yields the legal distance between them. This offsetting is large enough to avoid design rule violation, and small enough to keep the contact connected to the gate below it, without moving the devices apart on the grid.

A more complex situation occurs when a sequence of horizontally adjacent contacts exists in one or more rows. Obviously, not all the conflicts can be resolved by the preceding contact offsetting, and some extra grid spacings have to be inserted. As in the former example, this area penalty is being paid in all the rows simultaneously. Therefore, the problem is to decide about the contact offsetting such that the unresolved contact adjacencies can be handled with a minimal number of column insertions. See Fig. 3 for an example.

As before, assume that every row i, $1 \le i \le m$, contains k wiring tracks, and let t ($1 \le t \le k$) denote a spe-

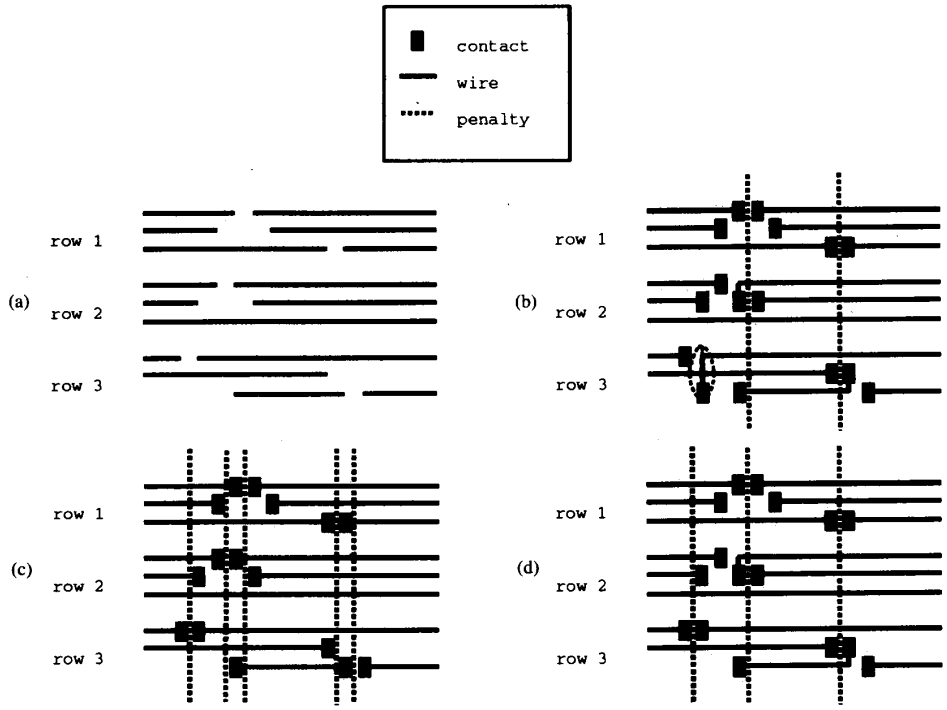


Fig. 2. Solving the multirow contact adjacency problem by vertical relocation. (a) initial routing (no contacts); (b) illegal solution; (c) nonoptimal solution (five penalties); (d) optimal solution (three penalties).

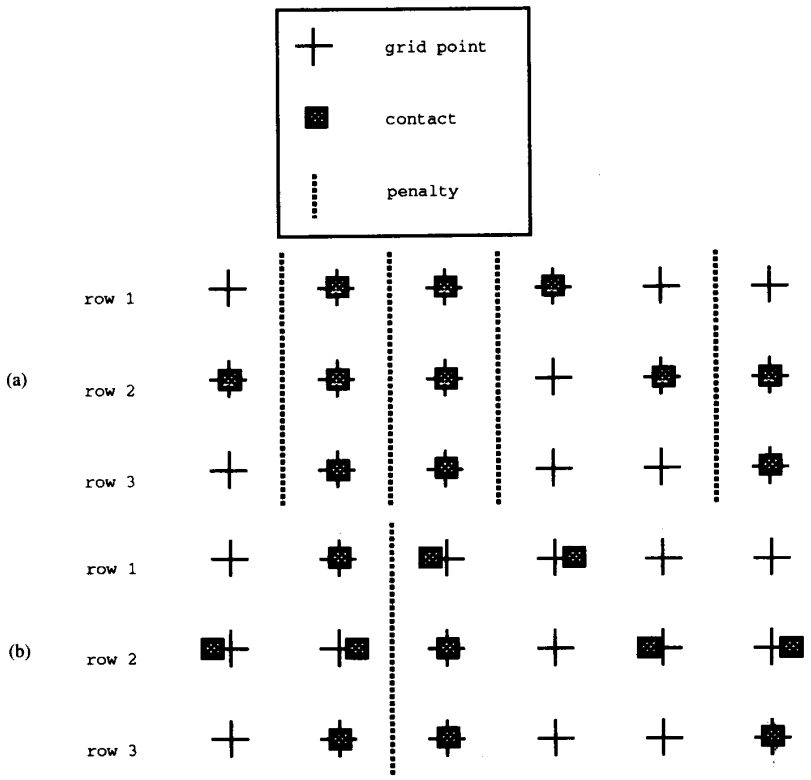


Fig. 3. Solving multirow contact adjacency problem by horizontal offsetting. (a) nonoptimal solution (four penalties); (b) optimal solution (one penalty).

cific track within the row. Let C be the poly-to-metal contact matrix in a symbolic form.

$$c_{i,j} = \begin{cases} t, & \text{there is a contact in row } i \text{ and column } j \text{ residing on track } t. \\ 0, & \text{otherwise.} \end{cases}$$

The decision variation are represented by an $m \times n$ matrix X. Let $x_{i,j}$ indicate whether a contact $c_{i,j}$ is shifted to the left, unshifted, or shifted to the right.

$$x_{i,j} = \begin{cases} 1, & \text{if } (c_{i,j} > 0) \text{ and offset to left.} \\ 2, & \text{if } (c_{i,j} > 0) \text{ and no offset.} \\ 3, & \text{if } (c_{i,j} > 0) \text{ and offset to right.} \\ 0, & \text{otherwise.} \end{cases}$$

The local cost resulting from two consecutive contacts is defined as

$$f(x_{i,j-1}, x_{i,j}) = \begin{cases} 1, & \text{if } (c_{i,j-1} = c_{i,j} > 0) \text{ and } (x_{i,j-1} \in \{2, 3\} \text{ or } x_{i,j} \in \{1, 2\}). \\ 0, & \text{otherwise.} \end{cases}$$

Then, the objective function that measures area penalty (in grid units) due to contact-to-contact conflicts is given by

$$F(X) = \sum_{j=2}^{n} \max_{i=1}^{m} \{f(x_{i,j-1}, x_{i,j})\}.$$

The optimal horizontal contact offsetting problem is to minimize $F(X)$.

III. Solving the General Multirow Problem

Although the preceding problems reflect different aspects of VLSI layout optimization, Section II showed how they are all mapped to the same optimization problem. In the following, the problem is investigated to provide the basis for an algorithm for its solution.

The idea behind the algorithm is to delay the first penalty payment to the rightmost possible column. Once a penalty is being paid, the remainder of decision variables are considered as a new, independent problem.

A. Some Definitions

Assume that X is an $m \times n$ matrix of variables $x_{i,j}$, in which each variable may take a value from a predefined finite set $s_{i,j}$, namely,

$$x_{i,j} \in s_{i,j}, \qquad 1 \le i \le m,\ 1 \le j \le n.$$

It is assumed that $s_{i,j} \subseteq \{0, \cdots, k\}$, where k is a small integer. According to the definitions in the previous section, in our three problems $k \le 4$. Let us define $\mathbf{X}$ as the set of all legal matrices X:

$$\mathbf{X} \triangleq \{X | \forall 1 \le i \le m,\ 1 \le j \le n: x_{i,j} \in s_{i,j}\}.$$

For each $1 \le i \le m$ and $2 \le j \le n$, define $f_{i,j}$ as a two-valued function, operating on two horizontally adjacent variables in X:

$$f_{i,j}(x_{i,j-1}, x_{i,j}): s_{i,j-1} \times s_{i,j} \mapsto \{0, 1\}.$$

Now consider a mathematical program, reflecting the property that once a penalty occurs in a row, it affects the whole column, regardless of what value f has in other rows.

Problem 1: Minimize:

$$F(X) = \sum_{j=2}^{n} \max_{i=1}^{m} \{f_{i,j}(x_{i,j-1}, x_{i,j})\}$$

subject to

$$X \in \mathbf{X}.$$

An optimization of this form has the property that the interdependency between the rows of X is only quantitative, that is, the contribution of a variable to the total cost depends on the values of its neighbors and on the contributions made by other variables in the same column, but not on the actual values of variables in other rows. The name *quantitative dependency* is suggested for this property.

In the sequel, an algorithm will be presented to solve this problem in time complexity $O(mnk^2)$.

One may look at the jth term of the summation in Problem 1 as a penalty that may be paid between columns $j-1$ and j, depending on the values of X in these columns.

Before solving Problem 1, some definitions are in order:

Definition 1: F^* is the minimum value of the objective function:

$$F^* \triangleq \min_{X \in \mathbf{X}} \{F(X)\}.$$

Definition 2: $F(X, j_1, j_2)$ is the contribution to $F(X)$ made by a submatrix of X, ranging from column j_1 to column j_2:

$$F(X, j_1, j_2) \triangleq \sum_{j=j_1+1}^{j_2} \max_{i=1}^{m} \{f_{i,j}(x_{i,j-1}, x_{i,j})\}.$$

Definition 3: $F^*(j_1, j_2)$ is the minimum of the contribution to $F(X)$ made by a submatrix of X, as defined in Definition 2:

$$F^*(j_1, j_2) \triangleq \min_{X \in \mathbf{X}} \{F(X, j_1, j_2)\}.$$

It follows from the above definitions that $F^* \triangleq F^*(1, m)$.

B. "Delay the Penalty as Much as You Can" Is an Optimal Strategy

The following discussion proves the optimality of delaying the penalty payment as much as possible:

Lemma 1: $F^*(p, q)$ is monotonous, that is, if $j_1 \le j_2 \le j_3 \le j_4$ (i.e., $[j_2, j_3] \subseteq [j_1, j_4]$ then:

$$F^*(j_1, j_4) \ge F^*(j_2, j_3)$$

Proof: It follows from the preceding definitions, that for every X there is.

$$F(X, j_1, j_4) = F(X, j_1, j_2) + F(X, j_2, j_3) + F(X, j_3, j_4).$$

Since $F(X)$ and hence F^* are always nonnegative, it follows that

$$F^*(j_1, j_4) \geq F^*(j_1, j_2) + F^*(j_2, j_3) + F^*(j_3, j_4).$$

Q.E.D.

Definition 4: Assume that $1 \leq j \leq n$. Considering all the possibilities for X, let $N(j)$ denote the next penalty location, which is the first column where a penalty is mandatory, starting at column j.

$$N(j) = \max_{j \leq l \leq n} \{l \mid \exists X: F(X, j, l) = 0\}$$

Note that $N(j)$ is always defined, since by definition: $F(X, j, j) = 0$. The procedure **NPL** listed in Fig. 4 computes $N(j)$ and assigns proper values to columns $j, j+1, \cdots, N(j)-1, N(j)$ of X.

Theorem 1: The following recursion holds for $F^*(j, n)$:

$$F^*(j, n) = \begin{cases} 0, & N(j) = n \\ 1 + F^*(N(j) + 1, n), & N(j) < n. \end{cases}$$

Proof: If $N(j) = n$ then, from the definition of $N(j)$ there is an X for which $F(X, j, n) = 0$, hence $F^*(j, n) = 0$. If $N(j) < n$, we show first that:

a) $F^*(j, n) \leq 1 + F^*(N(j) + 1, n)$: Since $F^*(j, n)$ is a minimum value, we only need to show that there exists some X for which $F(X, j, n) = 1 + F^*(N(j) + 1, n)$. Denote by X' a configuration for which $F(X', j, N(j)) = 0$ [such an X' exists due to the definition of $N(j)$]. In addition, denote by X'' a configuration such that $F(X'', N(j) + 1, n) = F^*(N(j) + 1, n)$. Now let us superpose X' and X'' to create X''' by taking columns $j, \cdots, N(j)$ from X' and columns $N(j) + 1, \cdots, n$ from X''. (Columns $1 \cdots j - 1$ are irrelevant from the matter.) Due to the construction of X''', the following three statements hold:

1) $F(X''', j, N(j)) = 0$
2) $F(X''', N(j), N(j) + 1) \leq 1$
3) $F(X''', N(j) + 1, n) = F^*(N(j) + 1, n)$

It follows that:

$$\begin{aligned} F^*(j, n) &\leq F(X''', j, n) \\ &= F(X''', j, N(j)) + (X''', N(j), N(j) + 1) \\ &\quad + F(X''', N(j) + 1, n) \\ &\leq 1 + F^*(N(j) + 1, n) \end{aligned}$$

b) $F^*(j, n) \geq 1 + F^*(N(j) + 1, n)$: From Lemma 1, F^* is monotonous, so we have.

$$F^*(j, n) \geq F^*(j, N(j) + 1) + F^*(N(j) + 1, n).$$

But $F^*(j, N(j) + 1) = 1$, since by definition $N(j)$ is the rightmost column for which a penalty ($f = 1$) can be avoided. Hence.

$$F^*(j, n) \geq 1 + F^*(N(j) + 1, n).$$

Composing claims **a** and **b** completes the proof of the theorem. Q.E.D.

```
Function NPL(j):integer;
var   STOP: boolean;
begin
(* 1. Find Next Penalty Location and assign back-values *)
   N(j) := j;
   STOP:= false;
   while (N(j) < n) and not(STOP) do
      if (∀i ∈ {1,...,m}∃u ∈ s_{i,j} : back[i, N(j), u] ≠ φ) then
         begin
            N(j) := N(j) + 1;
            for i := 1 to m do
               for all v ∈ s_{i,N(j)} do
                  back[i, N(j), v] = { u'  ∃u' ∈ s_{i,N(j)-1} such that:
                                     {       (N(j) = j or back[i, N(j) - 1, u'] ≠ φ) and
                                     {            f_{i,N(j)}(u', v) = 0
                                     { φ   otherwise
         end
      else STOP := true;
   NPL := N(j);
(* 2. Construct a partial solution for the columns [j,..., N(j) = NPL(j)] *)
   for i := 1 to m do
      find a u' such that (u' ∈ s_{i,N(j)}) and (back[i, N(j), u'] ≠ φ);
      x_{i,N(j)} := u';
   for t := N(j) - 1 downto j do
      for i := 1 to m do
         x_{i,t} := back[i, t, x_{i,t+1}]
end;
```

Fig. 4. Computing NPL (j).

C. Implementation

In the following we show a detailed implementation of an algorithm induced by Theorem 1, which is based on a recursive call of a function $F^*(j, n)$ (see Fig. 5) that computes a partial optimal solution. When called with $j = 1$, it computes the optimal value $F^*(X)$ and, as a by-product, proper values are assigned to X. This function calls another function, NPL(j) (see Fig. 4) to determine the column where the next penalty must be paid, and to assign proper values to the variables of X between columns j and $N(j)$.

The inputs for function **NPL** are j is the starting column, $f_{i,j}(x_{i,j-1}, x_{i,j})$ the local cost functions, and $s_{i,j}$ the set of values that variable $x_{i,j}$ may take, $s_{i,j} \subseteq \{1, \cdots, k\}$.

The algorithm proceeds from left to right in a greedy manner; it first scans the input matrix from column j to the right, until it finds a column where a penalty *must* be paid. The term greedy refers to the penalty location decisions, which are being taken "on the fly" in the forward scan. Then, the optimal partial solution is retrieved by going backward from column $N(j)$ to column j. This requires a backward pointer to indicate for each value v of a variable for which f may still get a zero value, its originating value in the previous column. These backward pointers are constructed during the forward (left to right) scan, and implemented by the data structure *back*:

$$back[i, j, v] = \begin{cases} u', & \exists u' \in s_{i,j-1} \text{ such that:} \\ & back[i, j-1, u'] \neq \phi \\ & \quad \text{and } f_{i,j}(u', v) = 0. \\ \phi, & \text{otherwise.} \end{cases}$$

```
Function F*(j,n):integer;
var   j': integer;
begin
   j' := NPL(j);
   if j' = n then  F* := 0
      else         F* := 1 + F*(j' + 1, n);
end;
```

Fig. 5. Computing $F^*(j, n)$ recursively.

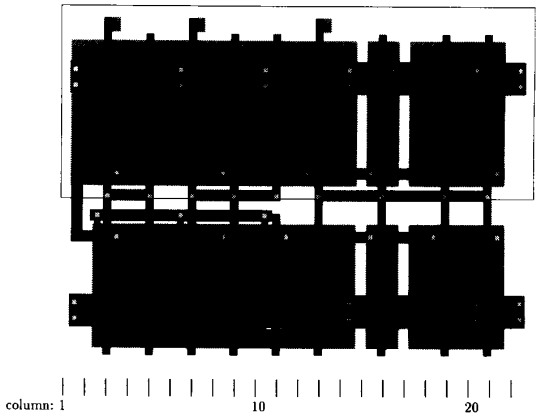


Fig. 6. Initial nonoptimized layout; width = 22 columns.

$back[i, j, v] = \phi$ means that assigning $x_{i,j}$ the value v necessarily results in a penalty between columns $j - 1$ and j.

The following corollary results from Theorem 1:

Corollary 1: Calling the function $F^*(j, n)$ with $j = 1$ solves Problem 1.

Proof: Follows from two facts:

a) The function $F^*(j, n)$ realizes the recursion formula of Theorem 1.

b) The function **NPL** indeed finds a column that obeys Definition 4. This follows from the fact that upon leaving the "while" loop, the variable $N(j)$ is set to the column with the highest index that can still be reached from j through zero-cost transitions, and is then returned as the function's value. The partial solution $X_{j,\ldots,N(j)}$ is obtained by a backward retrieval through the values of $back[i, t, v]$. Q.E.D.

To evaluate the time complexity of the algorithm, note that there are m rows and n columns in X. For each $x_{i,j}$ the algorithm checks all possible values of $x_{i,j}(s_{i,j})$ against all possible values of $x_{i,j-1}(s_{i,j-1})$. Since $\forall i, j$ there is $|s_{i,j}| \leq k$, it turns out that the overall time complexity is $O(mnk^2)$.

D. Results

The effectiveness of the algorithm described above is demonstrated in the CMOS cell in Figs. 6–9. There, the three optimization problems were addressed in the process of cell generation. In Fig. 6 a nonoptimized layout of a three-way NAND CMOS cell is shown. Its width is 22 columns. By applying the optimal flip algorithm, the layout in Fig. 7 results, in which the total cell width is reduced by two columns. Next, vertical contact relocation was employed, yielding an improvement of four columns, as shown in Fig. 8. Finally, the horizontal contact offset-

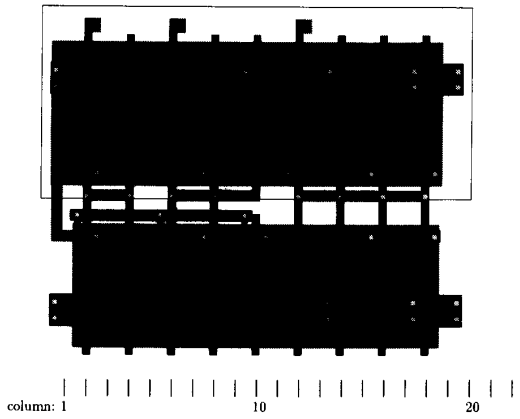


Fig. 7. Layout after optimal flipping; width = 20 columns.

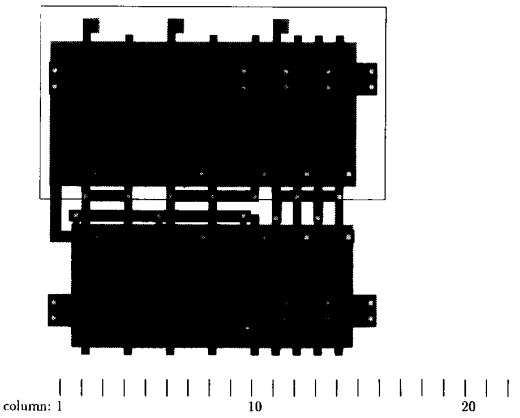


Fig. 8. Layout after optimal vertical contact relocation; width = 16 columns.

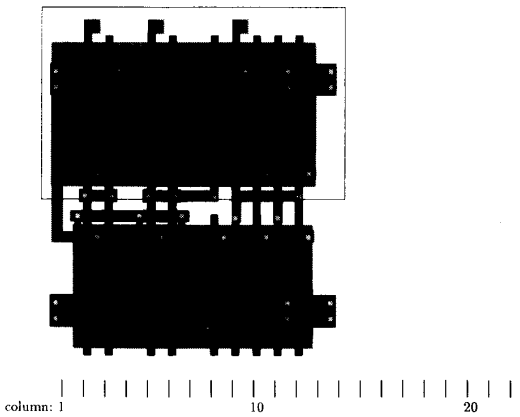


Fig. 9. Layout after optimal horizontal contact offsetting; width = 14 columns.

ting was introduced to create the layout in Fig. 9, where additional gain of two columns was achieved. Totally, right columns were reduced from cell width by applying our three algorithms in sequence.

IV. Conclusions

This paper addressed several layout problems occurring in various stages of a cell-generation process, such as device placement, intracell routing, and symbolic-to-shape mapping. We proved that this variety of problems can be captured in a single mathematical model that can be solved optimally in time linear in the size of the problem. The efficiency of the proposed algorithm is demonstrated in multirow fashioned layouts, such as those occurring in datapath architectures. The inherent property common to all of the layout problems discussed herein is that a local area penalty occurring in one row propagates through the rest of the rows. It was proved that problems having this property can be solved in linear time to minimize the total area penalty in the layout.

The model in this paper assumes that a single layout decision has a binary penalty: 1, if a space has to be inserted; 0, otherwise. This is a practical model for gridded layouts. For nongridded ones, a more general penalty model might be necessary. It is an open question whether the theory presented here may apply in such a case. It might be worth identifying more layout problems with similar properties, so that could benefit from the optimality and efficiency of the proposed algorithm.

References

[1] R. Bar-Yehuda, J. A. Feldman, R. Y. Pinter, and S. Wimer, "Depth-first-search and dynamic programming algorithms for efficient CMOS cell generation," *IEEE Trans. Computer-Aided Design*, vol. CAD-8, pp. 737-743, July 1989.

[2] S. Ben-Yehuda and R. Pinter, "Symbolic layout improvement using string matching based local transformations," in *Proc. Decennial Caltech Conf. VLSI*, Mar. 1989, pp. 227-239.

[3] J. Bhasker and S. Sahni, "Optimal linear arrangement of circuit components," *J. VLSI Computer Syst.*, vol. 1, pp. 87-109.

[4] S. Chakravarty, X. He, and S. S. Ravi, "Minimum area layout of series-parallel transistor network is NP-hard," *IEEE Trans. Computer-Aided Design*, vol. CAD-10, pp. 943-949, July 1991.

[5] C. K. Cheng, "Linear placement algorithms and applications to VLSI design," *Networks*, vol. 17, pp. 439-464, 1987.

[6] D. D. Hill, "Sc2—A hybrid automatic layout system," in *Proc. IC-CAD*, 1985, pp. 172-174.

[7] C. Hwang, Y. Hsieh, Y. Lin, and Y. Hsu, "An efficient layout style for 2-metal CMOS leaf cells and their automatic generation," in *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 481-486.

[8] J. F. Lee, "A layout compaction algorithm with multiple grid constraints," Research Report RC 16480 (#73183), IBM, Jan. 25, 1991.

[9] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. New York: Wiley, 1990.

[10] Y. Liao and C. K. Wong, "An algorithm to compact on VLSI symbolic layout with mixed constraints," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 2, pp. 62-69, Apr. 1983.

[11] R. Nair and A. Stauffer, "Optimal transistor orientation in CMOS cell layout," Research Report RC 13419 (#60053), IBM, Jan. 14. 1988.

[12] R. Nair, "MLG-A case for virtual grid symbolic layout without compaction," *Proc. IC-CAD*, 1987, pp. 180-183.

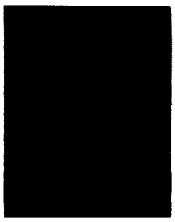
[13] R. D. Fiebrich, Y. Z. Liao, G. Koppelman, and E. Adams, "PSI: A symbolic layout system," *IBM Res. Develop.*, Sept. 1984.

[14] E. Seewann, S. L. Runyon, R. K. Montoye, Q. Nguyen, and J. C. Ridings, "VLSI circuit design for the RISC system/6000 processor," *IBM RISC System/6000 Technology*, SA23-2619, 1990, pp. 98-104.

[15] T. Uehara and W. M. van-Cleemput, "Optimal layout of CMOS functional arrays," *IEEE Trans. Computers*, vol. C-30, pp. 305-312, May 1981.

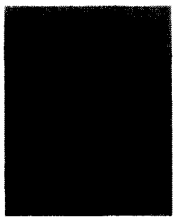
[16] N. Weste and K. Eshragian, *Principles of CMOS VLSI Design*. Reading, MA: Addison-Wesley, 1985, pp. 271-274.

[17] S. Wimer, R. Y. Pinter, and J. A. Feldman, "Optimal chaining of CMOS transistors in a functional cell," *IEEE Tran. Computer-Aided Design*, vol. CAD-6, no. 5, pp. 795-801, Sept. 1987.



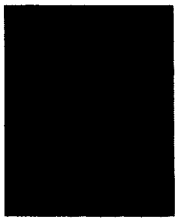
Jack A. Feldman received the B.Sc. and M.Sc. degrees in computer science from the Polytechnic Institute of Bucharest, and M.Sc. degree in electrical engineering from the Technion—Israel Institute of Technology, Haifa, Israel, in 1983 and 1988, respectively.

In 1985 he joined the IBM Haifa Research Group, where he has been research staff member since 1988. He was also a teaching assistant in the Electrical Engineering Department at the Technion. His research interests include combinatorial optimization, computational geometry, layout for integrated circuits, and CAD system integration.



Israel A. Wagner received the B.Sc. degree in computer engineering from the Technion—Israel Institute of Technology, Haifa, Israel, in 1987, cum laude, and M.Sc. degree in computer science from the Hebrew University, Jerusalem, Israel, in 1990, cum laude.

In 1990 he joined the IBM Haifa Research Group, where he has been research staff member since 1991. He was also a teaching assistant at the Hebrew University and a research engineer at General Microwave, Jerusalem, Israel. His research interests include manual and automatic VLSI design, computational geometry, and graph algorithms.



Shmuel Wimer received the B.Sc. and M.Sc. degrees in mathematics from Tel-Aviv University in 1977 and 1980, respectively, and the D.Sc. degree in electrical engineering from the Technion—Israel Institute of Technology, in 1988.

In 1977-1981 he worked for the Israeli Aircraft Industry in the area of simulation and modeling of aerodynamic systems. In 1981 he joined the VLSI design center of National Semiconductor in Tel-Aviv, and was there until 1985 in the area of development of CAD tools for logic simulation and optimization, and layout of VLSI circuits and systems. In 1985 he joined IBM Israel Science and Technology and Scientific Center, as a research staff member, where he works on development of tools and algorithms for the physical design of VLSI circuits and systems. From 1989 to 1992 he managed the physical design group, and he is presently a program manager for VLSI physical design.