Algorithmica (1989) 4: 263-291

Algorithmica
© 1989 Springer-Verlag New York Inc.

Floorplan Design of VLSI Circuits[1]

D. F. Wong[2] and C. L. Liu[3]

Abstract. In this paper we present two algorithms for the floorplan design problem. The algorithms are quite similar in spirit. They both use Polish expressions to represent floorplans and employ the search method of simulated annealing. The first algorithm is for the case where all modules are rectangular, and the second one is for the case where the modules are either rectangular or L-shaped. Our algorithms consider simultaneously the interconnection information as well as the area and shape information for the modules. Experimental results indicate that our algorithms perform well for many test problems.

Key Words. VLSI circuit layout, Floorplan design, Simulated annealing.

1. Introduction. Floorplan design [He 82], [Ma 82], [Ot 82], [LD 85], [PC 85], [WW 86] is the first stage of VLSI circuit layout. It is the problem of placing a given set of circuit modules in the plane to minimize a weighted sum of the following two quantities: (1) the area of the bounding rectangle containing all the modules; and (2) an estimation of the total interconnection wire length (or any suitable proximity measure). A given module can be classified as either *rigid* or *flexible*. A module is said to be *rigid* if its shape and dimensions are fixed. Predesigned library macrocells are examples of rigid modules. In this paper we consider both rectangular and L-shaped rigid modules. (A programmable logic array is an example of an L-shaped module.) A module is said to be *flexible* if its shape and dimensions are not fixed. Such flexibility represents the designer's freedom to manipulate the modules' internal structure at the floorplanning stage of design. We assume all flexible modules are rectangular. For each flexible module, we are given its area and limits of its allowed aspect ratio, where aspect ratio equals height divided by width. Floorplan design is a generalization of the classical placement problem [PV 79], [La 80], [SD 85] in which all modules are rectangular rigid modules. Traditional placement algorithms are no longer effective for this more general problem. For each flexible module, a floorplan design algorithm should be able to take advantage of the freedom of selecting a representative among many alternatives. For each L-shaped module, a floorplan design algorithm should consider the specific shape of the module rather than

[1] This work was partially supported by the Semiconductor Research Corporation under Contract 86-12-109, by the National Science Foundation under Grant MIP 8703273, and by a grant from the General Electric Company.
[2] Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, USA.
[3] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA.

Received June 28, 1987; revised November 4, 1987. Communicated by C. K. Wong.

just replacing the module by a bounding rectangle (which in general introduces unnecessary dead spaces).

A *floorplan* for n given modules (named $1, 2, \ldots, n$) consists of an enveloping rectangle R subdivided by horizontal and vertical line segments into n or more nonoverlapping rectilinear regions; n of these regions are labeled $1, 2, \ldots, n$. Region i must be large enough to accommodate module i. (Unlabeled regions, if any, correspond to dead spaces that sometimes are needed to achieve a more compact packing of the modules.) We require that the aspect ratio of R be between two given numbers p and q. We are also given an $n \times n$ interconnection matrix $C = (c_{ij})_{n \times n}$, with $c_{ij} \geq 0$, $1 \leq i, j \leq n$, which provides information on the wiring density between each pair of modules. The *center* of a region is the center of mass of the region. The *distance* between two regions is the Manhattan distance between their centers. For every pair of modules i and j, let d_{ij} be the distance between regions i and j. We use $W = \sum_{1 \leq i,j \leq n} c_{ij} d_{ij}$ as an estimate of the total interconnection wire length. Let A be the area of R. A and W will be referred to as *area* and *total wire length* of the floorplan, respectively. We use $A + \lambda W$ to measure the quality of a floorplan, where λ is a user-specified constant that controls the relative importance of A and W. The objective of the floorplan design problem is to find a floorplan such that $A + \lambda W$ is minimized. Our approach is quite flexible with respect to different quality measures. In particular, W can be replaced by any other proximity measure.

In this paper we present two algorithms for the floorplan design problem. The algorithms are quite similar in spirit. They both use Polish expressions to represent floorplans and employ a search method called simulated annealing [Ki 83]. The first algorithm is designed for the case where all modules are rectangular. The second algorithm is designed for the case where the modules are either rectangular or L-shaped. Many of the existing algorithms derive the final solution in two stages: they first determine the relative positions of the modules using primarily interconnection information, then they use the area and shape information to minimize the area of the bounding rectangle; whereas our algorithms consider simultaneously the interconnection information as well as the area and shape information. Also, existing algorithms that employ the method of simulated annealing either use a representation that leads to an unnecessarily large number of states and thus ultimately a slower rate of convergence [SS 85], or apply the search method only at a particular stage of a heuristic floorplan design algorithm [OG 84]. Finally, most existing algorithms can only handle rectangular modules.

This paper consists of two parts. Part 1 is for the case where all modules are rectangular. Part 2 is for the case where the modules are either rectangular or L-shaped.

2. Part 1: Rectangular Modules.

In this section we assume the given modules are all rectangular.[4] A *rectangular floorplan* is a floorplan where all the regions are rectangles. The rectangular regions in a rectangular floorplan are referred to

[4] A preliminary version of this section was presented in [WL 86].

as *basic rectangles.* We shall only consider rectangular floorplans. Let (A_1, r_1, s_1), $(A_2, r_2, s_2), \ldots, (A_n, r_n, s_n)$ be a list of n triplets of numbers corresponding to the n given modules. The triplet of numbers (A_i, r_i, s_i), with $r_i \le s_i$, specifies the area and the limits of the allowed aspect ratio for module i. Each module can also have a *fixed* or *free* orientation. If the orientation is free, rotation of the module (by 90°) is allowed. Otherwise, rotation of the module is not allowed. Let O_1 be the set of modules with fixed orientation, and O_2 be the set of modules with free orientation. Let w_i be the width and h_i be the height of module i, we must have:

(1) $w_i h_i = A_i$.
(2) $r_i \le h_i/w_i \le s_i$ if $i \in O_1$.
(3) $r_i \le h_i/w_i \le s_i$ or $1/s_i \le h_i/w_i \le 1/r_i$ if $i \in O_2$.

Note that module i is a rigid module if and only if $r_i = s_i$. Let x_i be the width and y_i be the height of region i. Clearly, we must have $x_i \ge w_i$ and $y_i \ge h_i$.

2.1. Slicing Floorplans. To *cut* a rectangle we mean to divide the rectangle into two rectangles by either a vertical or a horizontal line. A *slicing floorplan* is a rectangular floorplan with n basic rectangles that can be obtained by recursively cutting a rectangle into smaller rectangles (see Figure 1(a)). A slicing floorplan can be represented by an oriented rooted binary tree, called a *slicing tree* (see Figure 1(b)). Each internal node of the tree is labeled either $*$ or $+$, corresponding to either a vertical or a horizontal cut, respectively. Each leaf corresponds to a basic rectangle and is labeled by a number between 1 and n.

A slicing tree is a hierarchical description of the types of the cuts (vertical or horizontal) in a slicing floorplan. However, no dimensional information on the position of each cut is specified. In general, there are many floorplans represented by the same slicing tree. These floorplans differ in the geometric dimensions for the basic rectangles as well as in the adjacency relationship among the basic rectangles. We now define an *equivalence relation* $\sim$ on the set of all slicing floorplans. Let A and B be two slicing floorplans. We define $A \sim B$ iff they have the same slicing tree representation. The equivalence relation $\sim$ partitions the set

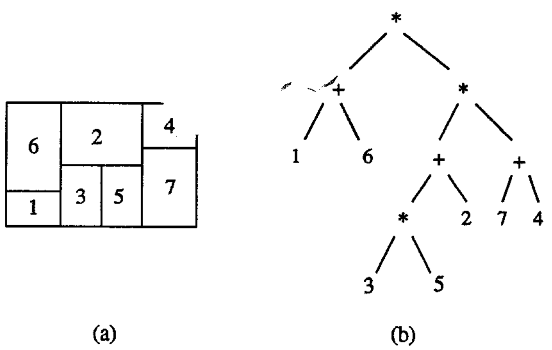


Fig. 1. Slicing floorplan and its slicing tree representation.

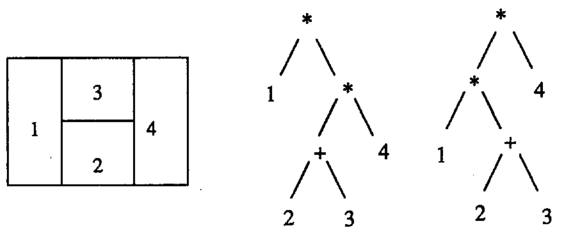


Fig. 2. Two different slicing trees for the same slicing floorplan.

of slicing floorplans into equivalence classes. Each equivalence class of slicing floorplans with n basic rectangles is called a *slicing structure.*

Note also that for a given slicing floorplan, there may be more than one slicing-tree representation (see Figure 2). The different slicing trees that represent the same floorplan correspond to different orders in which consecutive vertical cuts and consecutive horizontal cuts are made. A *skewed slicing tree* is a slicing tree in which no node and its right son have the same label in $\{*, +\}$ (see Figure 3). A slicing tree obtained by making consecutive vertical cuts in the order of from right to left, and making consecutive horizontal cuts in the order of from top to bottom is a skewed slicing tree. All other slicing trees representing the same floorplan are not skewed. Hence we can use skewed slicing trees to represent slicing floorplans. We have the following lemma, the proof of which is rather obvious and is thus omitted.

LEMMA 1. *There is a* 1-1 *correspondence between all skewed slicing trees with n leaves and all slicing structures with n basic rectangles.*

2.2. Solution Space. A slicing structure represents a set of equivalent slicing floorplans. We shall see in Section 2.4 that we can efficiently select the "best" floorplan among all these equivalent floorplans. Thus, instead of using the set of all slicing floorplans as the solution space, we can use the set of all slicing structures as the solution space. This substantially reduces the size of the solution space. We use a representation of slicing structures called *normalized Polish expressions* that is particularly suitable for the method of simulated annealing. Consequently, we use the set of normalized Polish expressions as the solution space.

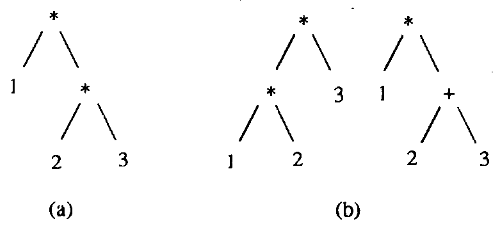


Fig. 3. (a) A nonskewed slicing tree; (b) skewed slicing trees.

A binary sequence $b_1 b_2 \cdots b_m$ is said to be a *balloting sequence* iff for any k, $1 \leq k \leq m$, the number of 0's is less than the number of the 1's in $b_1 b_2 \cdots b_k$. Let σ be a function σ: $\{1, 2, \ldots, n, *, +\} \rightarrow \{0, 1\}$ defined by $\sigma(i) = 1$, $1 \leq i \leq n$, and $\sigma(*) = \sigma(+) = 0$. A sequence $\alpha_1 \alpha_2 \cdots \alpha_{2n-1}$ of elements from $\{1, 2, \ldots, n, *, +\}$ is said to have the *balloting property* if $\sigma(\alpha_1)\sigma(\alpha_2) \cdots \sigma(\alpha_{2n-1})$ is a balloting sequence.

If we traverse a slicing tree in postorder [Ah 74], we obtain a *Polish expression.* Note that a Polish expression is a sequence of elements from $\{1, 2, \ldots, n, *, +\}$ with the balloting property. A Polish expression is said to be *normalized* if there is no consecutive $*$'s or $+$'s in the sequence. (For example, $1\,2 + 4\,3 * +$ is a normalized Polish expression, while $1\,2 + 4\,3 * *$ is not.) It is easy to see that the Polish expression obtained from a skewed slicing tree is normalized. Consequently, we have the following lemma and theorem. (Theorem 1 follows from Lemmas 1 and 2.)

LEMMA 2. *There is a* 1-1 *correspondence between the set of normalized Polished expressions of length* $2n-1$ *and the set of skewed slicing trees with n leaves.*

THEOREM 1. *There is a* 1-1 *correspondence between the set of normalized Polish expressions of length* $2n-1$ *and the set of slicing structures with n basic rectangles.*

A slicing tree can be viewed either as a *top down* or a *bottom up* description of a slicing floorplan. From a top down point of view, a slicing tree specifies how a given rectangle is cut into smaller rectangles by horizontal and vertical cuts. From a bottom up point of view, a slicing floorplan can also be described by how smaller slicing floorplans are combined recursively to yield larger slicing floorplans. Indeed, we can interpret the symbols $*$ and $+$ as two binary operators between slicing floorplans. If A and B are slicing floorplans, we can interpret $A + B$ and $A * B$ as the resulting slicing floorplans obtained by placing B on top of A, and B to the right of A, respectively, as shown in Figure 4. (From now on, we shall refer to the elements in $\{1, 2, \ldots, n\}$ as *operands,* and the elements in $\{*, +\}$ as *operators.*)

2.3. Neighborhood Structure. A sequence $d_1 d_2 \cdots d_k$ of k operators is called a *chain* of *length k.* (Note that $d_i \neq d_{i+1}$ in a normalized Polish expression, for all $1 \leq i \leq k-1$.) We use $l(c)$ to denote the length of a chain c. A chain of length 0 is defined to be the empty sequence. It is clear that for every $k > 0$, there are only two possible types of chains of length k in a normalized Polish expression:

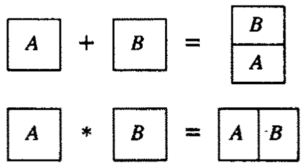


Fig. 4. Binary operators for slicing floorplans.

$$\begin{array}{ccccccc} 1 & 2 & 3 & \underline{* \;\; +} & 5 & 4 & \underline{+ \;\; *} \\ \pi_1 & \pi_2 & \pi_3 & c_3 & \pi_4 & \pi_5 & c_5 \end{array}$$

$c_0 = c_1 = c_2 = c_4 =$ the empty sequence.

Fig. 5. Chains in a Polish expression.

$*+*+*\cdots$ and $+*+*+\cdots$. We define the *complement* of a chain to be the chain obtained by interchanging the operators $*$ and $+$ (e.g., the complement of $*+*+*$ is $+*+*+$). Let $\boldsymbol{\alpha} = \alpha_1\alpha_2 \cdots \alpha_{2n-1}$ be a normalized Polish expression. Note that $\boldsymbol{\alpha}$ can also be written as $c_0\pi_1c_1\pi_2c_2 \cdots c_{n-1}\pi_nc_n$, where $\pi_1, \pi_2, \ldots, \pi_n$ is a permutation of $1, 2, \ldots, n$, the c_i's are chains (possibly of zero length), and $\sum_i l(c_i) = n-1$ (see Figure 5). Two operands in $\boldsymbol{\alpha}$ are said to be *adjacent* iff they are consecutive elements in $\pi_1 \cdots \pi_n$. An operand and an operator are said to be *adjacent* iff they are consecutive elements in $\alpha_1\alpha_2 \cdots \alpha_{2n-1}$. We define three types of moves, M1, M2, and M3, that can be used to modify a given normalized Polish expression. The definitions of M1, M2, and M3 are as follows:

M1. Swap two adjacent operands.
M2. Complement a chain of nonzero length.
M3. Swap two adjacent operand and operator.

See Figure 6 for a pictorial illustration of these three types of moves.

It is clear that M1 and M2 always produce a normalized Polish expression. This is not the case for M3. In fact, M3 might produce a sequence that contains identical consecutive operators or a sequence that violates the balloting property. We shall only accept those M3 moves that result in normalized Polish expressions. (Whether a given M3 move will result in a normalized Polish expression can be tested efficiently in constant time.)

These three types of move are used to define the neighborhood structure of our solution space. Two normalized Polish expressions are said to be *neighbors* if one can be obtained from the other via one of these three moves. For a given normalized Polish expression $\boldsymbol{\alpha}$, we first randomly select a type of move, and then randomly select a neighbor according to the type of move selected. The three types of moves are sufficient to ensure that it is possible to transform any normalized Polish expression into any other via a sequence of moves as the following theorem shows.

THEOREM 2. *Let $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ be two normalized Polish expressions. There exists a sequence of at most $2n^2+2n$ neighboring normalized Polish expressions between $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$.*

PROOF. We observe that given any normalized Polish expression $\boldsymbol{\alpha}$, $\boldsymbol{\alpha}$ is reachable from $\boldsymbol{\alpha}_0 = 12*3*\cdots*n$ via a sequence of moves, and vice versa. Therefore, for any pair of normalized Polish expressions $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$, there exists a sequence of moves that will transform $\boldsymbol{\alpha}'$ into $\boldsymbol{\alpha}$ with $\boldsymbol{\alpha}_0$ as an intermediate expression. In order to go from $\boldsymbol{\alpha}$ to $\boldsymbol{\alpha}_0$, we need at most $n^2/2$ moves of type M1 to sort the operands into the ordering $1, 2, 3, \ldots, n$; at most another $n^2/2$ moves of type M3

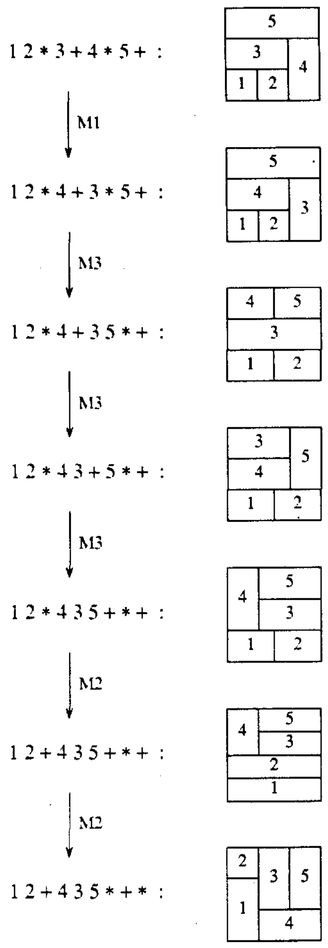


Fig. 6. Illustration of the moves.

to convert the expression into the form $12 \bullet 3 \bullet \cdots \bullet n$ where $\bullet$ is an operator, and at most n moves of type M2 to convert all operators into $*$. Hence it takes at most n^2+n moves to convert $\boldsymbol{\alpha}$ into $\boldsymbol{\alpha}_0$. Similarly, it takes at most n^2+n moves to convert $\boldsymbol{\alpha}_0$ to $\boldsymbol{\alpha}'$. Therefore, the total number of moves needed is at most $2n^2+2n$. □

2.4. Cost Function. Let $\boldsymbol{\alpha}$ be a normalized Polish expression. The expression $\boldsymbol{\alpha}$ represents a set $\mathbf{S}_\alpha$ of equivalent slicing floorplans. Roughly speaking, the relative positions of the modules in equivalent floorplans are essentially the same. Intuitively, those floorplans that have smaller area will in general also have shorter total wire length because the modules are pulled "closer together" in these floorplans. Let $\mathbf{f}_\alpha$ be a floorplan in $\mathbf{S}_\alpha$ with minimum area. Let $A(\boldsymbol{\alpha})$ and $W(\boldsymbol{\alpha})$ be the area and the total wire length of $\mathbf{f}_\alpha$, respectively. The cost function we use is $\Psi(\boldsymbol{\alpha}) = A(\boldsymbol{\alpha}) + \lambda W(\boldsymbol{\alpha})$. We show in this section how to compute efficiently

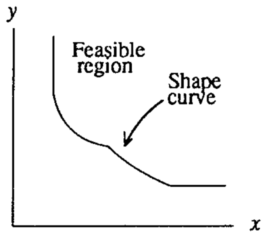


Fig. 7. Shape curve and feasible region.

$\Psi(\boldsymbol{\alpha})$ for a given normalized Polish expression $\boldsymbol{\alpha}$. Our method of cost computation is based on [Ot 83] and [St 83]. Since in our simulated annealing algorithm each move only leads to a minor modification of the present Polish expression, we can take advantage of the cost computation for the preceding move to save substantially computational effort for the current cost computation.

2.4.1. Basic Method. Shape Curve. Let Γ be a continuous curve on the plane. Γ is said to be *decreasing* if for any two points (x_1, y_1), (x_2, y_2) on Γ with $x_1 \le x_2$, we must have $y_1 \ge y_2$. Γ is a *shape curve* if it satisfies the following conditions: (1) it is decreasing and lies completely in the first quadrant; (2) $\exists k > 0$ such that all lines of the form $x = a$, $a > k$, intersect Γ; and (3) $\exists k > 0$ such that all lines of the form $y = b$, $b > k$, intersect Γ. Note that a shape curve Γ partitions the first quadrant into two connected regions. One of the connected regions that contains the points (a, a) for all large a's is called the *feasible region* (see Figure 7).

The shape curves in Figure 8 correspond to different kinds of shape constraints where the shaded areas are the feasible regions. A point in the feasible region corresponds to the dimensions of a basic rectangle that can accommodate a given module. (The x and y coordinates of the point are the horizontal and vertical

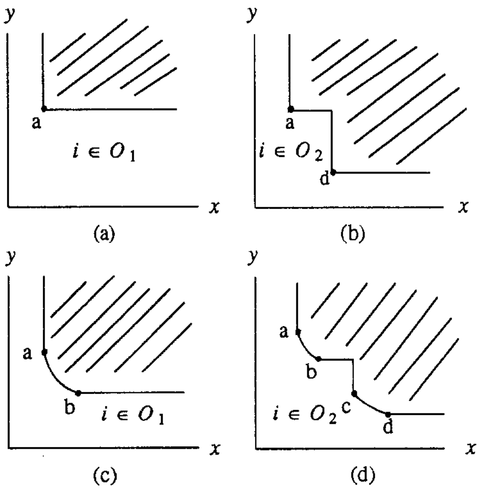


Fig. 8. Shape curves for different shape constraints.

dimensions of the basic rectangle.) Figure 8(a) and (b) corresponds to the case in which the module is rigid. Figure 8(c) and (d) corresponds to the case in which the module is flexible. Let H be the hyperbola $xy = A_i$, L_1 be the line $y = s_i x$, L_2 be the line $y = r_i x$, L_3 be the line $y = (1/r_i)x$, and L_4 be the line $y = (1/s_i)x$. In Figure 8(a)-(d) a, b, c, and d are the intersections between the hyperbola H and the lines L_1, L_2, L_3, and L_4, respectively.

Let Γ and Λ be two shape curves. We define $\Gamma + \Lambda$ to be the curve

$$\{(u, v+w) | (u, v) \in \Gamma \text{ and } (u, w) \in \Lambda\}$$

and define $\Gamma * \Lambda$ to be the curve

$$\{(u+v, w) | (u, w) \in \Gamma \text{ and } (v, w) \in \Lambda\}.$$

In other words, $\Gamma + \Lambda$ is obtained by adding the two curves Γ and Λ along the y-direction. $\Gamma * \Lambda$ is obtained by adding the two curves Γ and Λ along the x-direction. It is easy to see that $\Gamma + \Lambda$ and $\Gamma * \Lambda$ are also shape curves. Moreover, they are piecewise linear if Γ and Λ are both piecewise linear. For piecewise linear shape curves, there is an efficient algorithm to compute $\Gamma + \Lambda$ and $\Gamma * \Lambda$. Since a piecewise linear shape curve is completely characterized by an ordered list of all the "corners" of the curve, hence to add two piecewise linear shape curves (along either directions), we only need to add up the curves at the "corners."

Area Computation. Let $\mathbf{T}_\alpha$ be the slicing tree corresponding to $\boldsymbol{\alpha}$. For each node v in $\mathbf{T}_\alpha$, the subtree rooted at v defines a slicing structure R_v. The shape constraints for the modules in R_v define the shape constraints for R_v. Let Γ_v be the shape curve representing the shape constraints for R_v. For every three nodes u, v, w in $\mathbf{T}_\alpha$ with v being the father of u and w, Γ_v is either $\Gamma_u * \Gamma_w$ or $\Gamma_u + \Gamma_w$ depending on whether v is $*$ or $+$. Hence, all the Γ_v's can be computed by adding up the shape curves of the basic rectangles (the leaves). We assume the shape curves for the basic rectangles are all piecewise linear. In this case we can efficiently compute all the Γ_v's. (Note that we can have piecewise linear approximation of the shape curves for the basic rectangles with arbitrary precision.) Once we have computed all the Γ_v's, we can compute the area measure $A(\boldsymbol{\alpha})$ from Γ_r where r is the root of $\mathbf{T}_\alpha$. (Γ_r is the shape curve for the slicing structure $\boldsymbol{\alpha}$.) Let (a_1, b_1) be the point of intersection between Γ_r and the line $y = px$. Let (a_{l+1}, b_{l+1}) be the point of intersection between Γ_r and the line $y = qx$. Let (a_2, b_2), $(a_3, b_3), \ldots, (a_l, b_l)$ be all the "corners" of the curve that lies between the lines $y = px$ and $y = qx$ (i.e., $p \leq b_j/a_j \leq q$). Let $\mathbf{P} = \{(a_1, b_1), (a_2, b_2), \ldots, (a_{l+1}, b_{l+1})\}$. The next theorem shows that we can compute $A(\boldsymbol{\alpha})$ by only examining the points in $\mathbf{P}$.

THEOREM 3. *$A(\boldsymbol{\alpha}) = a_i b_i$ where (a_i, b_i) is a point in* $\mathbf{P}$ *such that $a_i b_i$ is minimum.*

PROOF. Let (a, b) be the dimensions of a minimum area floorplan. It is clear that the point (a, b) must be on Γ_r. Since if that is not the case, the intersection between Γ_r and the line joining $(0, 0)$ and (a, b) would give the dimensions of a floorplan with smaller area. To show that (a, b) is one of the (a_i, b_i)'s, we observe that, along a given line segment I in the first quadrant, the function $f(x, y) = xy$ attains its minimum value at one of the two endpoints of I. □

Wire Length Computation. Since $\mathbf{T}_\alpha$ is a slicing tree representation of $\mathbf{f}_\alpha$, each node v in $\mathbf{T}_\alpha$ corresponds to a rectangle K_v in $\mathbf{f}_\alpha$. Let (x_v, y_v) be the dimensions of K_v. After we have computed (x_r, y_r) for K_r where r is the root of $\mathbf{T}_\alpha$, we can recursively compute the dimensions of all the basic rectangles as follows: suppose that we have already determined (x_v, y_v) for some internal node v. Let u be the left son of v and w be the right son of v. Consider the case in which the operator at v is +. That is, we have K_v is formed by putting K_w directly on top of K_u. We have $x_u = x_w = x_v$; $y_u = y_v - y_w$; and $y_w = y'$ where y' is the y-coordinate of the point of intersection between the line $x = x_v$ and the curve Γ_w. (If there is more than one intersection, choose the one with the smallest y coordinate.) Similarly, for the case in which the operator as v is *, we have $y_u = y_w = y_v$; $x_u = x_v - x_w$; and $x_w = x'$ where x' is the x coordinate of the point of intersection between the line $y = y_v$ and the curve Γ_w.

Let (c_v^x, c_v^y) be the coordinates of the center of K_v. (We assume that the southwest corner of $\mathbf{f}_\alpha$ is placed at the origin.) Let (l_v^x, l_v^y) be the coordinates of the point of the southwest corner of K_v. Clearly, we have $c_v^x = l_v^x + \frac{1}{2}x_v$ and $c_v^y = \frac{1}{2}l_v^y$. Consequently, to compute the centers of the basic rectangles, it suffices to compute (l_v^x, l_v^y) for all v. Note that $(l_r^x, l_r^y) = (0, 0)$. We can recursively compute all (l_v^x, l_v^y)'s as follows. Suppose we have already determined (l_v^x, l_v^y) for some v. Let u be the left son of v and w be the right son of v. We have $(l_u^x, l_u^y) = (l_v^x, l_v^y)$. If the operator at v is +, we have $l_w^x = l_v^x$ and $l_w^y = l_v^y + y_u$. If the operator at v is *, we have $l_w^x = l_v^x + x_u$ and $l_w^y = l_v^y$.

Note that the centers of the basic rectangles are just the (c_v^x, c_v^y)'s for all leaf nodes v. It follows that the d_{ij}'s (the distances between the basic rectangles) and consequently $W(\boldsymbol{\alpha})$ can be easily computed from the (c_v^x, c_v^y)'s.

2.4.2. Incremental Computation of Cost Function. For a given normalized Polish expression, the shape curves associating with the nodes of its slicing tree are needed in both the area and the wire length computation. Let $\boldsymbol{\alpha}' = \alpha_1'\alpha_2' \cdots \alpha_{2n-1}'$ be the Polish expression obtained from $\boldsymbol{\alpha} = \alpha_1\alpha_2 \cdots \alpha_{2n-1}$ after a move. For $1 \le i \le 2n-1$, let T_i and T_i' be the trees rooted at α_i and α_i', and let Γ_i and Γ_i' be the shape curves for α_i and α_i', respectively. In our simulated annealing algorithm, each move leads to only a minor modification of the Polish expression currently being examined. Thus, in general, $\Gamma_i = \Gamma_i'$ for many i's. Therefore, in computing the cost for $\boldsymbol{\alpha}'$, we need only update those shape curves that are changed. Such an observation will substantially improve the efficiency of our algorithm because we need to compute the costs for a large number of solutions. We show in Theorem 4 that the two sets of shape curves $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_{2n-1}\}$ and $\{\Gamma_1', \Gamma_2', \ldots, \Gamma_{2n-1}'\}$ differ only at a set of vertices that lie along one or two paths in each of $\mathbf{T}_\alpha$ and $\mathbf{T}_{\alpha'}$.

For a type M1 or type M3 move, we swap two elements α_i and α_j. We refer to the elements α_i and α_j as the *base elements* in $\boldsymbol{\alpha}$, and the elements α_i' and α_j'

as the *base elements* in $\boldsymbol{\alpha}'$. For a type M2 move, we complement a chain $\alpha_k\alpha_{k+1}\alpha_{k+2}\cdots\alpha_l$. We refer to the elements $\alpha_k, \alpha_{k+1}, \ldots, \alpha_l$ as the *base elements* in $\boldsymbol{\alpha}$, and the elements $\alpha'_k, \alpha'_{k+1}, \ldots, \alpha'_l$ as the *base elements* in $\boldsymbol{\alpha}'$. The following lemma follows from the definition of base elements.

LEMMA 3.

(1) *α_i is a base element if and only if α'_i is a base element.*
(2) *If α_i is not a base element, then $\alpha_i = \alpha'_i$.*

Let us *mark* all the base elements together with all their ancestors in both $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$. Let $\mathbf{M}$ be the set of marked elements in $\boldsymbol{\alpha}$ and $\mathbf{M}'$ be the set of marked elements in $\boldsymbol{\alpha}'$. The elements which are not in $\mathbf{M}$ or $\mathbf{M}'$ are said to be *unmarked.* We have the following lemmas:

LEMMA 4. *α_i is unmarked if and only if α'_i is unmarked.*

PROOF. Note that T_i corresponds to $\boldsymbol{\beta}_i = \alpha_j\alpha_{j+1}\cdots\alpha_i$ and T'_i corresponds to $\boldsymbol{\beta}'_i = \alpha'_k\alpha'_{k+1}\cdots\alpha'_i$ where j and k are the unique indices such that $\sigma(\alpha_j)\sigma(\alpha_{j+1})\cdots\sigma(\alpha_i)$ and $\sigma(\alpha'_k)\sigma(\alpha'_{k+1})\cdots\sigma(\alpha'_i)$ are balloting sequences. Suppose α_i is unmarked. This means that $\boldsymbol{\beta}_i$ does not contain any base elements. It follows from Lemma 3 that $\alpha'_m = \alpha_m$ for all $j \le m \le i$. Hence $\sigma(\alpha'_j)\sigma(\alpha'_{j+1})\cdots\sigma(\alpha'_i)$ is a balloting sequence. Therefore, we must have $k = j$ and $\boldsymbol{\beta}'_i = \boldsymbol{\beta}_i$. Again, it follows from Lemma 3 that $\boldsymbol{\beta}'_i$ does not contain any base elements. Therefore α'_i is unmarked. Similarly, we can show the converse by interchanging the roles of $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ in the above arguments. □

LEMMA 5. *If α_i is unmarked (or equivalently α'_i is unmarked), then $\Gamma_i = \Gamma'_i$.*

PROOF. It follows from Lemma 4 that $T_i = T'_i$. Thus we have $\Gamma_i = \Gamma'_i$. □

LEMMA 6. *$\mathbf{M}$ corresponds to one or two paths in $\mathbf{T}_{\boldsymbol{\alpha}}$; $\mathbf{M}'$ corresponds to one or two paths in $\mathbf{T}_{\boldsymbol{\alpha}'}$.*

PROOF. For a type M1 or type M3 move, there are only two base elements in $\boldsymbol{\alpha}$. $\mathbf{M}$ corresponds to the two paths from the two base elements in $\boldsymbol{\alpha}$ to the root. For a type M2 move, let $\alpha_k, \alpha_{k+1}, \alpha_{k+2}, \ldots, \alpha_l$ be the base elements in $\boldsymbol{\alpha}$. In this case, α_i is the right son of α_{i+1} for all $k \le i \le l-1$. $\mathbf{M}$ corresponds to the path from α_k to the root. Similarly, we can show that $\mathbf{M}'$ corresponds to one or two paths in $\mathbf{T}_{\boldsymbol{\alpha}'}$. □

It follows from Lemma 5 that we need only recompute those shape curves that correspond to unmarked elements after each move. The next theorem clearly follows from the above lemmas.

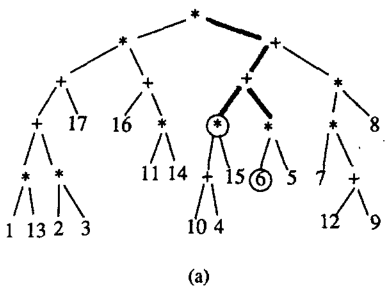
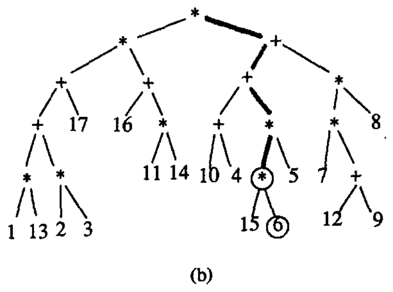



Fig. 9. The path(s) in $\mathbf{T}_\alpha$ and $\mathbf{T}_{\alpha'}$ after a type M3 move.

THEOREM 4. *Let α' be the Polish expression obtained from α after a move. The shape curves Γ_i's and the shape curves Γ_i''s differ only at a set of vertices that lie along one or two paths in each of $\mathbf{T}_\alpha$ and $\mathbf{T}_{\alpha'}$.*

Figure 9 illustrates the statement in Theorem 4. In the given example, the Polish expression is of the form $\cdots 17+16\cdots * 6\cdots 9+*8\cdots$. Consider the type M3 move of swapping $*$ and 6. Figure 9(a) and (b) shows the path(s) corresponding to this move in $\mathbf{T}_\alpha$ and $\mathbf{T}_{\alpha'}$, respectively.

Finally, we should point out how the d_{ij}'s can also be computed incrementally. Clearly, after each move, it is only necessary to recompute those terms $c_{ij}d_{ij}$ where the positions of one or both of the centers of the basic rectangle i and j have changed. Unfortunately, in general, even for a minor modification of a Polish expression, the positions of the centers of many of the basic rectangles will be changed (although most of the changes are small). In view of this, we can reduce the computation time by using a less accurate wire length estimation. Suppose that there is a grid of size ρ (i.e., the distance between every two adjacent lines in the grid is ρ) imposed on top of the floorplan. We define the *modified center* of a basic rectangle as the grid point that is closest to the center of the basic rectangle. (In the case when there are more than one such grid points, we use the one with the smallest x and y coordinates.) For every pair of basic rectangles i and j, we define d'_{ij} as the distance between their modified centers. Let $W'(\alpha)=\sum_{1\le i,j\le n} c_{ij}d'_{ij}$. We can use $A(\alpha)+\lambda W'(\alpha)$ as our cost function. (Note that $W'(\alpha)=W(\alpha)$ if ρ is 0.) Clearly, the number of *modified centers* which remain unchanged after a move decreases as ρ increases. Consequently, the computation time for updating the d'_{ij}'s is inversely proportional to ρ.

2.5. Annealing Schedule. We use a fixed ratio temperature schedule $T_k = rT_{k-1}$, $k = 1, 2, \ldots$. Our experiments indicate that setting $r = 0.85$ produces very satisfactory results.

To determine the value of T_0, we perform a sequence of random moves and compute the quantity Δ_{avg}, the average value of the magnitude of change in cost per move. We should have $e^{-\Delta_{\text{avg}}/T_0} = P \approx 1$ so that there will be a reasonable probability of acceptance at high temperatures. This suggests that $T = -\Delta_{\text{avg}}/\ln(P)$ is a good choice for T_0.

Our algorithm can start with any initial normalized Polish expression. In our experiments, we start with the Polish expression $12 * 3 * 4 * \cdots * n *$ which corresponds to placing the n modules horizontally next to each other. This Polish expression is usually far from the optimal solution.

At each temperature, we try enough moves until either there are N downhill moves or the total number of moves exceeds $2N$ where $N = \gamma n$ where γ is a user-specified constant. We terminate the annealing process if the number of accepted moves is less than 5% of all moves made at a certain temperature or the temperature is low enough.

2.6. Experimental Results. We have implemented our algorithm in PASCAL on a PYRAMID computer. The experimental results are summarized in Tables 1-3. The modules in all our tests problems are flexible modules with free orientations allowed and $r_i = 1/s_i$. The running time for the test problems range from 1 CPU minute for the 15-modules problem to 13 CPU minutes for the 40-modules problem.

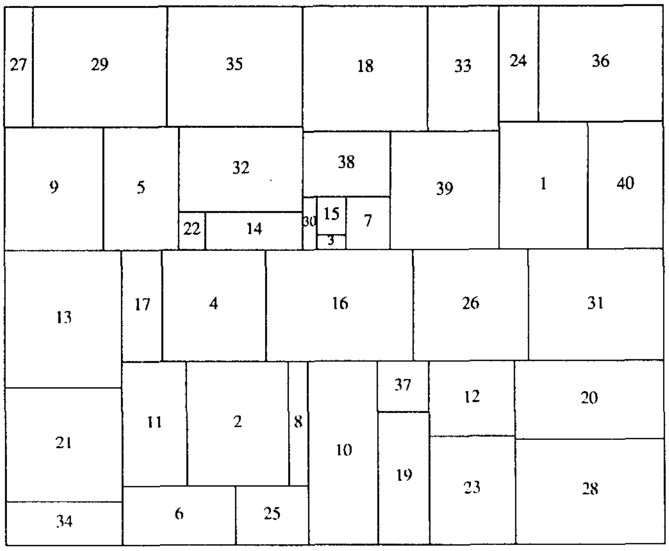
The problems in Table 1 are randomly generated. The areas of the modules (A_i) are chosen uniformly between 1 and 20. For P4 and P5, all modules have the same aspect ratio (s_i). It is 2 for P4 and 3 for P5. For the other problems, each s_i is chosen uniformly between 1 and 3. The maximum aspect ratio allowed for the final chip is 2. The interconnection matrices are also randomly generated such that the weights (c_{ij}) are between 0 and 1. Various values of λ are used in these test problems. Total area in the table is the sum of the areas of the given modules and hence is a lower bound on the area of the final chip. Comparing

Table 1

				Random feasible solution		Initial S.A. solution		Final S.A. solution	
Problem	n	λ_w	Total area	A	W	A	W	A	W
P1	15	1	137.08	491.64	106.02	514.59	59.41	137.86	34.97
P2	20	1	198.88	745.35	258.84	885.71	202.30	202.17	80.49
P3	20	0	197.15	808.72	423.90	819.64	276.50	198.98	197.40
P4	25	0	244.68	1165.70	398.21	1334.90	296.60	245.43	209.70
P5	25	1	238.15	1026.60	576.71	876.04	344.30	244.63	151.90
P6	30	0.5	333.92	1549.60	1023.20	2458.20	865.70	340.15	294.90
P7	30	0	314.45	1476.90	1095.70	2130.10	936.20	319.40	429.20
P8	40	1	407.44	1934.20	1002.30	3965.40	999.30	422.95	265.80

the results for problems with the same number of modules (i.e., P2 and P3, P4 and P5, and P6 and P7), we discover that the final area (A) of P2 and P3 are both very close to optimal while the final total wire length (W) of P3 is substantially higher than that of P2. Since we generate their interconnection matrices using the same probability distribution, we might expect the values of W to be about the same. The large difference in the final values of W is due to the fact that the value of λ is 0 for P3, and hence the algorithm makes no attempt to minimize W. We also observe that we can obtain reasonable tradeoffs between area and wire length without substantially increasing the area. Similar observation can be made for the other two pairs of problems. For some of the test problems with nonzero λ, we obtain solutions in which the value of $A+\lambda W$ is about the same while the value of A is smaller and the value of W is larger. This is to be expected because of the tradeoffs between A and W in the cost function. Also, for problems with zero λ, even though the final wire lengths are large, they are much smaller than those in random solutions. Since modules that are closely packed will in general lead to a reduction of total wire length too. For some of the test problems, suboptimality in area is due to the presence of very rigid modules (small s_i), and the tradeoffs between area and wire length. Figure 10 shows the final floorplan for problem P8 which contains 40 modules.

Table 2 is a summary of results for a random problem with 20 modules. The sum of the areas of the modules in this problem is 195.37. We used various value of λ to demonstrate the tradeoffs between area and wire length. As we increase λ from 0 to 3, we observe that A increases from 196.42 to 220.30 while W



34 21 + 13 + 6 25 * 11 2 * 8 * + 10 * 19 37 + * 23 12 + * 28 20 + * 17 4 * 16 *
26 * 31 * + * 9 5 * 22 14 * 32 + * 27 29 * 35 * + 30 3 15 + * 7 * 38 + 39 * 18
33 * + * 1 40 * 24 36 * + * +

Fig. 10. Final floorplan for problem P8.

Table 2

	S.A. solution					
λ_w	A	W	A	$A+W$	$A+2W$	$A+3W$
0	196.42	152.10	196.42†	348.52	500.62	652.72
1	206.60	103.70	206.60	310.30†	414.00	517.70
2	215.33	95.78	215.33	311.11	406.89†	502.67
3	220.30	93.96	220.30	314.26	408.22	502.18†

Table 3

Module ratio	Chip ratio	Total area	A
1	1	229.17	291.23
2	1	—	231.65
2	2	—	230.52
3	3	—	230.19

decreases from 152.1 to 93.96. We observe that those entries marked by a dagger (†) are the values of the cost function. They should be smaller than other values in the same column. This is indeed the case.

Table 3 shows the results for another 20-modules problem. For this problem, all modules have the same aspect ratio s which is referred to as module ratio in the table. The column chip ratio is the maximum aspect ratio allowed for the final chip. The value of λ is set to 0. This problem demonstrates the effect of module ratio and chip ratio on the final area of the chip. We observe that by relaxing either the module ratio or the chip ratio we can reduce the final area. We also observe that a module ratio of 2 gives enough flexibility for achieving close to optimal final area.

3. Part 2: Rectangular and L-Shaped Modules. In this section we present an algorithm to produce floorplans for rectangular and L-shaped modules. (Figure 11 shows examples of L-shaped module.) This algorithm is very similar in spirit to the algorithm described in the last section. We also use the same search method of simulated annealing, similar floorplan representation (Polish expressions with new operators), and similar ways to modify a floorplan locally. Not only can our algorithm handle L-shaped modules, in the case where all the modules are

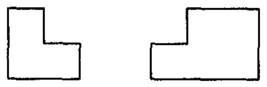


Fig. 11. Examples of L-shaped modules.

rectangular, our algorithm will, in general, be able to produce nonslicing rectangular floorplans.

For each module, there is a given set of possible shapes, dimensions, and orientation for the module. Each possible choice of shape, dimensions, and orientation for a module is called an *instance* of the module. (Clearly, a nonrotatable rigid module has only one instance.) We assume in the last section that all modules are rectangular. Also, we consider only slicing floorplans there. In that case, the technique of adding up shape curves allows us to consider simultaneously all possible instances of the modules efficiently. The situation in this section is slightly more complicated because there are L-shaped modules and therefore it is necessary to consider more general type of floorplans. The idea of adding up shape curves is no longer applicable. Instead, we examine the instances of a module one at a time and rely on probabilistic techniques to make selections from the different instances.

Let $\mathbf{\Omega}$ denote the set of rectangular and L-shaped geometric figures of all possible sizes and dimensions. For each module i, let $\mathbf{G}_i$, $\mathbf{G}_i \subseteq \mathbf{\Omega}$, denote the set of all possible instances for the module. We define four binary operators and one unary operator that operate on the geometric figures in $\mathbf{\Omega}$. Every algebraic expression within our algebraic system formed by geometric figures chosen from the $\mathbf{G}_i$'s corresponds to a floorplan. These algebraic expressions can be represented by Polish expressions. The method of simulated annealing is then used to search for an optimal floorplan among these Polish expressions.

3.1. Geometric Figures. We first introduce a way of representing the geometric figures in $\mathbf{\Omega}$. Let A be a geometric figure in $\mathbf{\Omega}$. The *orientation index* of A is defined to be 0 if A is rectangular, and is either 1, 2, 3, or 4 as shown in Figure 12 if A is L-shaped. It is clear that the dimensions of A are completely specified by the length of its four outermost boundary edges. Let x_1 and x_2, $x_1 \geq x_2$, be the lengths of the two outermost horizontal edges. Let y_1 and y_2, $y_1 \geq y_2$, be the lengths of the two outermost vertical edges. Let s be the orientation index of A. Then A can be represented by the 5-tuples (x_1, x_2, y_1, y_2, s) (see Figure 13). Consequently, the set of all rectangular geometric figures can be represented by $\mathbf{\Omega}_1 = \{(x_1, x_2, y_1, y_2, 0) | x_1 = x_2, y_1 = y_2\}$, and the set of all L-shaped geometric figures can be represented by

$$\mathbf{\Omega}_2 = \{(x_1, x_2, y_1, y_2, s) | x_1 \geq x_2, y_1 \geq y_2, (x_1 - x_2)(y_1 - y_2) > 0, \text{ and } s \in \{1, 2, 3, 4\}\}.$$

The condition $(x_1 - x_2)(y_1 - y_2) > 0$ excludes the possibility of the geometric figure being rectangular. Hence there is a 1-1 correspondence between $\mathbf{\Omega}$ and the set $\mathbf{\Omega}_1 \cup \mathbf{\Omega}_2$.

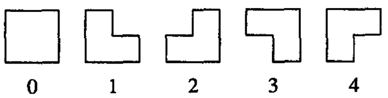


Fig. 12. Orientation indices of geometric figures in $\mathbf{\Omega}$.

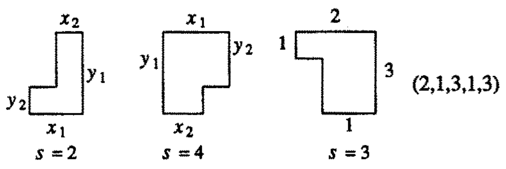


Fig. 13. Representation of geometric figures by 5-tuples (x_1, x_2, y_1, y_2, s).

3.2. The Operators. As was pointed out in Section 2.2, slicing floorplans can be obtained by recursively combining rectangles to form larger rectangles by means of two binary operators $*$ and $+$. For two rectangles A and B, $A+B$ and $A*B$ are the rectangles obtained by placing B on top of A, and B to the right of A, respectively. Since we are now dealing with both rectangular and L-shaped geometric figures, a natural extension is to define operators that combine rectangular and L-shaped geometric figures to form larger rectangular and L-shaped geometric figures. We define a unary operator $\neg$, and four binary operators $*_1$, $*_2$, $+_1$, and $+_2$ that operate on the geometric figures in $\mathbf{\Omega}$.

Figure 14 shows the definition of the unary operator $\neg$ which is a function from $\mathbf{\Omega}$ to $\mathbf{\Omega}$. For $A \in \mathbf{\Omega}$, $\neg A$ is defined to be the smallest bounding rectangle of A. We shall refer $\neg A$ as the *completion* of A. More precisely, we have $\neg(x_1, x_2, y_1, y_2, s) = (x_1, x_1, y_1, y_1, 0)$. The unary operator $\neg$ provides the possibility of placing an L-shaped geometric figure in a rectangular region. The four binary operators, which are functions from $\mathbf{\Omega} \times \mathbf{\Omega}$ to $\mathbf{\Omega}$, represent different ways of putting two geometric figures in $\mathbf{\Omega}$ together as compactly as possible, to form a larger geometric figure in $\mathbf{\Omega}$. Let $A, B \in \mathbf{\Omega}$. Then $A *_1 B$ and $A *_2 B$ are geometric figures obtained by putting A and B next to each other horizontally with B placed to the right of A, and $A +_1 B$ and $A +_2 B$ are geometric figures obtained by putting A and B next to each other vertically with B placed on top of A. The operators are defined in such a way that the rules for combining the geometric figures are completely determined by the orientation indices of the two geometric

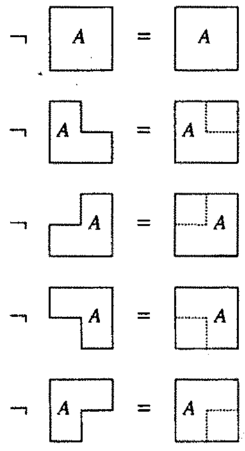


Fig. 14. Unary operator.

figures involved. The exact dimensions and orientation index of the new geometric figure are determined by the dimensions of the two geometric figures involved such that packing is as compact as possible. Hence, each binary operator can be completely described by its actions on the 25 possible combinations of the orientation indices of the two operands (five possible orientation indices for A and five possible orientation indices for B). Thus, for the four binary operators, there are altogether 100 possibilities. Figure 15 shows 20 of the 100 possible ways of combining two geometric figures based solely on their orientation indices. (In Figure 15 each circle at a T-intersection point corresponds to three possible outcomes for the resulting geometric figure. Figure 16 illustrates this notation.) For a complete description of the binary operators, see [Wo 87]. We now give three examples to illustrate these binary operations.

EXAMPLE 1. We consider here the case of $A *_1 B$ when the orientation indices of A and B are 3 and 1, respectively (see Figure 17(a)). Let $A=(a_1, a_2, b_1, b_2, 3) \in \mathbf{\Omega}$ and $B=(c_1, c_2, d_1, d_2, 1) \in \mathbf{\Omega}$. We have $A *_1 B=(x_1, x_2, y_1, y_2, s)$ where the values of x_1, x_2, y_1, y_2, and s are determined by the following procedure:

```
$x_1 \leftarrow a_1 + c_1;$
$x_2 \leftarrow a_2 + c_1;$
$y_1 \leftarrow \max(b_1, d_1);$
$y_2 \leftarrow b_2;$
 $s \leftarrow 3.$
```

For example, we have $(2, 1, 2, 1, 3) *_1 (2, 1.5, 2.5, 2, 1) = (4, 3, 2.5, 1, 3)$. (See Figure 17(b) for a pictorial illustration of this computation.)

EXAMPLE 2. We consider here the case of $A +_2 B$ when the orientation indices of A and B are 1 and 0, respectively (see Figure 18(a)). Let $A=(a_1, a_2, b_1, b_2, 1) \in \mathbf{\Omega}$ and $B=(c_1, c_1, d_1, d_1, 0) \in \mathbf{\Omega}$. We have $A +_2 B=(x_1, x_2, y_1, y_2, s)$ where the values of x_1, x_2, y_1, y_2, and s are determined by the following procedure:

```
$x_1 \leftarrow \max(a_1, a_2 + c_1);$
$y_1 \leftarrow \max(b_1, b_2 + d_1);$
**if** $(y_1 > b_1)$ **then**
  **begin**
    $x_2 \leftarrow c_1;$
    $y_2 \leftarrow b_1;$
    $s \leftarrow 2;$
  **end**
**else**
  **begin**
    $x_2 \leftarrow a_2;$
    $y_2 \leftarrow b_2 + d_1;$
    $s \leftarrow 1;$
  **end**;
**if** $(x_1 = x_2)$ or $(y_1 = y_2)$ **then**
  $(x_1, x_2, y_1, y_2, s) \leftarrow (x_1, x_1, y_1, y_1, 0);$
```

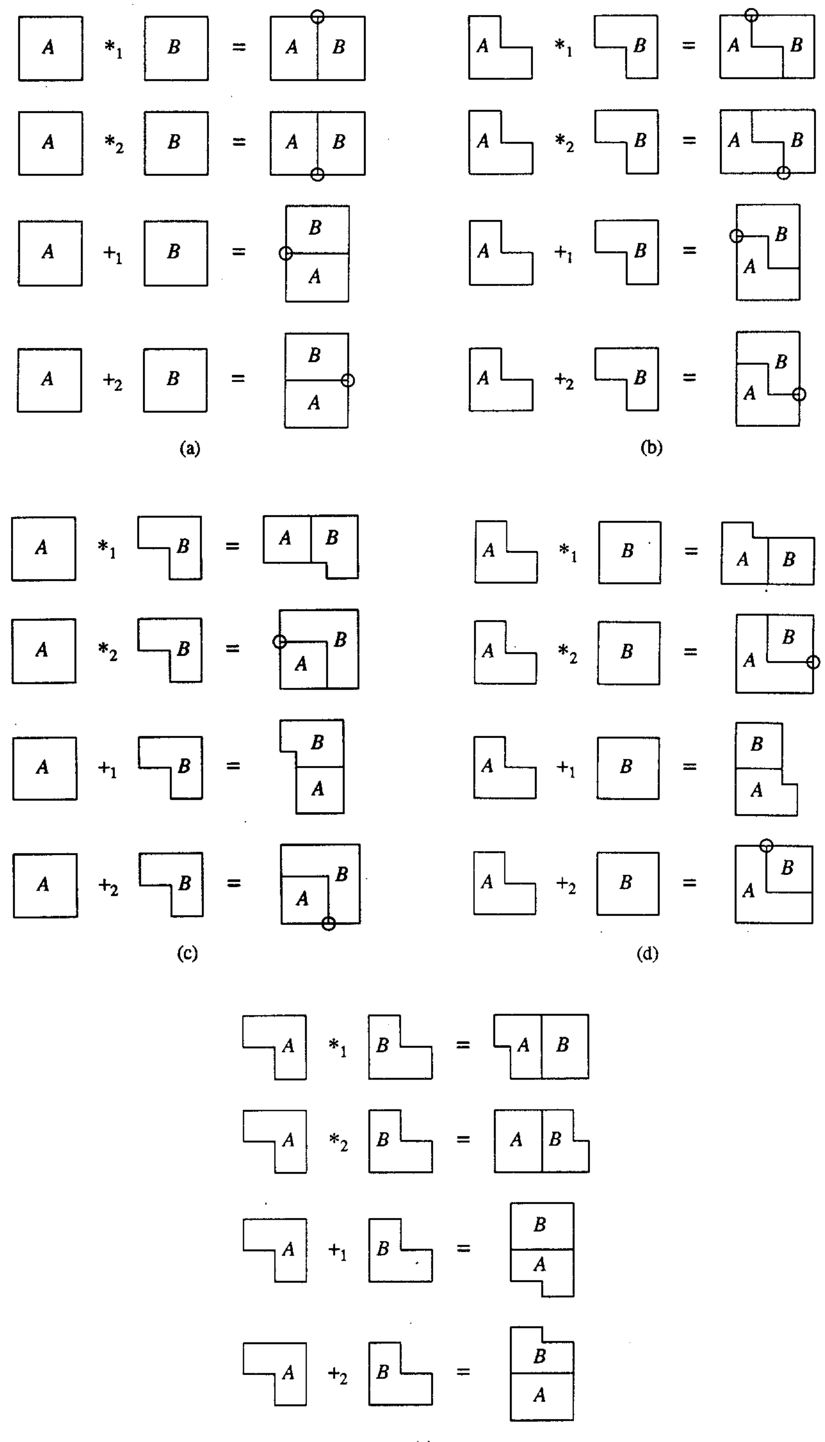


Fig. 15. Binary operators (20 out of 100 cases).

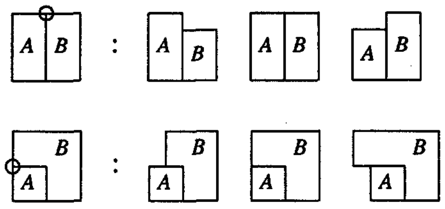
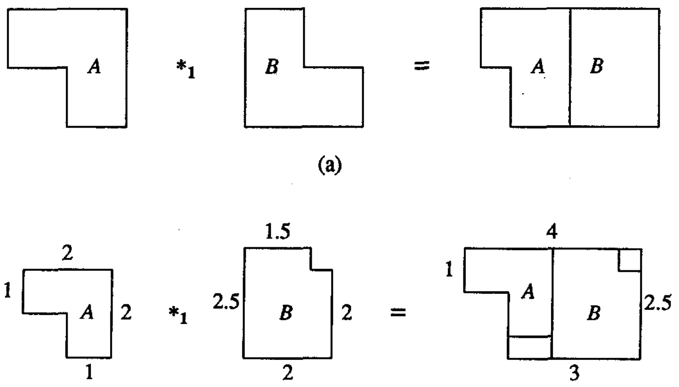


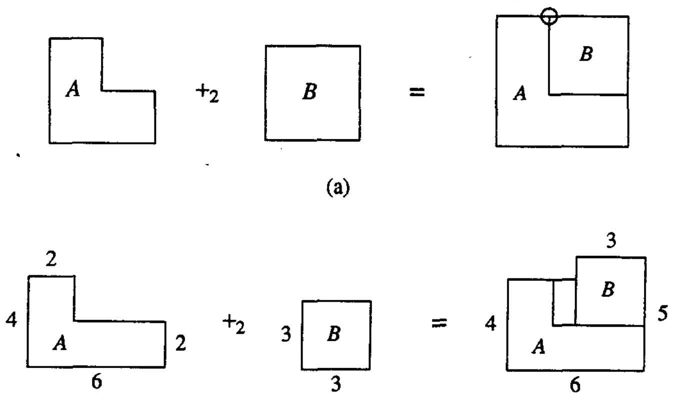
Fig. 16. Circle notation representing three possible outcomes of combining two figures.



$(2, 1, 2, 1, 3) *_1 (2, 1.5, 2.5, 2, 1) = (4, 3, 2.5, 1, 3)$

(b)

Fig. 17. An example of a binary operation.



$(6, 2, 4, 2, 1) +_2 (3, 3, 3, 3, 0) = (6, 3, 5, 4, 2)$

(b)

Fig. 18. An example of a binary operation.

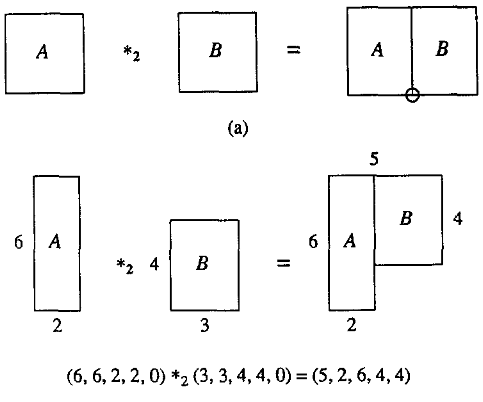


Fig. 19. An example of a binary operation.

The last **if-then** statement is for the case in which the resulting geometric figure is a rectangle because of the dimensions of A and B. (See Figure 18(b) for a pictorial illustration of the computation $(6, 2, 4, 1) +_2 (3, 3, 3, 3, 0) = (6, 3, 5, 4, 2)$.)

EXAMPLE 3. We consider here the case of $A *_2 B$ when the orientation indices of A and B are both 0 (see Figure 19(a)). Let $A = (a_1, a_1, b_1, b_1, 0) \in \mathbf{\Omega}$ and $B = (c_1, c_1, d_1, d_1, 0) \in \mathbf{\Omega}$. We have $A *_2 B = (x_1, x_2, y_1, y_2, s)$ where the values of x_1, x_2, y_1, y_2, and s are determined by the following procedure:

```
$x_1 \leftarrow a_1 + c_1$;
$y_1 \leftarrow \max(b_1, d_1)$;
if ($y_1 > d_1$) then
  begin
    $x_2 \leftarrow a_1$;
    $y_2 \leftarrow d_1$;
    $s \leftarrow 4$;
  end
else
  begin
    $x_2 \leftarrow c_1$;
    $y_2 \leftarrow b_1$;
    $s \leftarrow 3$;
  end;
if ($x_1 = x_2$) or ($y_1 = y_2$) then
  $(x_1, x_2, y_1, y_2, s) \leftarrow (x_1, x_1, y_1, y_1, 0)$;
```

As in the last example, the last **if-then** statement is for the case in which $A *_2 B$ is a rectangle. (See Figure 19(b) for a pictorial illustration of the computation $(6, 6, 2, 2, 0) *_2 (3, 3, 4, 4, 0) = (5, 2, 6, 4, 4)$.)

It should also be noted that the operators constitute a *complete* set in the sense that we can generate all minimally compacted combinations of geometric figures. More precisely, we have the following theorem.

THEOREM 5. *Let f: $\Omega \times \Omega \rightarrow \Omega$ be any function that combines two geometric figures to yield a larger geometric figure. For any A, $B \in \Omega$, there exists $C \in \Omega$ such that C is small enough to be placed inside $f(A, B)$, and C is of the form $A' \cdot B'$ or $B' \cdot A'$, where A' is either A or $\neg A$, B' is either B or $\neg B$, and $\cdot \in \{*_1, *_2, +_1, +_2\}$.*

PROOF. We assume that the orientation index of $f(A, B)$ is 1. The proof for the other cases are similar. $f(A, B)$ is an L-shaped region containing two nonoverlapping geometric figures A and B. The first step of our proof is to complete both A and B as much as possible. If $\neg A$ remains inside $f(A, B)$ and does not intersect B, we let A' be $\neg A$, otherwise we let A' be A. Similarly, if $\neg B$ remains inside $f(A, B)$ and does not intersect A', we let B' be $\neg B$, otherwise we let B' be B. There are several cases to be considered.

Case 1. Both A' and B' are rectangles. Suppose there is a vertical line separating A' and B'. Without loss of generality we may assume that A' is to the left of B'. C is the first geometric figure shown in Figure 20(a). If no such vertical line exists, there must be a horizontal line separating A' and B'. Again, we may assume

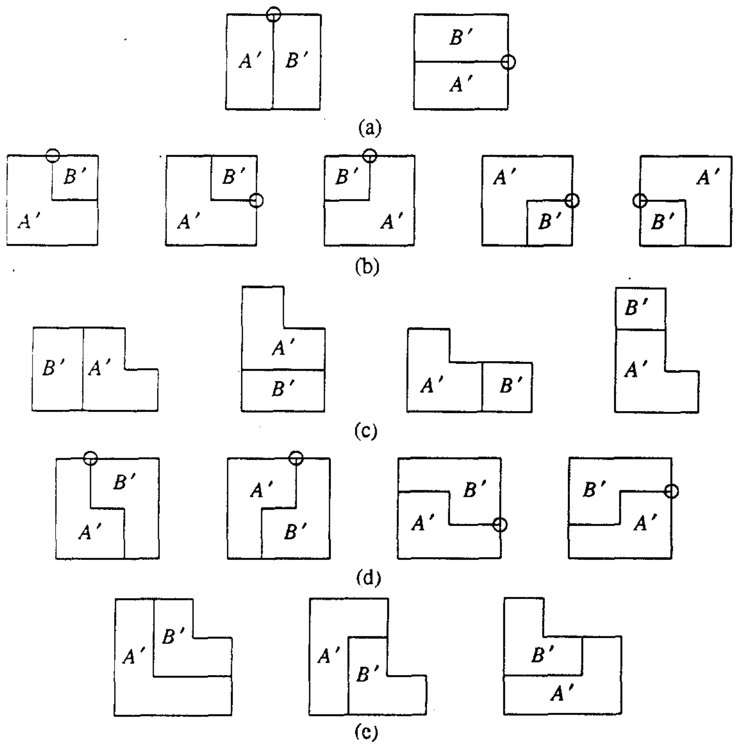


Fig. 20. Possible choices for C (proof of Theorem 5).

that A' is below B'. C is the second geometric figure shown in Figure 20(a). (Note that because the orientation index of $f(A, B)$ is 1, we need not consider the other two combining rules where the circle symbol appears either on the left side or at the bottom. This also applies to the other cases.)

Case 2. One of A' and B' is a rectangle, and the other is L-shaped. We may assume that A' is L-shaped. If $\neg A'$ intersects B', C can be chosen from the figures shown in Figure 20(b). Otherwise, we have $\neg A'$ intersects the outside of $f(A, B)$ and C can be chosen from the figures shown in Figure 20(c).

Case 3. Both A' and B' are L-shaped. Suppose $\neg A'$ intersects B' and $\neg B'$ intersects A'. We may assume that A' is either to the left of B' or below B'. C can be chosen from the figures shown in Figure 20(d). If the completions of A' and B' are not mutual intersecting, we may assume that $\neg A'$ intersects B', $\neg B'$ intersects the outside of $f(A, B)$, and $\neg B'$ does not intersect A'. In this case, C can be chosen from the figures shown in Figure 20(e).

Note that the southwest corner of C is well defined except when C is given by the last geometric figure in Figure 20(b). In that case we use the southwest corner of $\neg C$ instead. Now if we place the southwest corner of C at the southwest corner of $f(A, B)$, C stays completely inside $f(A, B)$. The theorem follows from the fact that all the geometric figures in Figure 20 can be generated by our operators. □

Figure 21 shows three examples of $f(A, B)$ together with the corresponding smaller geometric figures generated by our operators.

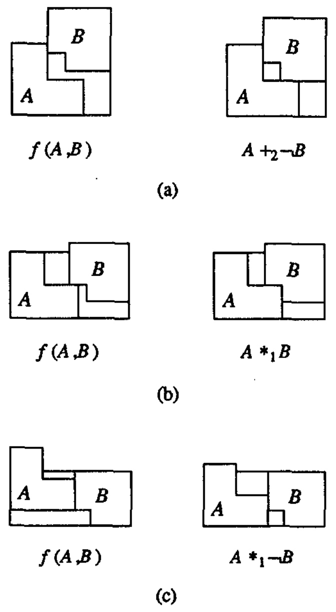


Fig. 21. Compact geometric figures generated by our operators.

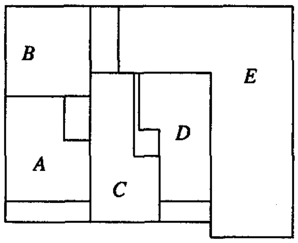


Fig. 22. Geometric figure corresponding to $(\neg A +_1 B) *_1 (\neg(C *_2 D) +_2 E)$.

3.3. Floorplan Representation. Consider the *algebraic system*

$$(\mathbf{\Omega}, *_1, *_2, +_1, +_2, \neg).$$

It is clear that an *algebraic expression* in $(\mathbf{\Omega}, *_1, *_2, +_1, +_2, \neg)$ represents a way to combine a set of geometric figures in $\mathbf{\Omega}$ to yield a resultant geometric figure in $\mathbf{\Omega}$. Consider as an example the algebraic expression $(\neg A +_1 B) *_1 (\neg(C *_2 D) +_2 E)$ with $A, B, C, D, E \in \mathbf{\Omega}$. (The order of precedence of the binary operations is specified by parentheses and we use the convention that unary operation has priority over binary operations.) Figure 22 shows the geometric figure corresponding to the expression.

Similar to the case discussed in Section 2, an algebraic expression γ can be represented by a tree **T**. The internal nodes of **T** correspond to the operators in γ and the leaf nodes correspond to the geometric figures in γ. Let v be an internal node of **T**. If v is a binary operator, then v has two subtrees. These subtrees correspond to the two geometric figures that v combines. If v is the unary operator $\neg$, then v has one subtree which corresponds to the geometric figures that v operates on. Figure 23 shows the tree representation of $(\neg A +_1 B) *_1 (\neg(C *_2 D) +_2 E)$. If we traverse the tree **T** in postorder, we obtain a *Polish expression* representation of γ. For example, the Polish expression obtained from $(\neg A +_1 B) *_1 (\neg(C *_2 D) +_2 E)$ is $A \neg B +_1 CD *_2 \neg E +_2 *_1$.

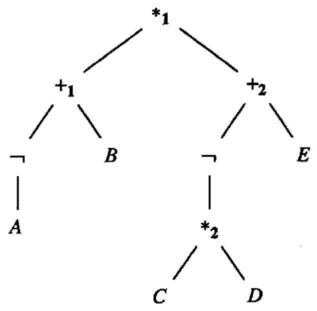


Fig. 23. Tree representation for $(\neg A +_1 B) *_1 (\neg(C *_2 D) +_2 E)$.

Let $\mathbf{E}$ be the set of all algebraic expressions with *operands*[5] $m_1, m_2, \ldots, m_n$, where $m_i \in \mathbf{G}_i$, $1 \leq i \leq n$. The set E represents different way of compactly combining the given set of modules. Clearly, each algebraic expression $\boldsymbol{\alpha}$ in $\mathbf{E}$ represents a floorplan for the set of given modules. (More precisely, $\neg\boldsymbol{\alpha}$, the completion of $\boldsymbol{\alpha}$, represents a floorplan.) Furthermore, the regions in the floorplan are either rectangular or L-shaped. Let $\mathbf{E_P}$ be the set of Polish expressions corresponding to the algebraic expressions in $\mathbf{E}$. Since there is a 1-1 correspondence between $\mathbf{E_P}$ and $\mathbf{E}$, we may assume that the set of floorplans under consideration are given by $\mathbf{E_P}$. We shall search for an optimal floorplan among all the Polish expressions in $\mathbf{E_P}$.

3.4. The Algorithm. Let $\boldsymbol{\alpha}$ be a Polish expression and $\mathbf{f}_{\boldsymbol{\alpha}}$ be the floorplan represented by $\boldsymbol{\alpha}$. Let $A(\boldsymbol{\alpha})$ be the area of $\mathbf{f}_{\boldsymbol{\alpha}}$. In other words, if $(x_1, x_2, y_1, y_2, s) \in \boldsymbol{\Omega}$ is the geometric figure corresponding to $\boldsymbol{\alpha}$, $A(\boldsymbol{\alpha}) = x_1 y_1$. (In the case where there are two numbers p and q specifying the allowed range of aspect ratio of the final chip, $A(\boldsymbol{\alpha})$ is defined as the area of the smallest rectangle, with aspect ratio between p and q, containing (x_1, x_2, y_1, y_2, s).) Let $W(\boldsymbol{\alpha})$ be the total wire length of $\mathbf{f}_{\boldsymbol{\alpha}}$. The cost function is $\Psi(\boldsymbol{\alpha}) = A(\boldsymbol{\alpha}) + \lambda W(\boldsymbol{\alpha})$, where λ is a given constant that controls the relative importance of area and wire length.

Our floorplan design algorithm employs the method of simulated annealing to search for a Polish expression $\boldsymbol{\alpha}$ in $\mathbf{E_P}$ such that $\Psi(\boldsymbol{\alpha})$ is minimized. Let $\boldsymbol{\alpha} \in \mathbf{E_P}$ be a Polish expression. It is clear that $\boldsymbol{\alpha}$ is of the form $\alpha_1\beta_1\alpha_2\beta_2 \cdots \alpha_{2n-1}\beta_{2n-1}$ where n of the α_i's are geometric figures for the n modules (operands), the other n α_i's are binary operators, and each β is either the unary operator $\neg$ or the empty string. We define four types of moves that can be used to modify $\boldsymbol{\alpha}$. Two Polish expressions $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ in $\mathbf{E_P}$ are said to be *neighbors* if one can be obtained from the other via one of these moves. The four types of moves are defined as follows:

M1. Modify α_i for some i.
M2. Modify β_i for some i.
M3. Swap two operands.
M4. Swap two adjacent operand and binary operator in $\alpha_1\alpha_2 \cdots \alpha_{2n-1}$.

We now give a more detailed description of the moves. For M1 moves, there are two cases corresponding to whether α_i is an operand or a binary operator. If α_i is an operand, we have $\alpha_i = A \in \mathbf{G}_k$ for some k. We set $\alpha_i \leftarrow A'$ where $A' \in \mathbf{G}_k$ and $A \neq A'$. This corresponds to selecting another instance for a module. (Note that if $|\mathbf{G}_k| = 1$, no modification is possible.) On the other hand, if α_i is a binary operator, we select a different binary operator for α_i. For M2 moves, we change β_i to the other element in $\{\varepsilon, \neg\}$. Note that in a Polish expression $\boldsymbol{\alpha}$, each operand represents a geometric figure for a module, and the subtree rooted at each operator represents a geometric figure for a "supermodule." We refer to these geometric figures as the geometric figures represented by the α_i's and the β_i's in $\boldsymbol{\alpha}$. Clearly,

[5] We shall refer to the elements in $\boldsymbol{\Omega}$ that appear in an algebraic expression as the *operands* of the expression.

M2 has no effect at all when α_i is a rectangle. Therefore, we only modify those β_i where α_i is of L-shape. For M3 moves, we swap two operands α_i and α_j. This corresponds to swapping two modules. Finally, M4 moves correspond to swapping α_i and α_{i+1} for some i when one of them is an operand and the other is a binary operator. Note that M1, M2, or M3 moves always result in another Polish expression in $\mathbf{E_P}$, but M4 moves may produce an expression which is not a Polish expression. We only use those M4 moves that produce a valid Polish expression. As in Section 2, the four types of moves are sufficient to ensure that it is possible to transform any Polish expression into any other via a sequence of moves. Again, there is a sequence of $O(n^2)$ neighboring Polish expressions between any two Polish expressions.

We use a temperature schedule of the form $T_k = rT_{k-1}$ where r is usually between 0.8 and 0.9. In the current implementation, the stopping criteria are: (1) the number of accepted moves in that temperature is very small; or (2) the average cost remains unchanged for three consecutive temperatures.

We should point out that throughout the entire annealing process, we need to compute the cost of an algebraic expression many times. Since each move is only a minor modification of a Polish expression, we should avoid recomputations as much as possible. Recall that in a Polish expression $\boldsymbol{\alpha}$, each operand represents a geometric figure for a module, and the subtree rooted at each operator represents a geometric figure for a "supermodule." Computation of the cost of $\boldsymbol{\alpha}$ amounts to determining all these geometric figures. Fortunately, after every move, many of these geometric figures remain unchanged. The following theorem characterizes the set of geometric figures that need to be recomputed. The proof of the theorem is very similar to that of Theorem 4 in Section 2.

THEOREM 6. *Let $\boldsymbol{\alpha}'$ be the expression obtained from $\boldsymbol{\alpha}$ after a move. Let $\mathbf{T}_{\alpha}$ and $\mathbf{T}_{\alpha'}$ be the trees corresponding to $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$, respectively. The geometric figures that need to be recomputed correspond to subtrees rooted at operators that lie on one or two paths in each of $\mathbf{T}_{\alpha}$ and $\mathbf{T}_{\alpha'}$.*

Finally, we note that for the case in which the given modules are rectangular, our algorithm is capable of producing a nonslicing floorplan. As an example, we note that the algebraic expression $((B +_2 A) +_2 C) *_1 (D +_1 E)$ corresponds to the nonslicing floorplan shown in Figure 24.

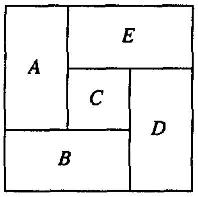


Fig. 24. A nonslicing floorplan.

Table 4

Problem	n	δ	A_L/A_S
Q1	20	0.81	0.85
Q2	20	0.84	0.89
Q3	25	0.85	0.88
Q4	40	0.83	0.86
Q5	25	0.66	0.75

3.5. Experimental Results. We have implemented our floorplan design algorithm in PASCAL on a PYRAMID computer. We compared our results on several test problems with those obtained by the floorplan design algorithm in Part 1. The algorithm in Part 1 was designed to handle only rectangular modules and only produces slicing floorplans. In order to use that algorithm, we replaced each L-shaped module by its smallest bounding rectangle when the algorithm is tested. The results are summarized in Table 4 where n is the number of modules and δ measures the amount of dead space introduced by replacing all L-shaped modules by their bounding restangles. More precisely, if A is the total area of all the modules and A' is the total area of the bounding rectangles of all the modules, $\delta = A/A'$. A_S is the area of the floorplan produced by the algorithm in Part 1 and A_L is the area of the floorplan produced by the algorithm described in this section. In our test problems the values of A_S/A_L and δ are quite close. This indicates that our algorithm can take full advantage of the L-shaped modules to reduce total chip area. Figure 25 shows the final floorplan for problem Q4 which contains 40 modules. The execution time for this problem was about 10 CPU minutes.

Finally, Table 5 shows comparison between the two algorithms in the case where all given modules are rectangular. We performed the experiments on four problems which were obtained by changing the shape flexibilities of the modules of a 20-modules problem. All modules have the same maximum allowed aspect ratio which is referred to as module ratio in the table. We required the final chip to be a square. The columns D_S and D_L denote the percentage of dead spaces in the floorplans obtained by the algorithm in Part 1 and the algorithm in Part 2, respectively. Note that, for the four tested problems, both D_S and D_L increase as the module ratio increases. This is to be expected because we can pack modules closer together if the modules are more flexible. Also, for module ratio = 1.0, 1.25, and 1.5, $D_S > D_L$. This indicates that the algorithm in Part 2 obtained smaller chips by generating nonslicing floorplans. For module ratio = 2.0, D_S was smaller than D_L. There are two reasons for this result: (1) slicing floorplans are very good solutions when modules are very flexible (e.g., module ratio = 2); and (2) the current implementation of the algorithm in Part 2 can only consider a finite set of shape alternatives for each module and hence did not consider all possible shapes alternatives when the algorithm was tested. In order to obtain a better result for the case module ratio = 2, we should use a larger (finite) set of shape alternatives.

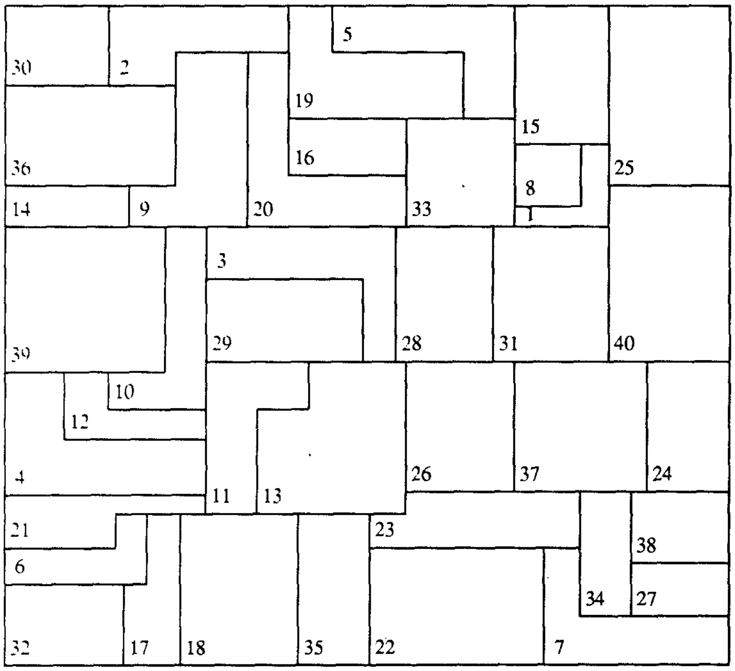


Fig. 25. Final floorplan for problem Q4.

4. Concluding Remarks. We have presented in this paper two algorithms for the floorplan design problem. The algorithms are very similar in spirit. They both use Polish expressions to represent floorplans, employ the same search method of simulated annealing, and use similar ways to modify a floorplan locally. The first algorithm is designed for the case where all the modules are rectangular. All floorplans produced by this algorithm are slicing floorplans. The second algorithm is designed for the case where the modules are either rectangular or L-shaped. This algorithm is capable of producing a nonslicing floorplan when all the modules are rectangular. Our algorithms consider simultaneously the interconnection information as well as the area and shape information. Experimental results indicate that our algorithms perform well for many test problems.

Table 5

Module ratio	D_S (%)	D_L (%)
1.0	25.0	18.0
1.25	21.5	11.7
1.5	12.6	8.0
2.0	1.1	4.7

References

[Ah 74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison Wesley, Reading, MA, 1974.

[He 82] W. R. Heller, G. Sorkin, and K. Maling, The Planar Package for System Designers, *Proc. 19th ACM/IEEE Design Automation Conf.* (1982), pp. 253-260.

[Ki 83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by Simulated Annealing, *Science*, **220** (1983), 671-680.

[La 80] U. Lauther, A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation, *Journal of Digital Systems*, **IV** (1) (1980), 21-34.

[LD 85] D. P. LaPotin and S. W. Director, Mason: A Global Floor-Planning Tool, *Proc. Intl. Conf. on Computer-Aided Design* (1985), pp. 143-145.

[Ma 82] K. Maling, S. H. Mueller, and W. R. Heller, On Finding Most Optimal Rectangular Package Plans, *Proc. 19th ACM/IEEE Design Automation Conf.* (1982), pp. 663-670.

[OG 84] R. H. J. M. Otten and L. P. P. P. van Ginneken, Floorplan Design using Simulated Annealing, *Proc. Intl. Conf. on Computer-Aided Design* (1984), pp. 96-98.

[Ot 82] R. H. J. M. Otten, Automatic Floorplan Design, *Proc. 19th ACM/IEEE Design Automation Conf.* (1982), pp. 261-267.

[Ot 83] R. H. J. M. Otten, Efficient Floorplan Optimization, *Proc. Intl. Conf. on Computer Design* (1983), pp. 499-502.

[PC 85] B. Preas and C. S. Chow, Placement and Routing Algorithms for Topological Integrated Circuit Layout, *Proc. Intl. Symp. on Circuits and Systems* (1985), pp. 17-20.

[PV 79] B. Preas and W. M. VanCleemput, Placement Algorithms for Arbitrary Shaped Blocks, *Proc. 16th ACM/IEEE Design Automation Conf.* (1979), pp. 474-480.

[SD 85] L. Sha and R. W. Dutton, An Analytical Algorithm for Placement of Arbitrary Sized Rectangular Blocks, *Proc. 22nd ACM/IEEE Design Automation Conf.* (1985), pp. 602-608.

[SS 85] C. Sechen and A. Sangiovanni-Vincentelli, The Timberwolf Placement and Routing Package, *IEEE Journal of Solid-State Circuits*, **20** (2) (1985), 510-522.

[St 83] L. Stockmeyer, Optimal Orientations of Cells in Slicing Floorplan Designs, *Information and Control*, **59** (1983), 91-101.

[WL 86] D. F. Wong and C. L. Liu, A New Algorithm for Floorplan Design, *Proc. 23rd ACM/IEEE Design Automation Conf.* (1986), pp. 101-107.

[Wo 87] D. F. Wong, Algorithmic Aspects of VLSI Circuit Layout, Ph.D. Thesis, University of Illinois at Urbana-Champaign, January, 1987.

[WW 86] L. S. Woo, C. K. Wong, and D. T. Tang, Pioneer: A Macro-Based Floor-Planning Design System, *VLSI Systems Design* (1986), pp. 32-43.