A NEW ALGORITHM FOR FLOORPLAN DESIGN.[1]

D. F. Wong and C. L. Liu

Department of Computer Science
University of Illinois at Urbana–Champaign
1304 W. Springfield, Urbana, IL 61801

Abstract

We present in this paper a new algorithm for floorplan design using the method of simulated annealing. The major contributions of the paper are: 1. A new representation of floorplans (normalized Polish expressions) which enables us to carry out the neighborhood search effectively. 2. A simultaneous minimization of area and total interconnection length in the final solution. Experimental results indicate that the algorithm performs well in many test problems.

1. Introduction

Floorplan design ([He 82], [Ma 82], [Ot 82]) is one of the most important problems in VLSI circuit layout. It is a generalization of the classical module placement problem ([PV 79], [La 80]). Both problems are concerned with the placement of rectangular modules of arbitrary size and dimensions such that the total area occupied by the modules and the interconnections is minimum. In the module placement problem, the dimensions of the modules are fixed. On the other hand, in the floorplan design problem the modules may assume any shape permitted by its shape constraints (e.g. a range of possible aspect ratios). The flexibility in the shape of the modules represents the designer's freedom to manipulate the modules' internal structure, and is hence a more accurate reflection of the steps carried out in the physical design process.

We are given a set of n modules named 1,2,...,n and a corresponding list of n triplets of numbers (A_1,r_1,s_1), (A_2,r_2,s_2), $\cdots$, (A_n,r_n,s_n). The triplet of numbers (A_i,r_i,s_i), with $r_i \le s_i$, specifies the area and the shape constraints for the module i. In fact, if we let w_i be the width of module i 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 $i \in S_1$,

(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 $i \in S_2$.

Module i is a *rigid module* if $r_i = s_i$, otherwise it is a *flexible module*. We are given two set S_1 and S_2 such that $S_1 \cup S_2 = \{1,2,...,n\}$. The set S_1 specifies the set of modules with fixed orientation, and the set S_2 specifies the set of modules with free orientations. The range of the aspect ratio of the final chip that contain the n modules is specified by two numbers (p,q), $p \le q$. We are also given an $n \times n$ interconnection matrix $C = (c_{ij})_{n\times n}$, with $0 \le c_{ij} \le 1$, $1 \le i,j \le n$, which provides information on the wiring density between each pair of modules.

[1] This work was partially supported by the National Science Foundation under grant DCR 84-04239 and by a grant from the General Electric Company.

A *feasible solution* of the floorplan design problem consists of an enveloping rectangle R subdivided by horizontal and vertical line segments into n nonoverlapping rectangles labeled 1,2,...,n. The aspect ratio of R is between p and q, and for each i, rectangle i (with dimensions (x_i,y_i)) is large enough to accommodate module i. (*i.e.* $x_i \ge w_i$ and $y_i \ge h_i$ where w_i and h_i satisfy the above three conditions.)

In this paper, we present an algorithm that produces a feasible solution to the floorplan design problem which simultaneously minimizes the area of the chip and a proximity measure based on a total wire length estimation among the modules. The technique of simulated annealing ([Ki 83],[Le 85]) is employed to solve the problem.

There are several advantages of our algorithm over existing algorithms for the solution of the floorplan design problem (*e.g.* [Ma 82], [Ot 82a]). Some of the existing algorithms derive the final solution in two stages. They first determine the relative positions of the modules on the chip using primarily interconnection information, then they use the area and shape information to minimize the area of the chip. Whereas our algorithm considers simultaneously the interconnection information as well as the area and shape information. Existing algorithms that employ the technique of simulated annealing either use a representation that leads to an unnecessarily large number of states and thus ultimately a slow rate of convergence ([SS 85]), or apply the technique only at a particular stage of a heuristic floorplan design algorithm ([OG 84]). Our algorithm is based on a new representation of floorplans called *normalized Polish expressions* which enables us to speed up the search procedure significantly.

2. Definitions

A *rectangle dissection* is a subdivision of a given rectangle by horizontal and vertical line segments into a finite number of non-overlapping rectangles. The non-overlapping rectangles are called *basic rectangles*. By *cutting* a rectangle, we mean to divide the rectangle into two rectangles by a vertical or horizontal line. A *slicing structure* is a rectangle dissection that can be obtained by recursively cutting rectangles into smaller rectangles (see Fig.1a). The hierarchical structure of a slicing structure can be described by an oriented rooted binary tree, called a *slicing tree* (see Fig.1b). 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 when the slicing structure has n basic rectangles. A *skewed slicing tree* is a slicing tree in which no node and its right son has the same label in $\{*,+\}$. (See Fig.2.)

A binary sequence $b_1b_2 \cdots b_m$ is a *balloting sequence* iff for any k, $1 \leq k \leq m$, the number of 0 's in $b_1 \cdots b_k$ is less than the number of the 1 's in $b_1 \cdots b_k$. Let σ be a function $\sigma : \{1,2,...,n,*,+\} \rightarrow \{0,1\}$ 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 $\{1,2,...,n,*,+\}$ is a *Polish expression* of length $2n-1$ iff

(1) every i appears exactly once in the sequence, $1 \leq i \leq 2n-1$,

(2) $\sigma(\alpha_1)\sigma(\alpha_2) \cdots \sigma(\alpha_{2n-1})$ is a balloting sequence.[2]

A Polish expression $\alpha_1\alpha_2 \cdots \alpha_{2n-1}$ is said to be *normalized* iff there is no consecutive $*$'s or $+$'s in the sequence. (*e.g.* 1 2 + 4 3 * + is a normalized Polish expression.)

3. Solution Representation

In this paper, we restrict our search to floorplans that are slicing structures. Otten ([Ot 82b]) pointed out that slicing structures have several advantages over general rectangle dissections in floorplan design. It is also known that slicing structures are computationally easier to handle ([St 83],[Ot 83]). Slicing structures can be represented by either series-parallel graphs ([Ot 82]) or slicing trees ([Ot 83]). In this section, we introduce a new representation of slicing structures that is most suitable for the technique of simulated annealing. For the sake of brevity, we omit all the proofs of the lemmas and theorems in the rest of this paper.

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

In general, there might be two or more Polish expressions (slicing trees) that correspond to a given slicing structure (see Fig.3). The number of Polish expressions corresponding to a slicing structure can vary from slicing structure to slicing structure. This makes Polish expressions an undesirable choice for representation of solutions in a simulated annealing setting for the following reasons: 1. There is an unnecessary increase in the number of states. 2. The set of slicing structures is unevenly distributed over the set of Polish expressions, which might lead to unintentional and undesirable biases toward some slicing structures. The next theorem completely resolves the multiple representation problem. The theorem follows from the next lemma and the observation that given any slicing structure, it can be described by a unique skewed slicing tree by performing the cuts always from right to left and from top to bottom. Hence, we shall use the set of normalized Polish expressions as the solution space in our simulated annealing algorithm.

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

[2] $\alpha_1\alpha_2 \cdots \alpha_{2n-1}$ is said to have the balloting property if condition (2) is satisfied.

THEOREM. 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 is essentially a *top down* description of a slicing structure. It specifies how a given rectangle is cut into smaller rectangles by horizontal and vertical cutting lines. On the other hand, a slicing structures can also be described by recursively combining smaller slicing structures. Indeed, we can view a Polish expression as a *bottom up* description of a slicing structure. In fact, we can interpret the symbols $*$ and $+$ as two binary operators between slicing structures. If A and B are slicing structures, we can interpret $A+B$ and $A*B$ as the resulting slicing structures obtained by placing B on top of A, and B to the right of A, respectively as shown in Fig.4. An inorder traversal ([Ah 74]) of the slicing tree result in an "arithmetic expression" with $*$ and $+$ as the operators, and the basic rectangles as operands. (See Fig.5.) This expression specifies how to build the final slicing structure from smaller ones. The Polish expression in fact is the Polish postfix notation for this "arithmetic expression". From now on, we shall refer the elements in $\{1,2,...,n\}$ as *operands*, and the elements in $\{*,+\}$ as *operators*.

4. Neighborhood Structure

A sequence $b_1b_2 \cdots b_k$ of k operators is a called a *chain of length* k iff $b_i \neq b_{i+1}$, $1 \leq i \leq k-1$. 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 : $*+*+* \cdots$ 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 $\alpha = \alpha_1\alpha_2 \cdots \alpha_{2n-1}$ be a normalized Polish expression. Note that α can also be written as $c_0\pi_1c_1\pi_2c_2 \cdots c_{n-1}\pi_nc_n$, where $\pi_1,\pi_2,...,\pi_n$ is a permutation of $1,2,...,n$, the c_i's are chains (possibly of zero length), and $\sum_i (\textit{length of } c_i) = n-1$ (see Fig.6).

Two operands in α 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 that can be used to modify a given normalized Polish expression.

M1. Swap two adjacent operands.

M2. Complement some chain of nonzero length.

M3. Swap two adjacent operand and operator.

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 α, we can choose a neighbor of α by first randomly select a type of move and then randomly select a pair of adjacent elements or a chain according to the type of move selected. We also want to make sure that the move selected will also produce a normalized Polish expression. It is clear that M1 and M2 always produce a normalized Polish expression. This is not always the case for M3. In fact, M3 will change the sequence $\sigma(\alpha_1)\sigma(\alpha_2) \cdots \sigma(\alpha_{2n-1})$. It might produce a sequence that contains identical consecutive operators or violates the balloting property. To generate a move of type M3, we can repeatedly choose a pair of adjacent operator and operand, and then check whether swapping the operator and operand will lead to an expression that is not a normalized Polish expression. Since we have to make type M3 moves very often, efficient algorithm for checking the conditions is crucial. Fortunately, whether a given M3 move will result in a normalized Polish expression can be tested very rapidly ($O(1)$ time). Since it is trivial to determine whether a M3 move will introduce identical consecutive operators, we only need to show how to test efficiently whether a sequence possesses the balloting property. Let d_k denote the number of 0's in $\sigma(\alpha_1) \cdots \sigma(\alpha_k)$, $1 \leq k \leq 2n-1$. Consider a M3 move that swaps α_i and α_{i+1}. We only need to consider the case when α_i is an operand and α_{i+1} is an operator. In this case, it can be shown that swapping α_i and α_{i+1} will not violate the balloting property iff $2d_{i+1} < i$. Moreover, the d_i's can be updated easily after every move. Finally, it can be shown that the three types of moves are sufficient to ensure that it is possible to go from any normalized Polish expression to any other via a sequence of moves. (See Fig.7 for a pictorial demonstration of the three types of moves.)

5. The Cost Function

For a given feasible solution of the floorplan design problem, we shall use the area of the enveloping rectangle and the center to center total wire length estimation as the measures of the quality of floorplans. Let d_{ij} be the Manhattan distance between the centers of basic rectangles i and j, $1 \leq i,j \leq n$. The *total wire length* is given by $\sum\limits_{1 \leq i,j \leq n} c_{ij} d_{ij}$. In fact, our approach is quite flexible with respect to different kinds of quality measures (especially variations of proximity measures). We have arbitrarily chosen the frequently used total wire length in our experiments only to demonstrate the flexibility of our algorithm.

A *floorplan realization* of a given slicing structure is a feasible solution of the floorplan design problem where the basic rectangles in the slicing structure are the spaces allocated to the modules. In general, there are many floorplan realizations of a given slicing structure. The relative positions of the modules in different floorplan realizations are essentially fixed by the given slicing structure. Those floorplan realizations with smaller area, in general, tend to pull the modules closer together and hence achieve shorter wire length. For a given normalized Polish expression α, let S_α denote its corresponding slicing structure. We define the area measure A and the total wire length measure W of α to be the area and the total wire lengths of a minimum area floorplan realization of S_α. The cost function we use is of the following form :

$$\Psi = A + \lambda_w W$$

where λ_w control the relative importance of A and W.

6. Cost Computation

6.1. Bounding 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 $x_1 \leq x_2$, we must have $y_1 \geq y_2$. Γ is a *bounding curve* if it satisfies the following three conditions. 1. It is decreasing. 2. It lies completely in the first quadrant, *i.e.* $u > 0$ and $v > 0$ for all $(u,v) \in \Gamma$. 3. It partitions the first quadrant into two connected regions. The connected region which contains all the points (a,a) for very large a is called the *bounded area* with respect to the bounding curve. (see Fig.8.) Let Γ and Λ be two bounding curves. We define $\Gamma + \Lambda$ to be the curve $\{(u,v+w) \mid (u,v) \in \Gamma \text{ and } (u,w) \in \Lambda\}$ and $\Gamma * \Lambda$ to be the curve $\{(u+v,w) \mid (u,w) \in \Gamma \text{ and } (v,w) \in \Lambda\}$. $\Gamma + \Lambda$ ($\Gamma * \Lambda$) is obtained by adding the two curves Γ and Λ along the y-direction (x-direction). It is easy to see that $\Gamma + \Lambda$ and $\Gamma * \Lambda$ are also bounding curves. Moreover, they are piecewise linear if Γ and Λ are both piecewise linear. For piecewise linear bounding curves, there is an efficient algorithm to compute $\Gamma + \Lambda$ and $\Gamma * \Lambda$. A piecewise linear bounding curve is completely characterized by an ordered list of all the "corners" of the curve. To add two piecewise linear bounding curves (along either directions), we only have to add up the curves at the "corners".

6.2. Area Computation

For a given module i, let w_i and h_i denote the width and the height of the module, and let x_i and y_i denote the width and the height of the basic rectangle i. We must have $x_i \geq w_i$ and $y_i \geq h_i$. The bounding curves in Fig.9 correspond to different kinds of shape constraints for a module where the shaded regions are the bounded areas. A point in the bounded area corresponds to a basic rectangle that can accommodate a given module. (The x and y coordinates of the point are the horizontal and vertical dimensions of the basic rectangle.) Fig.9a and Fig.9b correspond to the case in which the module is rigid. Fig.9c and Fig.9d correspond 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 these figures, 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 T_α be the slicing tree corresponding to α. We can associate a bounding curve Γ_v with each node v in T_α. For each node v in T_α, the subtree rooted at v defines a slicing structure R_v. Let D_v be the set of all possible dimensions of R_v. Let Γ_v be the boundary of D_v. It can be shown that Γ_v is a bounding curve with D_v as the bounding area. For every three nodes u,v,w in T_α with u as the father of v and w, we note that Γ_u is either $\Gamma_v * \Gamma_w$ or $\Gamma_v + \Gamma_w$ depending on whether u is $*$ or $+$, respectively. Hence, all the Γ_v's can be computed by adding up the bounding curves of the basic rectangles (the leaves). We assume the bounding curves for the basic rectangles are all piecewise linear ([Ot 83]) so that we can compute efficiently all the Γ_v's. Once we have computed all the Γ_v's , we can compute the area measure A from Γ_r where r is the root of T_α. (Γ_r is the bounding curve for the slicing structure α.) Let (a_1,b_1) ((a_{l+1},b_{l+1})) be the point of intersection between Γ_r and the line $y = px$ ($y = qx$). Let (a_2,b_2), (a_3,b_3), ... ,(a_l,b_l) be all the "corners" of the curve that lies between the lines $y = px$ and $y = qx$. (*i.e.* $p \leq b_j/a_j \leq q$.) The dimensions of a minimum area realization are then given by the point (a_i,b_i) such that $a_i b_i$ is minimum. We have $A = a_i b_i$. Since we can have piecewise linear approximation of the bounding curves for the basic rectangles with arbitrary precision, it follows that we can determine A with arbitrary precision.

In our simulated annealing algorithm, we need to perform area computation many times. Since each move is only a local perturbation of the present Polish expression, many of the Γ_v's remain unchanged. The next theorem describes the set of Γ_v's that need to be recomputed for the new slicing structure. Indeed, the number of Γ_v's that need to be recomputed is usually quite small. Thus, the computation time for the area of each slicing structure examined in the annealing process can be significantly reduced. Before we state the next theorem, we need one more definition. A *fork* in a tree consists of two paths from two distinct nodes to the root as illustrated in Fig.10.

THEOREM. After any move of type M1, M2, or M3, the set of Γ_v's that are changed corresponds to a path or a fork in T_α.

We use Fig.11 to illustrate the above theorem. In the given example, the Polish expression is of the form $\cdots 17 + 16 \cdots * 6 \cdots 9 + * 8 \cdots$. We consider the following three moves : 1. Swapping 17 and 16 (M1). 2. Complement the chain $+ *$ (M2). 3. Swapping $*$ and 6 (M3). The forks (or paths) associated with these three moves are shown in Fig.11a, Fig.11b and Fig.11c.

We have designed an algorithm to compute the area measure A. Our algorithm only recompute those Γ_v's that are changed. It scans the Polish expression once, stores intermediate results on a stack, and add up bounding curves whenever it is necessary.

6.3. Wire Length Computation

Once we have determined the dimension of a minimum area floorplan realization, we can compute recursively the dimensions and the centers of all the basic rectangles using the Γ_v's. After each move, we only recompute those terms $c_{ij}d_{ij}$ where either one or both of the centers of the basic rectangle i and j changed. We can then subtract the old value of $c_{ij}d_{ij}$ and add the new value of $c_{ij}d_{ij}$ to W.

7. The Annealing Schedule

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

To determine the value of T_0, we can perform a sequence of random moves and compute the quantity Δ_{avg}, the average of change in cost in uphill moves. We should have $e^{-\Delta_{avg}/T_0} = P \simeq 1$ so that there will be a reasonable probability of acceptance at high temperatures. This suggests that $T = -\Delta_{avg} / \ln(P)$ is a good choice for T_0. Our experiments indicate that T should actually be smaller then T. We use $T_0 = \lambda_t T$ where $\lambda_t < 1$ is a constant used to compensate for initial solutions that are too far from optimal.

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 $N = O(\text{number of neighbors}) = O(n)$. 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.

8. Experimental Results and Discussions

We have implemented our simulated annealing algorithm. The program is written in PASCAL and is run on a PYRAMID. The experimental results are very encouraging. They are summarized in Tables 1, 2 and 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 0 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_w 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 the results for problems with the same number of modules (*i.e.* P2 & P3, P4 & P5, and P6 & 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, One 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_w 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_w, we obtain solutions in which the value of $A + \lambda_w W$ 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_w, 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. Fig.12 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_w to demonstrate the tradeoffs between area and wire length. As we increase λ_w from 0 to 3, we observe that A increases from 196.42 to 220.30 while W decreases from 152.1 to 93.96. We observe that those entries marked by a † are the values of the cost function. They should be the minimum values for the columns. 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 table 3. The column Chip Ratio in Table 3 is the maximum aspect ratio allowed for the final chip. The value of λ_w 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. Also, we observe that a Module Ratio of 2 gives enough flexibility for achieving close to optimal final area.

REFERENCES

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

[He 82] W. R. Heller, G. Sorkin and K. Maling, "The Planar Package For System Designers," *Proc. 19th D. A. Conf.*, (1982), 253-260.

[Ki 83] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing," *Science, Vol. 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, Vol. IV, Issue 1*, (1980), 21-34.

[Le 85] H. W. Leong, D. F. Wong and C. L. Liu, "A Simulated-Annealing Channel Router," *Proc. Intl. Conf.on Computer-Aided-Design, IEEE*, (1985).

[Ma 82] K. Maling, S. H. Mueller and W. R. Heller, "On Finding Most Optimal Rectangular Package Plans," *Proc. 19th D. A. Conf.*, (1982), 663-670.

[Ot 82a] R. H. J. M. Otten, "Automatic Floorplan Design," *Proc. 19th D. A. Conf.*, (1982), 261-267.

[Ot 82b] R. H. J. M. Otten, "Layout Structures," *Proc. IEEE Large Scale Systems Symp.* (1982).

[Ot 83] R. H. J. M. Otten, "Efficient Floorplan Optimization," *Proc. ICCD 83*, (1983), 499-502.

[OG 84] R. H. J. M. Otten, L. P. P. P. van Ginneken, "Floorplan Design using Simulated Annealing," *Proc. ICCAD 84*, (1984), 96-98.

[PV 79] B.T. Preas, W. M. VanCleemput, "Placement Algorithms for Arbitrary Shaped Blocks," *Proc. 16th D. A. Conf.*, (1979), 474-480.

[SS 85] C. Sechen, A. Sangiovanni-Vincentelli, "The Timberwolf Placement and Routing Package," *IEEE Journal of Solid-State Circuits, Vol. SC-20, No.2*, (1985), 510-522.

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

Problem	n	λ_w	Total Area	Random Feasible Solution		Initial S.A. Solution		Final S.A. Solution	
				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	202.3	202.17	80.49
p3	20	0	197.15	808.72	423.90	819.64	276.5	198.98	197.4
p4	25	0	244.68	1165.7	398.21	1334.9	296.6	245.43	209.7
p5	25	1	238.15	1026.6	576.71	876.04	344.3	244.63	151.9
p6	30	0.5	333.92	1549.6	1023.2	2458.2	865.7	340.15	294.9
p7	30	0	314.45	1476.9	1095.7	2130.1	936.2	319.40	429.2
p8	40	1	407.44	1934.2	1002.3	3965.4	999.3	422.95	265.8

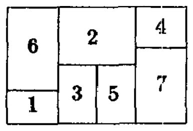
Table 1

λ_w	S.A. Solution		A	A + W	A + 2W	A + 3W
	A	W				
0	196.42	152.1	196.42*	348.52	500.62	652.72
1	206.60	103.7	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 2

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

Table 3


Slicing structure
Fig.1a

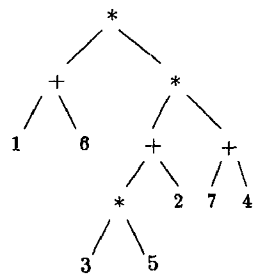
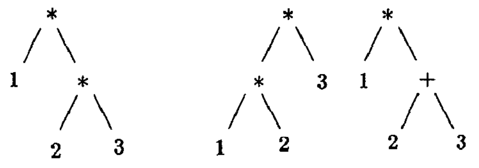


Fig.1b. Slicing tree


not skewed skewed
Fig.2

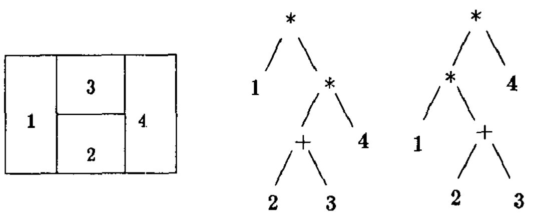


Fig.3. Two different slicing trees for the same slicing structure.

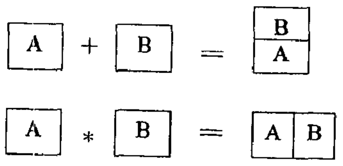


Fig.4

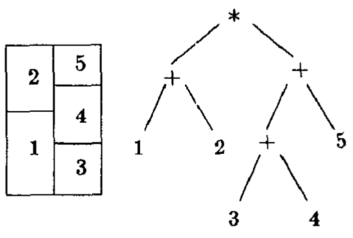


Fig.5

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

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

Fig.6

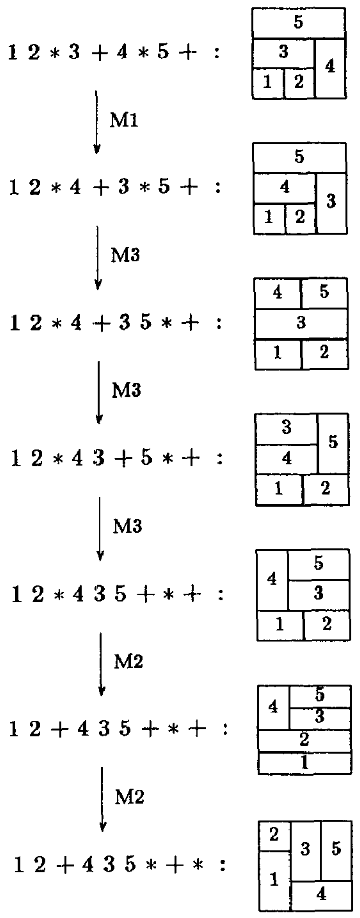


Fig.7

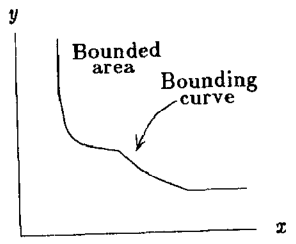


Fig.8

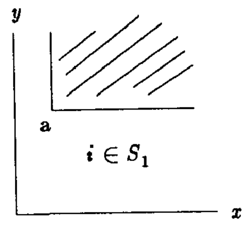


Fig.9a

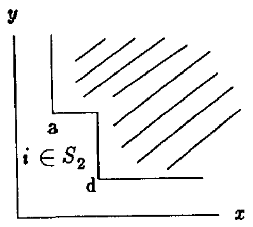


Fig.9b

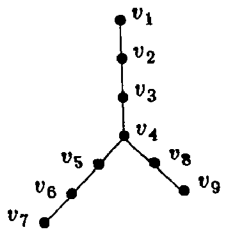


Fig.10

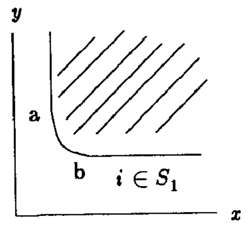


Fig.9c

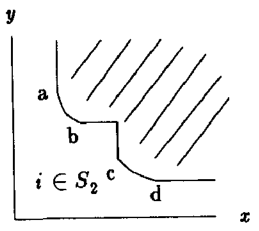


Fig.9d

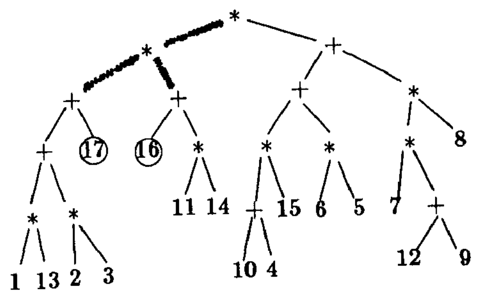
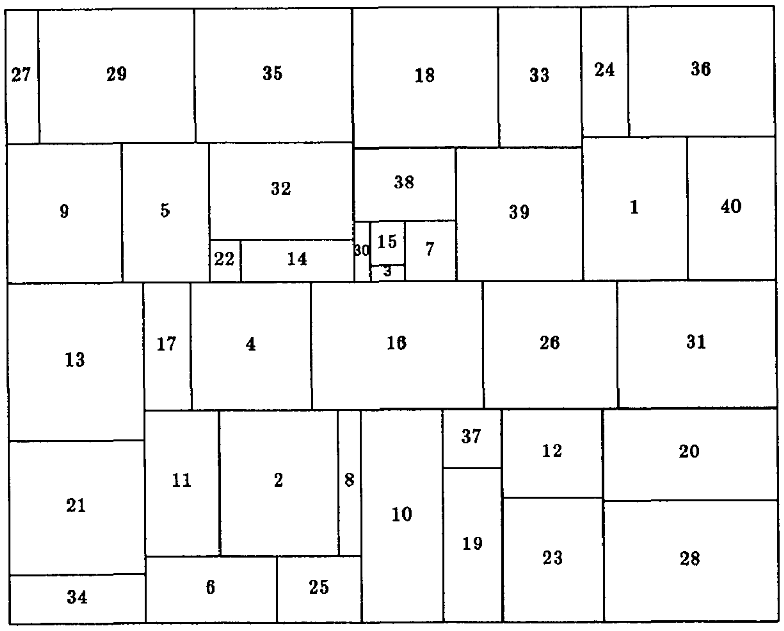


Fig.11a



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.12

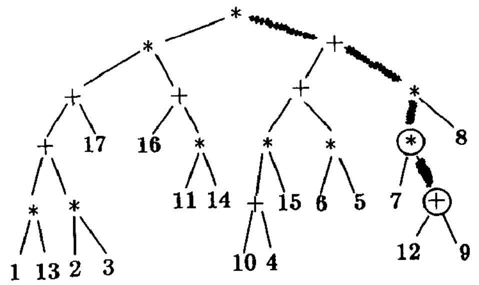


Fig.11b

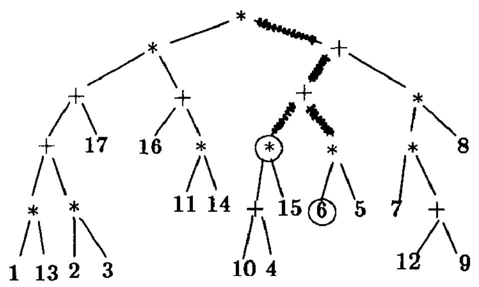


Fig.11c