SLICING FLOORPLANS WITH PRE-PLACED MODULES*

F.Y. Young and D.F. Wong
Department of Computer Sciences
The University of Texas at Austin
fyyoung@cs.utexas.edu wong@cs.utexas.edu

Abstract

Existing floorplanners that use slicing floorplans are efficient in runtime and yet can pack modules tightly. However, none of them can handle pre-placed modules. In this paper, we extend a well-known slicing floorplanner [10] to handle pre-placed modules. Our main contribution is a novel shape curve computation procedure which can take the positions of the pre-placed modules into consideration. The shape curve computation procedure is used repeatedly during the floorplanning process to fully exploit the shape flexibility of the modules to give a tight packing. Experimental results show that the extended floorplanner performs very well.

1 Introduction

Floorplan design is an important step in the physical design of VLSI circuits. It is the problem of placing a set of circuit modules on a chip to minimize total area and interconnect cost. In this early stage of physical design, most of the modules are not yet designed and thus are flexible in shape (soft modules), some are completely designed and have no flexibility in shape (hard modules), and some are even pre-placed at certain fixed positions (pre-placed modules).

Many existing floorplanners are based on slicing floorplans [10, 3, 5, 9] and they are designed to handle both soft and hard modules. There are several advantages of using slicing floorplans: 1) Focusing only on slicing floorplans significantly reduces the searching space and this leads to fast runtime. 2) The shape flexibilty of the modules can be fully exploited to pack modules tightly based on an efficient shape curve computation technique [8, 6]. It has been shown mathematically that a tight packing is achievable [11]. As a result, existing floorplanners that use slicing floorplans are very efficient in runtime and yet can pack modules tightly.

Recently, there are some interesting research activities in the direction of non-slicing floorplans. Two methods, bound-sliceline-grid (BSG) [7] and sequence-pair [2], are proposed. These methods are originally designed for placement of hard modules only. The sequence-pair method is extended recently to handle pre-placed modules [1] and soft modules [4]. In order to handle soft modules, the sequence-pair method [4] has to solve an expensive convex programming problem to determine the exact shape of each module numerous times during the floorplanning process, and thus results in long runtime. For the same set of benchmark problems (apte, xerox, hp, ami33, ami49) used in [4], we run the slicing floorplanner in [10] and obtain comparable results using only a fraction of the runtime. In fact, we have less than 1% dead space using no more than 7 seconds for each test problem.

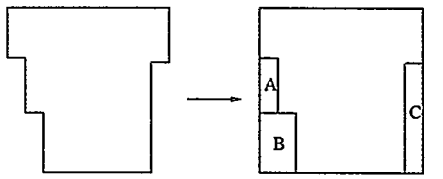

Figure 1: Floorplan with irregular boundaries

It is important that a floorplanner can handle pre-placed modules. In fact, the problem of floorplanning with irregular boundaries can also be solved by treating the protruding parts along the boundaries as pre-placed modules. An example is shown in figure 1. Unfortunately, none of the existing slicing floorplanners has that capability. In this paper, we extend a well-known slicing floorplanner by Wong and Liu [10] to handle pre-placed modules. Our main contribution is a novel shape curve computation procedure which can take the positions of the pre-placed modules into consideration. The shape curve computation procedure is used repeatedly during the floorplanning process to fully exploit the shape flexibility of the modules. Experimental results show that the extended floorplanner performs very well.

2 Problem Definition

A module A is a rectangle of height $h(A)$, width $w(A)$ and area $area(A)$. The aspect ratio of A is defined as $h(A)/w(A)$. A floorplan for n modules consists of an enveloping rectangle R subdivided by horizontal and vertical line segments into n non-overlapping rectangles such that each rectangle i must be large enough to accommodate the

*This work was partially supported by the Texas Advanced Research Program under Grant No. 003658288 and by a grant from the Intel Corporation.

ICCAD98, San Jose, CA, USA

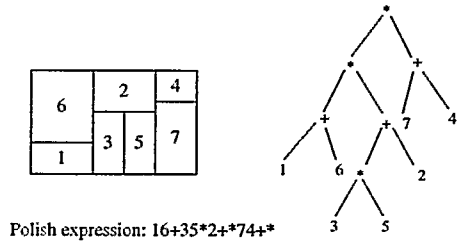

Figure 2: Slicing floorplan and its slicing tree representation

module assigned to it. A supermodule is a sub-floorplan which contains one or more modules. For example, modules "2", "3" and "5" in figure 2 form a supermodule. There are two kinds of floorplans: slicing and non-slicing. A slicing floorplan is a floorplan which can be obtained by recursively cutting a rectangle into two parts by either a vertical or a horizontal line. A non-slicing floorplan is a floorplan which is not a slicing floorplan.

There are three kinds of modules:

Hard modules A hard module is free to move and rotate. The height and width are given, and it has no flexibility in shape.

Soft modules A soft module is also free to move. We are only given its area, and the lower and upper aspect ratio bounds, p and q. The shape can be changed as long as the aspect ratio is within the range $[p, q]$

Pre-placed modules A pre-placed module is fixed in position, height and width.

In our problem, we are given three kinds of modules $M = H \cup F \cup P$. H is a set of hard modules , F is a set of soft modules and P is a set of pre-placed modules. We assume that the given pre-placed modules do not overlap and all of them lie in the first quadrant of the xy-plane. We also assume that the pre-placed modules do not form a non-slicing structure. However this can be easily handled by cutting the pre-placed modules, which are given in a non-slicing manner, into smaller ones so that the packing becomes slicing. A packing is a non-overlap placement of the basic modules. A feasible packing is a packing in the first quadrant such that all the pre-placed modules are placed at their specified positions, and the width and height of all the soft modules are consistent with their aspect ratio constraints and area constraints. Our objective is to construct a feasible floorplan R to minimize $A + \lambda W$ where A is the total area of the packing R , W is an estimation of the interconnect cost and λ is a user-specified constant that controls the relative importance of A and W. We require that the aspect ratio of the final packing is between two given numbers r_{min} and r_{max}.

3 Wong & Liu's Floorplanner

A slicing floorplan can be represented by an oriented rooted binary tree, called a slicing tree (figure 2). Each internal node of the tree is labeled by a * or a +, corresponding to a vertical or a horizontal cut respectively. Each leaf corresponds to a basic module and is labeled by a number from 1 to n. No dimensional information on the position of each cut is specified in the slicing tree. If we traverse a slicing tree in postorder, we obtain a Polish expression. A Polish expression is said to be *normalized* if there is no consecutive

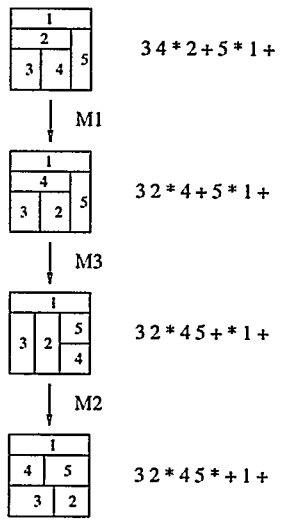

Figure 3: Floorplan transformation in Wong and Liu's algorithm

*'s or +'s in the sequence. It is proved in [10] that there is a 1-1 correspondence between the set of normalized Polish expressions of length $2n - 1$ and the set of slicing floorplans with n modules.

In [10], Wong and Liu used the set of all normalized Polish expressions as the solution space. The simulated annealing method was used. In order to search the solution space efficiently, they defined three types of moves (M1, M2 and M3) to transform a Polish expression into another. Figure 3 shows a series of floorplan transformations. They can make use of the flexibility of the soft modules to select the "best" floorplan among all the equivalent ones represented by the same slicing structures. This is done by carrying out an efficient shape curve computation [6, 10] whenever a Polish expression is examined. The cost function is $A + \lambda W$ where A is the total packing area and W is the interconnect cost. It is shown in [10] that this algorithm is very efficient and the performance is very well. We will describe in the following sections our extensions in order to handle pre-placed modules.

4 Our method

Our method is based on Wong and Liu's algorithm. They assumed $P = \emptyset$ in the problem and we extended their algorithm to handle pre-placed modules. In our algorithm, there is a reference point associated with each supermodule to take into account the pre-placed coordinates. We re-define the operations of combining supermodules to handle reference points, and we add two new types of transformations which are crucial in the case that some modules are pre-placed.

4.1 Supermodules with Reference Points

A normalized Polish expressions can only specify the relative positions of the modules. In order to place the pre-placed modules at some fixed positions, we need to consider their pre-placed coordinates when we combine modules. A simple example is shown in figure 4. Module A and module B are pre-placed with lower left corners at $(1, 1)$ and $(2, 2)$ respectively. In the original algorithm, we do not consider

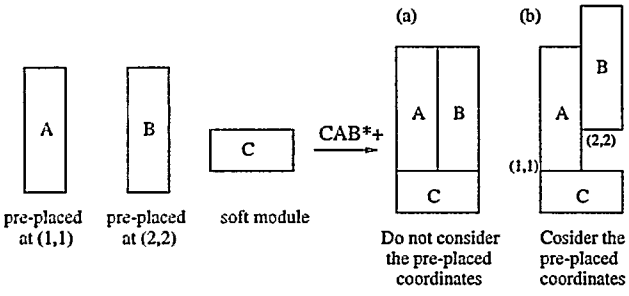

Figure 4: Consider the pre-placed coordinates when combining modules

the pre-placed coordinates, and the expression $CAB*+$ will give us a floorplan as shown in figure 4(a), but this has violated the requirements of placing module A and module B at $(1,1)$ and $(2,2)$ respectively. We want to obtain a floorplan as shown in figure 4(b).

In order to take into consideration the pre-placed coordinates, we associate with each supermodule X a reference point $ref(X)$ if it contains at least one pre-placed module. These reference points correspond to some fixed coordinates on the xy-plane. We also keep the possible positions of this reference point within the supermodule by four numbers, $bottom(X)$, $top(X)$, $left(X)$ and $right(X)$. The reference point can be anywhere within the rectangle enclosed by the four lines: $y = bottom(X)$, $y = h(X) - top(X)$, $x = left(X)$ and $x = w(X) - right(X)$, taking the lower left corner of X as the origin. For example, in figure 8(a), the reference point can be anywhere within the dotted rectangle. The reference point of a pre-placed module is its lower left corner, e.g. module A in figure 5 is pre-placed with lower-left corner at $(1,1)$, so $ref(A) = (1,1)$ and its possible position in the module is simply that point, so $bottom(A) = 0$, $top(A) = 2$, $left(A) = 0$ and $right(A) = 1$. A supermodule will inherit its reference point from one of its two children, e.g. the reference point of X in figure 5 is inherited from A, so $ref(X) = (1,1)$. Since B is not pre-placed, it can slide vertically along the right boundary of A as long as the total area of X remains minimum, so the reference point of X can locate anywhere on the vertical line from P to Q along the left boundary, i.e. $bottom(X) = 0$, $top(X) = 2$, $left(X) = 0$ and $right(X) = 2$.

If we combine the supermodule X in figure 5 with another soft module C as in $Y = CAB*+$ (figure 6), the reference point of the new supermodule Y is inherited from $AB*$, so $ref(Y) = (1,1)$. Since B and C are not pre-placed, we can move B vertically and move C horizontally as long as the total area of Y remains the smallest possible, so $bottom(Y) = 1$, $top(Y) = 2$, $left(Y) = 0$, $right(Y) = 2$.

Intuitively, we can imagine the reference point of a supermodule X as a pin fixed on the xy-plane (figure 7). The four lines: $y = bottom(X)$, $y = h(X) - top(X)$, $x = left(X)$ and $x = w(X) - right(X)$ encloses a rectangular hole in X. We can put X on the xy-plane, with the pin inside the hole, and we can move it freely on the plane as long as the pin is within the hole and X is in the first quadrant. In the next section, we will describe the operations of combining supermodules with reference points and it is not difficult to prove lemma 1.

Lemma 1 *The possible positions of a reference point in a supermodule can only be a single point, a vertical line, a horizontal line or an upright rectangle.*

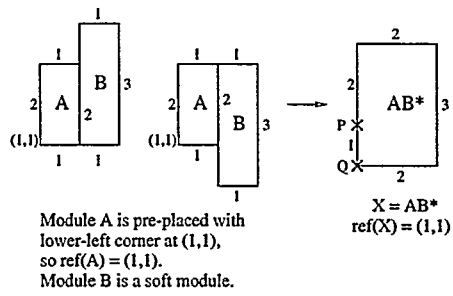

Figure 5: Possible location of the reference point in $X = AB*$

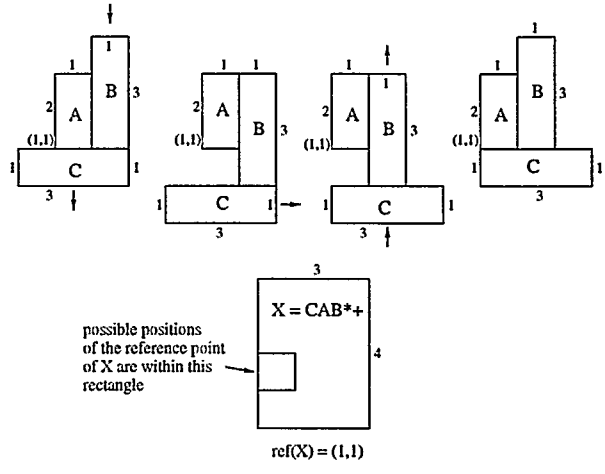

Figure 6: Possible location of the reference point in $Y = CAB*+$

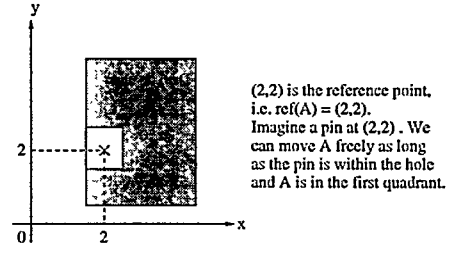

Figure 7: Placing a supermodule with reference point on the xy-plane

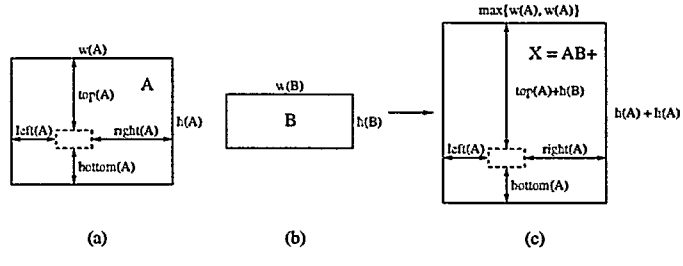

Figure 8: Adding a supermodule with reference point to one without reference point

4.2 Combining Supermodules with Reference Points

In this section, we consider in details how we can combine supermodules with reference points. Here we only consider the addition operation (packing supermodules vertically). The multiplication operation can be done symmetrically. Consider $X = AB+$, it is either (i) one of A or B has pre-placed modules, or (ii) both A and B have pre-placed modules. The case that neither A nor B has pre-placed module can be handled as in [10].

4.2.1 (i) Only one of A or B contains pre-placed modules

Without loss of generality, we assume that only A has pre-placed modules. Let a be the reference point of A. We now put B above A as in figure 8. The reference point of the new supermodule X is inherited from A, so $ref(X) = a$ and $bottom(X) = bottom(A)$, $top(X) = top(A) + h(B)$, $left(X) = left(A)$ and $right(X) = right(A)$.

4.2.2 (ii) Both A and B contain pre-placed modules

Let $a(x_a, y_a)$ and $b(x_b, y_b)$ be the reference point of A and B respectively. X can inherit the reference point from either A or B. Assume that X inherits the reference point from B, i.e. $ref(X) = b$. Now we want to put B above A and we can consider the operation along the vertical and horizontal direction separately:

Vertical direction First, we check that the reference point of B is above that of A and the separation between these two reference points is large enough to put all the modules between them :

$$\begin{aligned} y_b &\geq y_a \\ y_b - y_a &\geq top(A) + bottom(B) \end{aligned}$$

If the above conditions are satisfied, we can put B above A in one of the two ways as illustrated by figure 9, depending on the distance between a and b and the size of the modules between them. We can imagine two pins on the xy-plane, one at a and one at b, and we place A and B on the plane such that the pin at a must be inside the rectangular hole enclosed by $y = bottom(A)$, $y = h(A) - top(A)$, $x = left(A)$ and $x = w(A) - right(A)$ in A taking the lower left corner of A as the origin, and the pin at b must be inside the hole enclosed by $y = bottom(B)$, $y = h(B) - top(B)$, $x = left(B)$ and $x = w(B) - right(B)$ in B taking the lower left corner of B as the origin. We shift A and B on the plane to minimize the total area of the resulting supermodule. We also want $bottom(X)$ and $top(X)$ to be as small as possible in order to have the highest flexibility in subsequent packing. In case one of figure 9, there is no way of shifting A or B vertically without increasing the total area of X, so the possible position of the reference point in X has no flexibility in the y-direction, i.e. top(X) + bottom(X) = h(X). In case two, the values of $bottom(X)$ and $top(X)$ are dependent on the positions of a and b. Figure 10 explains this further by showing the four subcases which give the formula of $top(X)$ and $bottom(X)$ in figure 9.

Horizontal direction Without loss of generality, we assume that $x_b \geq x_a$. Similarly, we can imagine the

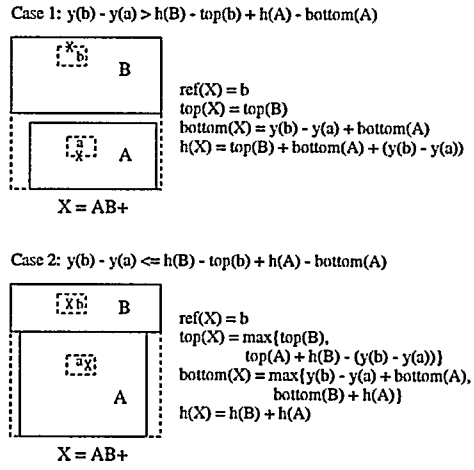

Figure 9: Consider the vertical direction when adding two supermodules with reference points

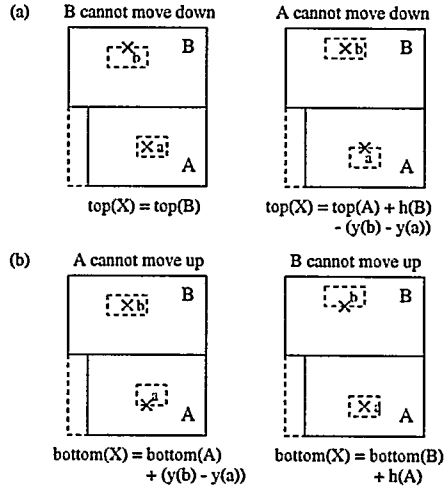

Figure 10: Calculate top(X) and bottom(X) in case 2 of figure 9

pins and the holes on the xy-plane. We want to place the supermodules such that the resulting total area is minimal and the values of $left(X)$ and $right(X)$ are as small as possible. Figure 11 summarizes how we can compute $left(x)$, $right(x)$ and $w(X)$ and figure 12 shows the different subcases which give the formulae of $left(X)$ and $right(X)$ in figure 11. The width of X is also dependent on the positions of a and b and this is explained by figure 13.

We can summarize the computations as follows:

1. $X = AB+$

Pre-conditions:

$$\begin{aligned} y_b &\geq y_a \\ y_b - y_a &\geq top(A) + bottom(B) \end{aligned}$$

Computations: There are two cases:

if ($x_b \geq x_a$)

$ref(X) = ref(B)$

$$h(X) = \max \left\{ \begin{array}{l} top(B) + bottom(A) + (y_b - y_a), \\ h(B) + h(A) \end{array} \right\}$$

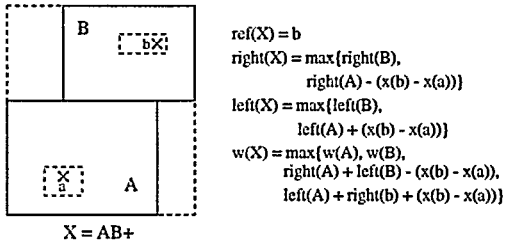

Figure 11: Consider the horizontal direction when adding two supermodules with reference points

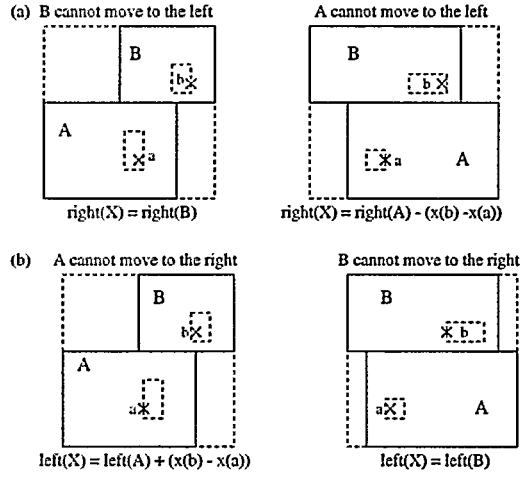

Figure 12: Calculate $left(X)$ and $right(X)$ in figure 11

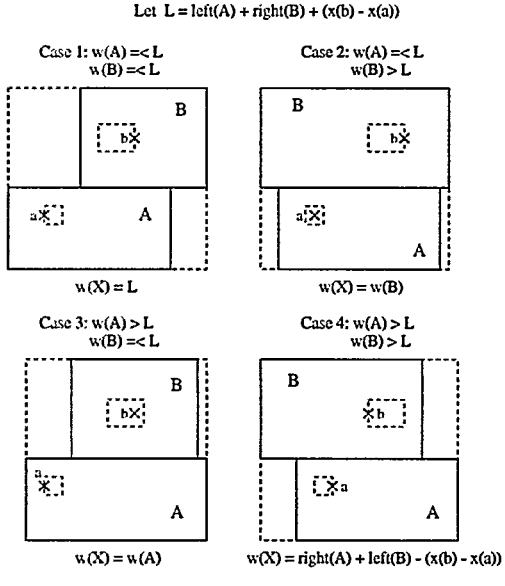

Figure 13: Calculate the width of X in figure 11

$$w(X) = \max\left\{\begin{array}{l} w(A), w(B), \\ right(A) + left(B) - (x_b - x_a), \\ left(A) + right(B) + (x_b - x_a) \end{array}\right\}$$
$$bottom(X) = \max\left\{\begin{array}{l} bottom(A) + y_b - y_a, \\ bottom(B) + h(A) \end{array}\right\}$$
$$top(X) = \max\left\{\begin{array}{l} top(B), \\ top(A) + h(B) - (y_b - y_a) \end{array}\right\}$$
$$left(X) = \max\left\{\begin{array}{l} left(B), \\ left(A) + (x_b - x_a) \end{array}\right\}$$
$$right(X) = \max\left\{\begin{array}{l} right(B), \\ right(A) - (x_b - x_a) \end{array}\right\}$$

else

$$ref(X) = ref(B)$$
$$h(X) = \max\left\{\begin{array}{l} top(B) + bottom(A) + (y_b - y_a), \\ h(B) + h(A) \end{array}\right\}$$
$$w(X) = \max\left\{\begin{array}{l} w(A), w(B), \\ left(A) + right(B) - (x_a - x_b), \\ right(A) + left(B) + (x_a - x_b) \end{array}\right\}$$
$$bottom(X) = \max\left\{\begin{array}{l} bottom(A) + y_b - y_a, \\ bottom(B) + h(A) \end{array}\right\}$$
$$top(X) = \max\left\{\begin{array}{l} top(B), \\ top(A) + h(B) - (y_b - y_a) \end{array}\right\}$$
$$left(X) = \max\left\{\begin{array}{l} left(B), \\ left(A) - (x_a - x_b) \end{array}\right\}$$
$$right(X) = \max\left\{\begin{array}{l} right(B), \\ right(A) + (x_a - x_b) \end{array}\right\}$$

Post-conditions:

$$\begin{aligned} x_b &\geq left(X) \\ y_b &\geq bottom(X) \end{aligned}$$

2. $X = AB*$

Pre-conditions:

$$\begin{aligned} x_b &\geq x_a \\ x_b - x_a &\geq right(A) + left(B) \end{aligned}$$

Computations: There are two cases:

if ($y_b \geq y_a$)

$$ref(X) = ref(B)$$
$$h(X) = \max\left\{\begin{array}{l} h(A), h(B), \\ top(A) + bottom(B) - (y_b - y_a), \\ bottom(A) + top(B) + (y_b - y_a) \end{array}\right\}$$
$$w(X) = \max\left\{\begin{array}{l} right(B) + left(A) + (x_b - x_a), \\ w(B) + w(A) \end{array}\right\}$$
$$bottom(X) = \max\left\{\begin{array}{l} bottom(B), \\ bottom(A) + (y_b - y_a) \end{array}\right\}$$
$$top(X) = \max\left\{\begin{array}{l} top(B), \\ top(A) - (y_b - y_a) \end{array}\right\}$$
$$left(X) = \max\left\{\begin{array}{l} left(A) + x_b - x_a, \\ left(B) + w(A) \end{array}\right\}$$
$$right(X) = \max\left\{\begin{array}{l} right(B), \\ right(A) + w(B) - (x_b - x_a) \end{array}\right\}$$

else

$$ref(X) = ref(B)$$
$$h(X) = \max\left\{\begin{array}{l} h(A), h(B), \\ bottom(A) + top(B) - (y_a - y_b), \\ top(A) + bottom(B) + (y_a - y_b) \end{array}\right\}$$
$$w(X) = \max\left\{\begin{array}{l} right(B) + left(A) + (x_b - x_a), \\ w(B) + w(A) \end{array}\right\}$$
$$left(X) = \max\left\{\begin{array}{l} left(A) + x_b - x_a, \\ left(B) + w(A)\} \end{array}\right\}$$
$$right(X) = \max\left\{\begin{array}{l} right(B), \\ right(A) + w(B) - (x_b - x_a) \end{array}\right\}$$
$$bottom(X) = \max\left\{\begin{array}{l} bottom(B), \\ bottom(A) - (y_a - y_b) \end{array}\right\}$$
$$top(X) = \max\left\{\begin{array}{l} top(B), \\ top(A) + (y_a - y_b) \end{array}\right\}$$

Post-conditions:

$$\begin{aligned} x_b &\geq left(X) \\ y_b &\geq bottom(X) \end{aligned}$$

4.3 Computation of Shape Curves

In Wong and Liu's algorithm, they can exploit fully the flexibility of the soft modules to select the "best" floorplan among all the equivalent ones represented by the same slicing structures. This is done by representing the shape of each module by a shape curve. A shape curve is a decreasing function y of x and it represents the tradeoff between the height and the width of a module. In the original algorithm, all shape curves are piecewise linear. Starting from the modules at the leaves of the slicing trees, the algorithm works upwards to the root, computing the shape curves at each internal node. At the end, the shape curve at the root represents the possible shapes of the floorplan, and we select one such that the aspect ratio is within the bounds.

In our case, the shape curves are either piecewise linear (if the supermodule does not contain any pre-placed module) or are step functions (otherwise). Let Γ and Λ be two shape curves in our case, to compute $\Gamma + \Lambda$ (or $\Gamma * \Lambda$), we can combine pairwise the points on Γ and the points on Λ to give a resultant list. However, we found from practice that it is much more efficient if we combine only those pairs of points with the same x values (or y values for $\Gamma * \Lambda$) and there is no degradation in performance. The resultant list will still be a shape curve, i.e. y decreases with x, and we can repeat this computation procedure bottom up until reaching the root of the slicing tree.

4.4 Moves

In order to handle pre-placed modules, we modify the moves M1 and M3 in [10], and introduce a new move M4. A contiguous sequence of operators is called a *chain*. We use $l(c)$ to denote the length of a chain c. The *complement* of a chain is one obtained by interchanging the operator * and +. For example, in 1 2 3 + * 4 5 6 7 * + * + 8 *, the subsequence * + *+ is a chain, and complementing it gives a new expression: 1 2 3 + * 4 5 6 7 + * + * 8 *. Let $\alpha = \alpha_1\alpha_2 \ldots \alpha_{2n-1}$ be a normalized Polish expression. α can also be written as $c_0\pi_1c_1\pi_2c_2 \ldots c_{n-1}\pi_nc_n$ where $\pi_1\pi_2 \ldots \pi_n$ is a permutation of $1, 2, \ldots, n$, and the c_i's are chains (possibly of zero length), and $\sum_i l(c_i) = n-1$. Two operands in α are said to be adjacent iff they are consecutive elements in $\pi_1\pi_2 \ldots \pi_n$. An operand and an operator are said to be adjacent iff they are consecutive elements in $\alpha_1\alpha_2 \ldots \alpha_{2n-1}$.

Here we define the operation of *combining* two operands. We can combine two operands $combine(\pi_i, \pi_j)$, where $\pi_i = \alpha_k$, $\pi_j = \alpha_l$ and $l(c_i) \neq 0$, by moving α_k and α_{k+1} to position $l+1$ and $l+2$ respectively, giving $\ldots\alpha_l\alpha_k\alpha_{k+1}\ldots$. For example, let α = 1 2 3 + * 4 5 * +. Since "3" is followed by a chain of non-zero length, we can combine "3" and "4" to give a new expression 1 2 * 4 3 + 5 * +. In the following, we define the *reduced Polish expression* of a normalized Polish expression α assuming that α contains pre-placed modules. α can be written as $d_0\beta_1d_1\beta_2d_2 \ldots d_{2m-2}\beta_{2m-1}d_{2m-1}$ where m is the number of pre-placed modules and β_i is either an operand corresponding to a pre-placed module or an operator in α such that both its children subtrees in the slicing tree of α contain pre-placed modules. We define $\beta = \beta_1\beta_2 \ldots \beta_{2m-1}$ as the *reduced Polish expression* of α, which is another Polish expression giving the relative positions of the pre-placed modules. A simple example is shown in Figure 14.

Let $\alpha = c_0\pi_1c_1\pi_2c_2 \ldots c_{n-1}\pi_nc_n$ be the current Polish expression, and β be the reduced Polish expression of α. Recall that P and F are the set of pre-placed modules and soft modules respectively. We have four kinds of moves.

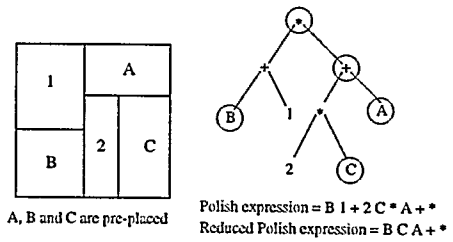

Figure 14: An example of reduced Polish expression

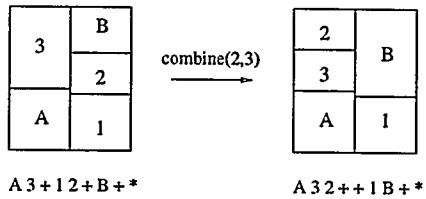

Figure 15: An example of M4

M1 Swap adjacent operands:

- **M1.1** Swap two adjacent operands of which at least one is in P.
- **M1.2** Swap two adjacent operands of which at least one is in F.

M2 Complement a chain of non-zero length.

M3 Swap adjacent operand and operator:

- **M3.1** Swap two adjacent operand and operator in β.
- **M3.2** Swap two adjacent operand and operator in α.

M4 Combine two operands $combine(\pi_i, \pi_j)$ where π_i and π_j are operands and $l(c_i) \neq 0$.

It is clear that M1 and M2 always produce a normalized Polish expression. This is not the case for M3 and M4. We shall only accept those M3 and M4 which result in normalized Polish expressions. M1.1 and M3.1 are specially designed to change the relative positions of the pre-placed modules quickly. They are crucial in speeding up the searching process. For example, if module A and B are pre-placed with lower left corners at $(0,0)$ and $(0,1)$ respectively, B must be to the right of A in the Polish expression. If this is violated, it will probably take a very long time to correct α without these moves. M4 is designed to move modules out of some congested area. A simple example is shown in figure 15. If module A and B are pre-placed with lower left corner at $(0,0)$ and $(1,1)$ respectively, it is infeasible to pile up module 1 and 2 on the right hand side of A below B. $Combine(2,3)$ can move 2 out of the congested area.

These moves are invoked with different probability at different stages. When the relative positions of the pre-placed modules are incorrect, we will invoke M1.1, M2 and M3.1 with higher probability, At the other times, M1.2, M2, M3.2 and M4 are invoked more frequently. These four types of moves are sufficient to ensure a possible transformation from a normalized Polish expression into any other via a sequence of moves.

4.5 Cost Function

The cost function is defined as $A + \lambda W + \gamma D$ where A is the total area of the packing obtained from the shape curve at the root of the slicing tree. W currently is a simple estimation of the interconnect cost as in [10]. This can be easily replaced by a more sophisticated and accurate estimation in the future. D is a penalty term which is zero when the packing is feasible, and is an estimation of the area where the free modules overlap with the pre-placed modules otherwise. D will drop to zero as the annealing process proceeds. λ and γ are constants which control the relative importance of the three terms. λ is usually set such that the area term and the interconnect term are approximately balanced. γ is set to a large value to pull the pre-placed modules to the desired positions at the beginning of the process.

5 Experimental Results

We tested our method on some MCNC building blocks examples and on some randomly generated data. For the MCNC data, we picked three to four of the larger ones as pre-placed modules and treated the rest as soft modules. The starting temperature was decided such that an accept ratio was almost 100%. The temperature was lowered at a constant rate, and the number of iterations for one temperature step varied between twenty to thirty times the number of modules. All the experiments were carried out on a 300 MHz Pentium II Intel processor.

The results are shown in table 1. We can see that the dead space percentage is at most 6.6% in the presence of pre-placed modules and the time taken is less than six minutes. We also tested the method with some larger randomly generated examples (50 to 70 modules) and the runtime is still reasonably about six minutes. Figure 16 shows the result packing of ami33. The packing is reasonably tight given the constraints of the pre-placed modules and the runtime is short, though it can be faster when there is no pre-placed module.

Data A (n, $\|P\|$)	Total area with ($\times 1000 \mu m^2$)	Dead space in area (%)	Time (sec)
apte (9, 2)	49578	2.9	4.94
xerox (10, 2)	19813	1.7	2.75
hp (11, 2)	9621	6.6	3.96
ami33 (33, 3)	1212.42	4.0	270.7
ami49 (49, 4)	3857.9	6.6	336.9

Table 1: Performance of our method on MCNC examples

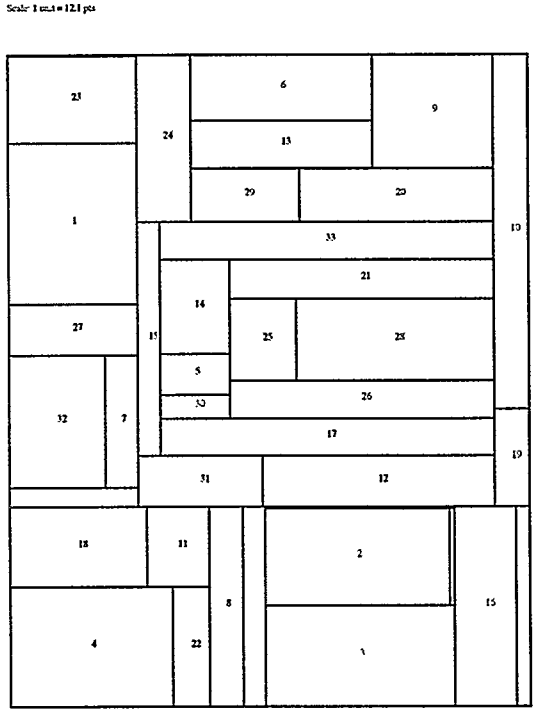

Figure 16: Results of running our program on ami33. The shaded modules are pre-placed.

References

[1] K. Fujiyoushi H. Murata and M. Kaneko. VLSI/PCB placement with obstacles based on sequence-pair. *International Symposium on Physical Design*, pages 26–31, 1997.

[2] S. Nakatake H. Murata, K. Fujiyoushi and Y. Kajitani. Rectangle-packing-based module placement. *Proceedings IEEE International Conference on Computer-Aided Design*, pages 472–479, 1995.

[3] D.P. Lapotin and S.W. Director. A new algorithm for floorplan design. *Proceedings IEEE International Conference on Computer-Aided Design*, pages 143–145, 1985.

[4] H. Murata and Ernest S. Kuh. Sequence-pair based placement method for hard/soft/pre-placed modules. *International Symposium on Physical Design*, 1998.

[5] R.H.J.M. Otten. Automatic floorplan design. *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 261–267, 1982.

[6] R.H.J.M. Otten. Efficient floorplan optimization. *IEEE International Conference on Computer Design*, pages 499–502, 1983.

[7] H. Murata S. Nakatake, K. Fujiyoushi and Y. Kajitani. Module placement on BSG-structure and IC layout applications. *Proceedings IEEE International Conference on Computer-Aided Design*, pages 484–491, 1996.

[8] L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, 59:91–101, 1983.

[9] T. Tamanouchi, K. Tamakashi, and T. Kambe. Hybrid floorplanning based on partial clustering and module restructuring. *Proceedings IEEE International Conference on Computer-Aided Design*, pages 478–483, 1996.

[10] D.F. Wong and C.L. Liu. A new algorithm for floorplan design. *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 101–107, 1986.

[11] F.Y. Young and D.F. Wong. How good are slicing floorplans. *Integration, the VLSI journal*, 23:61–73, 1997. Also appeared in ISPD97.