[3] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 530–533.
[4] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning—A survey," *Integration*, vol. 19, pp. 1–81, 1995.
[5] A. E. Caldwell, A. B. Kahng, and I. L. Markov. GSRC bookshelf for VLSI CAD algorithms. [Online]. Available: HTTP: http://vlsicad.cs.ucla.edu/GSRC/bookshelf.
[6] ——, "Partitioning with terminals—A ~'new' problem and new benchmarks," presented at the ISPD-99.
[7] J. A. Davis, V. K. De, and J. D. Meindl, "A stochastic wire-length distribution for gigascale integration (GSI)—Part I: Derivation and validation," *IEEE Trans. Electron Devices*, vol. 45, pp. 580–589, Mar. 1998.
[8] A. E. Dunlop and B. W. Kernighan, "A procedure for placement of standard cell VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. 4, pp. 92–98, Jan. 1985.
[9] S. Dutt and W. Deng, "VLSI circuit partitioning by cluster-removal using iterative improvement techniques," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1996, pp. 194–200.
[10] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175–181.
[11] D. J. Huang and A. B. Kahng, "Partitioning-based standard cell global placement with an exact objective," in *Proc. ACM/IEEE Int. Symp. Physical Design*, 1997, pp. 18–25.
[12] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning—Applications in VLSI design," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 526–529.
[13] B. Landman and R. Russo, "On a pin versus block relationship for partitioning of logic graphs," *IEEE Trans. Comput.*, vol. C-20, pp. 1469–1479, Dec. 1971.
[14] P. R. Suaris and G. Kedem, "Quadrisection—A new approach to standard cell layout," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1987, pp. 474–477.
[15] D. Sylvester and K. Keutzer, "Getting to the bottom of deep-submicron," presented at the IEEE Int. Conf. Computer-Aided Design, Nov. 1998.

Slicing Floorplans with Range Constraint

F. Y. Young, D. F. Wong, and Hannah H. Yang

***Abstract*—In floorplanning, it is important to allow users to specify placement constraints. Floorplanning with preplaced constraint is considered recently in Murata *et al.* (1997) and Young and Wong (1998). In this paper, we address a more general kind of placement constraint called range constraint in which a module must be placed within a given rectangular region in the floorplan. This is a more general formulation of the placement constraint problem and any preplaced constraint can be written as a range constraint. We extend the Wong-Liu algorithm (1986) to handle range constraint. Our main contribution is a novel shape curve computation which takes range constraint into consideration. Experimental results show that the extended floorplanner performs very well and, in particular, it out-performs the floorplanner in [13] when specialized to handle preplaced modules.**

***Index Terms*—Floorplanning, placement constraint, physical design, simulated annealing, slicing floorplan.**

Manuscript received June 1, 1999; revised September 29, 1999. This work was supported in part by a grant from the Intel Corporation. This paper was recommended by Associate Editor D. Hill.

F. Y. Young is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, New Territories, Hong Kong.
D. F. Wong is with the Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712-1188 USA.
H. H. Yang is wit the Intel Corporation, Hillsboro, OR 97124-5961 USA.

Publisher Item Identifier S 0278-0070(00)01800-5.

I. Introduction

Floorplan design is an important step in physical design of very large scale integration circuits. It is the problem of placing a set of circuit modules on a chip to minimize the 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), while some are reused modules and their shapes are fixed (hard modules). There are two kinds of floorplans: slicing and nonslicing. Many existing floorplanners are based on slicing floorplans [1], [2], [7], [10], [11]. There are several advantages of using slicing floorplans. Firstly, focusing only on slicing floorplans significantly reduces the search space which in turn leads to a faster runtime, especially when the simulated annealing method is used. Secondly, the shape flexibility of the soft modules can be fully exploited to give a tight packing based on an efficient shape curve computational technique [8], [9]. It has been shown mathematically that a tight packing is achievable [12] for slicing floorplans.

There are some interesting results in the direction of nonslicing floorplans recently. Two methods, sequence-pair [4] and bound-slice-line-grid (BSG) [6], have been proposed for placement of hard modules. The sequence-pair method has later been extended to handle preplaced modules [3] and soft modules [5]. In order to handle soft modules, it has to solve a mathematical programming problem to determine the exact shape of each module numerous times in the floorplanning process, and this results in long runtime.

In floorplanning, it is important to allow users to specify placement constraints. Three common types of placement constraints are preplaced constraint, boundary constraint, and range constraint. For preplaced constraint, we require a module to be placed exactly at a certain position in the final packing. In fact, the problem of floorplanning with obstacles can be solved by treating the obstacles as preplaced modules. This problem has been considered in both slicing and nonslicing floorplans [3], [5], [13]. For boundary constraint, we require a module to be placed along one particular side of the final floorplan: on the left, on the right, at the bottom, or at the top. This is useful when users want to place some specific modules along the boundary for input–output connections. This problem is considered recently in a slicing floorplanner [14]. In this paper, we consider the range constraint problem as an enhancement and extension of [13]. In this problem, we require a module to be placed within a given rectangular region in the final packing. This is indeed a more general formulation of the placement constraint problem and any preplaced constraint can be written as a range constraint by specifying the rectangular region such that it has the same size as the module itself. This less-restrictive constraint is useful in practice, but no previous work is reported on it before.

In this paper, we address this range constraint problem by extending the Wong–Liu algorithm [11]. Our main contribution is a novel shape curve computation which takes range constraint into consideration. When our algorithm is specialized to handle preplaced modules, we out-perform the floorplanner in [13]. Note that the floorplanner in [13] is also based on the Wong–Liu algorithm [11]. It extends the shape curve computation to understand preplaced constraint using the notion of reference point and a more complicated set of moves in the annealing process. In this paper, we use a simplier approach which uses only the original set of simple moves in [11]. Experimental results show that the extended floorplanner performs very well. The rest of the paper is organized as follows. We first define the problem formally in Section II. Section III presents our method to handle range constraint. Experimental results are shown in Section IV.

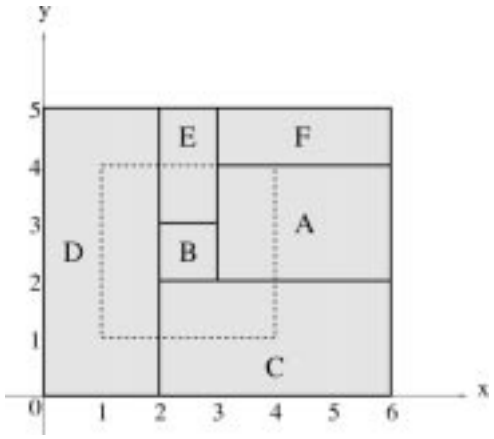

Fig. 1. An example of the preplaced constraint and range constraint.

II. Problem Definition

A module A is a rectangle of height $h(A)$ and width $w(A)$. Let $area(A) = h(A) \times w(A)$ denotes the area of A. The *aspect ratio* of A is defined as $h(A)/w(A)$. A floorplan for n modules consists of an enveloping rectangle subdivided by horizontal and vertical line segments into n nonoverlapping rectangles such that each rectangle must be large enough to accommodate the module assigned to it. A *supermodule* is a subfloorplan which contains one or more modules. There are two kinds of floorplans: *slicing* and *nonslicing*. A slicing floorplan is a floorplan which can be obtained by recursively cutting a rectangle into two parts by either a vertical line or horizontal line. A nonslicing floorplan is a floorplan which is not restricted to be slicing. A module can either be *hard* or *soft*. The height and width of a hard module are fixed but the module is free to rotate. The shape of a soft module can be changed as long as the area remains a constant and the aspect ratio is within a given range. We assume that the floorplan is in the first quadrant with its lower-left corner at the origin and we consider two types of placement constraints:

1) Preplaced constraint: Given a hard module A with fixed orientation and a point (x_1, y_1) in the first quadrant, A must be placed with its lower left corner at (x_1, y_1) in the final floorplan.

2) Range constraint: Given a hard module A and a rectangular region $R_1 = \{(x, y)|x_1 \leq x \leq x_2, y_1 \leq y \leq y_2\}$, A must be placed inside R_1 in the final floorplan.

An example is shown in Fig. 1. In Fig. 1, module A has size 3×2 and it must be placed with its lower left corner at (3, 2) (preplaced constraint). Module B has size 1×1 and it must be placed within the dotted line region $\{(x, y)|1 \leq x \leq 4, 1 \leq y \leq 4\}$ (range constraint). The floorplan shown in Fig. 1 is a feasible one in which both constraints are satisfied.

In our problem, we are given two kinds of modules $M = M_f \cup M_p$ where M_f contains modules which do not have any constraint in placement and M_p contains modules which have either preplaced constraint (P) or range constraint (R). We assume that the preplaced modules are given as nonoverlapping and all of them lie in the first quadrant of the x-y plane. We also assume that they do not form a nonslicing structure. (If they are given as nonslicing, we can preprocess them by cutting their total occupied area into a slicing structure.) Notice that preplaced constraint is a special kind of range constraint in which the module has no freedom of movement. (For example, in Fig. 1, the preplaced constraint for module A can be specified by a range constraint requiring the module to be placed within the region $\{(x, y)|3 \leq x \leq 6, 2 \leq y \leq 4\}$.) Therefore, we will focus on solving the more general range constraint problem. A *packing* is a nonoverlap placement of all the modules in M. A *feasible packing* is a packing in the first quadrant such that all the placement constraints are satisfied, and the widths and heights of all the soft modules are consistent with their aspect ratio constraints and area constraints. Our objective is to construct a feasible floorplan F to minimize $A + \lambda W$ where A is the total area of the packing, W is an estimation of the interconnect cost, and λ is a user-specified constant which controls the relative importance of A and W in the cost function. We require that the aspect ratio of the final packing is between two given numbers $r_{\min}$ and $r_{\max}$.

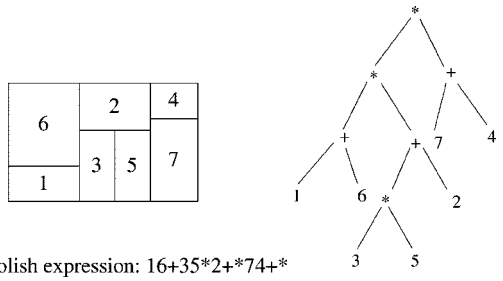

Fig. 2. Slicing tree representation and Polish expression representation of a slicing floorplan.

III. Floorplanning with Range Constraint

In the Wong–Liu algorithm, a slicing floorplan is represented by an oriented rooted binary tree, called a slicing tree (Fig. 2). Each internal node of the tree is labeled by a $*$ or a $+$ operator, corresponding to a vertical or a horizontal cut, respectively. Each leaf corresponds to a basic module and is labeled by a number from one 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 $*$'s or $+$'s in the sequence. It is proved in [11] 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. The Wong–Liu algorithm can fully exploit the shape 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 piecewise linear decreasing curve which represents the tradeoff between the height and width of a module. Starting from the basic modules at the leaves of the slicing trees, the algorithm works upwards to the root, computing the shape curve at each internal node. At the end, the shape curve at the root represents all possible shapes of the final floorplan, and we select one such that the aspect ratio is within the required bounds.

Now some of the basic modules at the leaves have placement constraints. A key observation is that when we put together two modules at least one of which has range constraint, the combined supermodule will also have range constraint. The range constraint information will, thus, be propagated upward from the leaves to the root, and we need to keep in the shape curves both the dimensional information, i.e., the height and width, and the placement constraint information. Let X be a basic module or a supermodule with range constraint, we use four variables to represent the constraint.

- $right(X)$: The shortest distance of the right boundary of X from the y-axis.
- $left(X)$: The longest distance of the left boundary of X from the y-axis.
- $top(X)$: The shortest distance of the upper boundary of X from the x-axis.
- $bottom(X)$: The longest distance of the lower boundary of X from the x-axis.

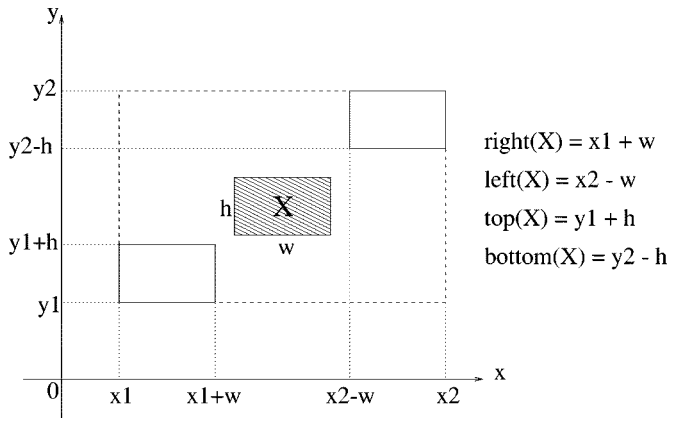

Fig. 3. An example of a module with range constraint.

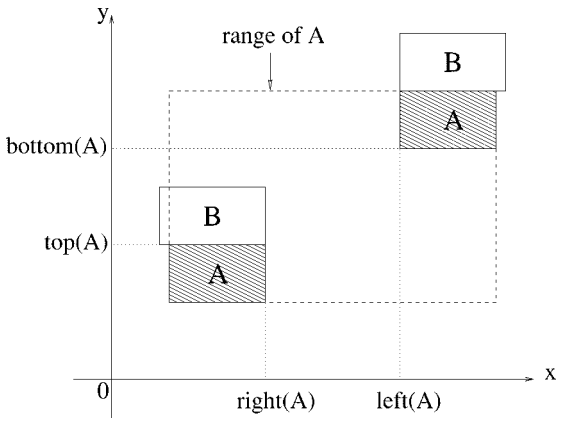

Fig. 4. Only module A has range constraint in $AB+$.

An example is shown in Fig. 3. In Fig. 3, module X has width w and height h and it is constrained to be placed inside the dotted line rectangle $\{(x, y)|x_1 \leq x \leq x_2, y_1 \leq y \leq y_2\}$. Then $right(X) = x_1 + w$, $left(X) = x_2 - w$, $top(X) = y_1 + h$ and $bottom(X) = y_2 - h$. Similarly, we work upwards from the basic modules at the leaves to the root. We compute shape curve at each internal node from the shape curves of its two children, taking into account the placement constraint information. Finally, the shape curve at the root represents the possible shapes of the final floorplan as well as its possible positions on the xy-plane. Consider an internal node v in the slicing tree, let Γ and Λ be the shape curves of its two children, we will combine Γ and Λ point by point to obtain a shape curve for v. (Note that Γ and Λ are piecewise linear with a finite number of corners.) For each pair of points p_1 and p_2 where $p_1 \in \Gamma$ and $p_2 \in \Lambda$, we combine the module represented by the point p_1 with the module represented by the point p_2 to obtain a module which will be represented by a point on the resultant shape curve. We should combine pairwise the points on Γ and Λ. However, we found from practice that it is much more efficient if we just add the two shape curves, i.e., combining only those pairs of points with the same x values if v corresponds to a $+$ operation, and combining only those pairs of points with the same y values if v corresponds to a $*$ operation, and there is no observable degradation in performance. The details of combining two modules with range constraints will be described in Section III-A.

A. Combining Modules with Range Constraints

In this section, we will show how to compute the range constraint of a combined supermodule based on the range constraints of its two children modules. We consider combining two modules A and B vertically to obtain $AB+$, i.e., putting module B above module A. The case where we combine two modules horizontally to obtain $AB*$ can be considered similarly. Assuming that at least one of the two modules has range constraint, there are three different cases.

1) Only A has range constraint: Module B has no placement constraint, so we can put it wherever above module A (Fig. 4). Let X be the combined supermodule, i.e., $X = AB+$, then $h(X) = h(A) + h(B)$, $w(X) = \max\{w(A), w(B)\}$, $right(X) = \max\{w(X), right(A)\}$, $left(X) = left(A)$, $top(X) = top(A) + h(B)$ and $bottom(X) = bottom(A)$.

2) Only B has range constraint: There is a condition which must be satisfied in this case, which is, $bottom(B) \geq h(A)$, because module A will be placed below the x-axis otherwise (Fig. 5). If this condition is satisfied, we can put module A wherever below module B as long as they are both in the first quadrant (Fig. 6). Let X be the combined supermodule, i.e., $X = AB+$, then $h(X) = h(A) + h(B)$, $w(X) = \max\{w(A), w(B)\}$, $right(X) = \max\{w(X), right(B)\}$,

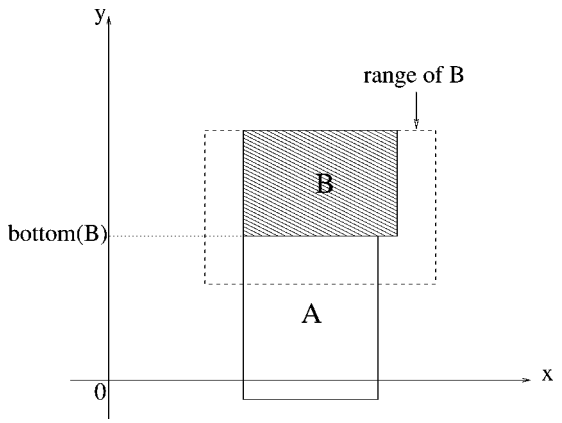

Fig. 5. The necessary condition when only module B has range constraint in $AB+$.

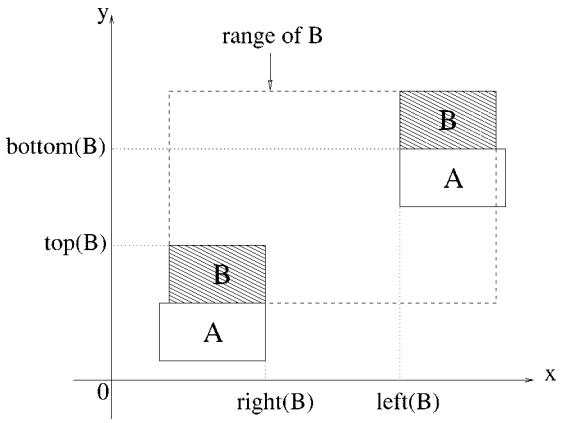

Fig. 6. Only module B has range constraint in $AB+$.

$left(X) = left(B)$, $top(X) = \max\{h(X), top(B)\}$, and $bottom(X) = bottom(B) - h(A)$.

3) Both A and B have range constraints: There is a condition which must be satisfied in this case, which is, $bottom(B) \geq top(A)$, because A and B will overlap vertically otherwise (Fig. 7). If this condition is satisfied, we can compute $X = AB+$ by considering the vertical and horizontal directions separately.

Vertical Direction—We consider two different cases:

Case1) Assume that the range of A does not overlap with the range of B vertically as shown in Fig. 8. We want to put A and B as close to each other as possible, so $h(X) = top(B) -$

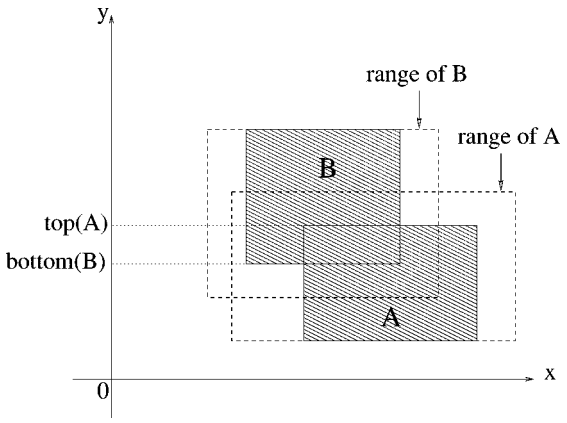

Fig. 7. The necessary condition when both A and B have range constraints in $AB+$.

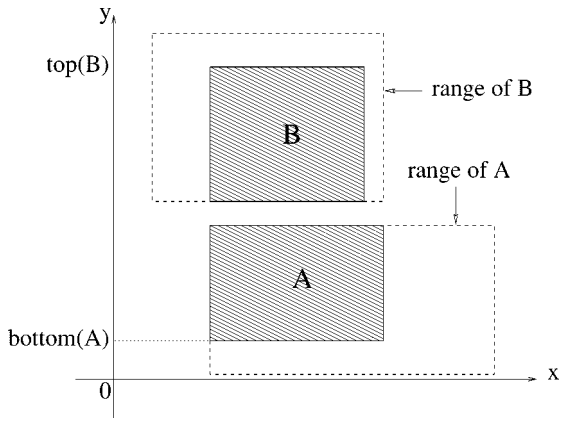

Fig. 8. The ranges of A and B do not overlap vertically.

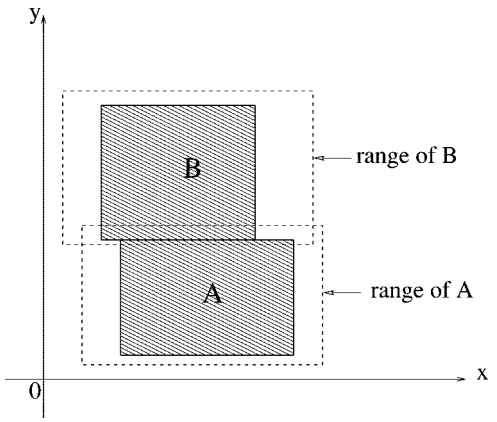

Fig. 9. The ranges of A and B overlap vertically.

$bottom(A)$. The combined supermodule X is fixed in position vertically. $top(X) = top(B)$ and $bottom(X) = bottom(A)$.

Case2) Assume that the range of A and B overlap vertically as shown in Fig. 9. In order to have the smallest combined area, we will put B right above A, so $h(X) = h(A) + h(B)$. For $top(X)$, it is constrained by either $top(A)$ or $top(B)$, so $top(X) = \max\{top(B), top(A) + h(B)\}$. Similarly, $bottom(X)$ is constrained by either $bottom(A)$ or $bottom(B)$, so $bottom(X) = \min\{bottom(B) - h(A), bottom(A)\}$.

Combining the above two cases, we obtain $h(X) = \max\{top(B) - bottom(A), h(A) + h(B)\}$, $top(X) = \max\{top(B), top(A) + h(B)\}$ and $bottom(X) = \min\{bottom(A), bottom(B) - h(A)\}$.

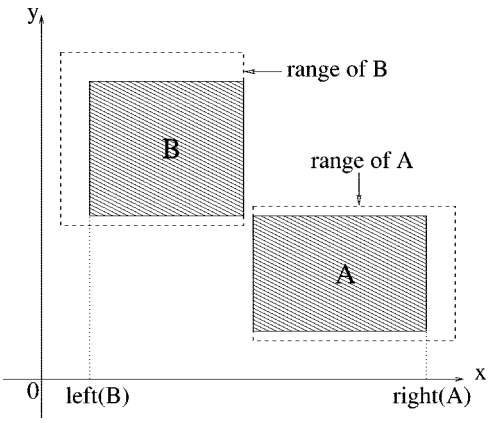

Fig. 10. The ranges of A and B do not overlap horizontally.

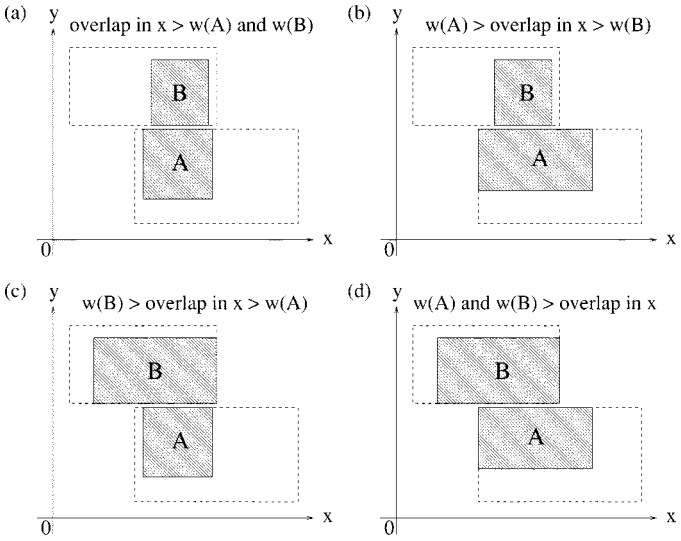

Fig. 11. The ranges of A and B overlap horizontally.

Horizontal Direction—Again, we consider two different cases:

Case1) Assume that the range of A does not overlap with the range of B horizontally as shown in Fig. 10. We will put A and B as close to each other as possible. If the range of A is on the right, $w(X) = right(A) - left(B)$, $right(X) = right(A)$, and $left(X) = left(B)$. Otherwise, $w(X) = right(B) - left(A)$, $right(X) = right(B)$ and $left(X) = left(A)$. Putting them together $w(X) = \max\{right(A) - left(B), right(B) - left(A)\}$, $right(X) = \max\{right(A), right(B)\}$, and $left(X) = \min\{left(A), left(B)\}$.

Case2) Assume that the ranges of A and B overlap horizontally as shown in Fig. 11. If the length of the overlap in the x-direction is greater than either $w(A)$ or $w(B)$ [Fig. 11(a)–(c)], $w(X) = \max\{w(A), w(B)\}$. Otherwise [Fig. 11(d)], we will put A and B as close to each other as possible, so $w(x) = right(A) - left(B)$ if the range of A is on the right, and $w(x) = right(B) - left(A)$ otherwise. Putting them together, $w(X) = \max\{right(A) - left(B), right(B) - left(A), w(A), w(B)\}$. For $right(X)$, it is constrained by either $right(A)$ or $right(B)$, so $right(X) = \max\{right(A), right(B)\}$. Similarly, $left(X)$ is constrained by either $left(A)$ or $left(B)$, so $left(X) = \min\{left(A), left(B)\}$.

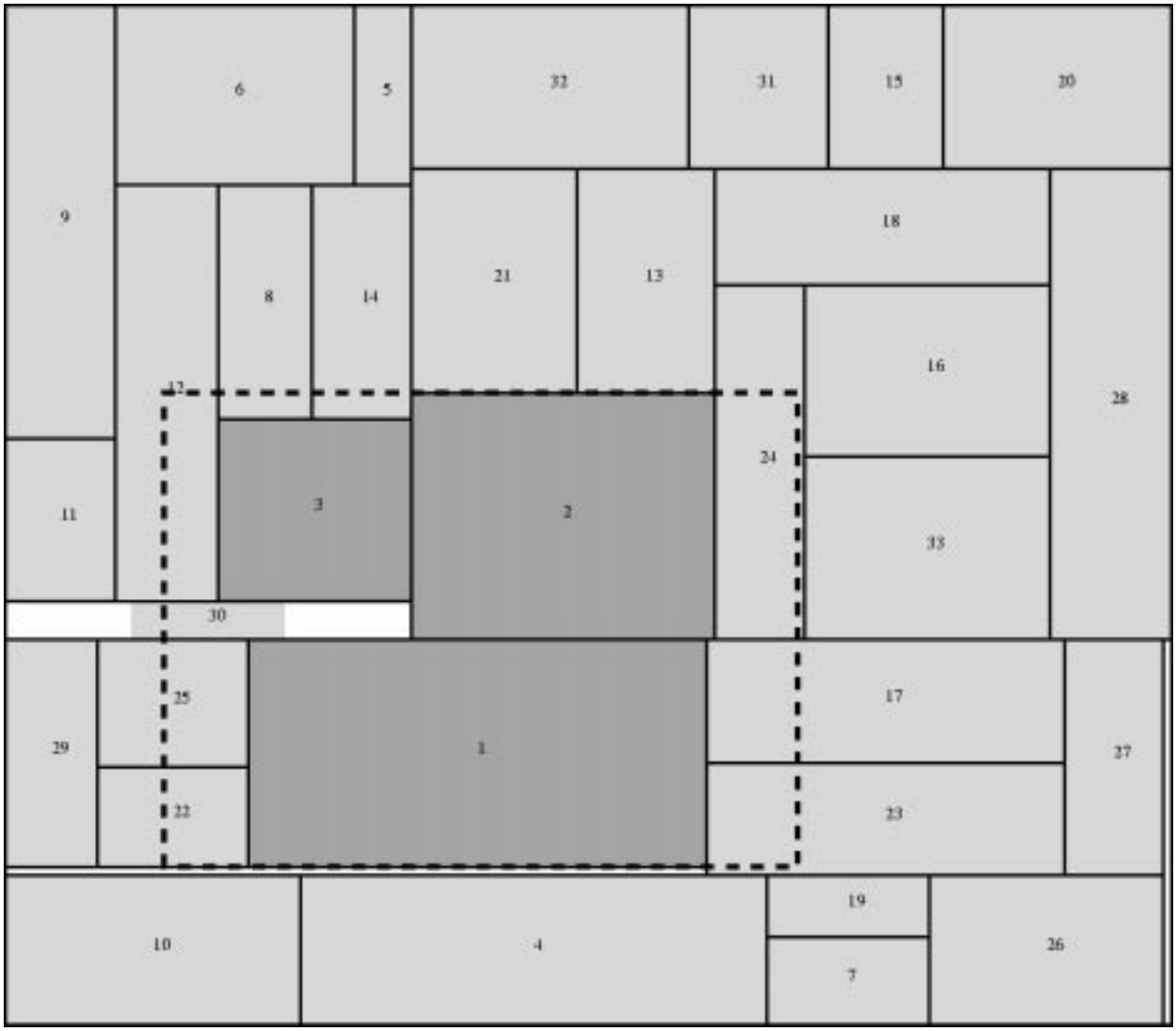

Fig. 12. Result packing of ami33-rc5. Module 1, 2, and 3 are constrained to be placed within the dotted rectangle.

Combining the above two cases, we have $w(X) = \max\{right(A) - left(B), right(B) - left(A), w(A), w(B)\}$, $right(X) = \max\{right(A), right(B)\}$, and $left(X) = \min\{left(A), left(B)\}$.The following formulas summarize the above discussions for the case when both A and B have range constraints. The formulas for the case of putting module B horizontally on the right of A can be derived similarly.

1) $X = AB+$
Necessary condition: $bottom(B) \geq top(A)$
Computations:

$$\begin{aligned} w(X) &= \max\{right(A) - left(B), right(B) \\ &\quad - left(A), w(A), w(B)\} \\ h(X) &= \max\{top(B) - bottom(A), h(A) + h(B)\} \\ top(X) &= \max\{top(B), top(A) + h(B)\} \\ bottom(X) &= \min\{bottom(A), bottom(B) - h(A)\} \\ right(X) &= \max\{right(A), right(B)\} \\ left(X) &= \min\{left(A), left(B)\}. \end{aligned}$$

2) $X = AB*$
Necessary condition: $left(B) \geq right(A)$
Computations:

$$\begin{aligned} w(X) &= \max\{right(B) - left(A), w(A) + w(B)\} \\ h(X) &= \max\{top(A) - bottom(B), top(B) \\ &\quad - bottom(A), h(A), h(B)\} \\ top(X) &= \max\{top(A), top(B)\} \\ bottom(X) &= \min\{bottom(A), bottom(B)\} \\ right(X) &= \max\{right(B), right(A) + w(B)\} \\ left(X) &= \min\{left(A), left(B) - w(A)\}. \end{aligned}$$

B. Moves, Cost Function and Annealing Schedule

We use the same set of moves (M1, M2, and M3) as in [11]. 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. In our current implementation, W is the half perimeter estimation of

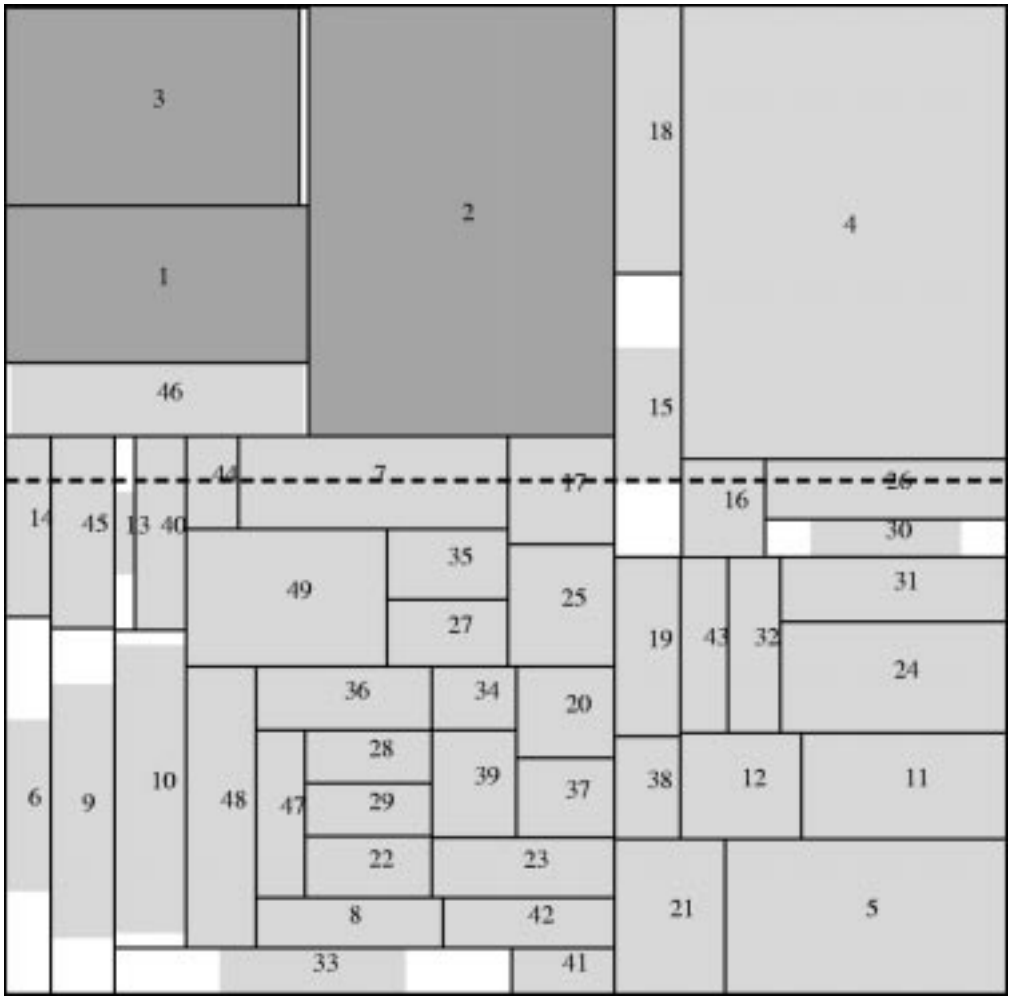

Fig. 13. Result packing of ami49-rc5. Module 1, 2, and 3 are constrained to be placed above the dotted horizontal line.

the interconnect cost. Clearly, this term can be replaced by any more sophisticated interconnect cost estimation. D is a penalty term which is zero when the packing is feasible, and is otherwise an estimation of the total distance of the modules which have range constraints from their desired positions. Notice that if a Polish expression does not correspond to a feasible packing, we will pack the modules in the usual way as if there is no range constraint and the penalty term D will be the total distance of the constrained modules from the ranges they are constrained to. This gives a good estimation of how far the modules are from their desired positions. D will drop to zero as the annealing process proceeds. This can be done by putting γ very large because D becomes the most important factor to be minimized in this case. λ is usually set such that the area term and the interconnect term are approximately balanced. Note that we need to accept infeasible intermediate solutions in the annealing process because it may happen in some cases that good feasible solutions can only be reached from an initial starting point with some infeasible intermediate solutions in the searching process.

The temperature schedule is of the form $T(k) = rT(k-1)$ for all $k \geq 1$. A typical value for r is 0.9. At each temperature, enough number of moves are attempted until there are N downhill moves or the total number of moves exceeds $2N$, where $N = np$, p is a user defined constant and n is the total number of modules. The annealing process terminates when the number of accepted moves is less than 5% of all moves made at a certain temperature or when the temperature is low enough.

TABLE I
RESULTS OF TESTING THE PRE-PLACED CONSTRAINT. COLUMNS 4 AND 5 ARE OBTAINED BY OUR FLOORPLANNER. COLUMNS 6 AND 7 ARE RESULTS FROM [13]

			Our floorplanner		Floorplanner in [13]	
Data	n	# pre-placed	% Dead-space	Time (sec)	% Dead-space	Time (sec)
apte-pc	9	2	1.78	3.51	2.9	4.94
xerox-pc	10	2	1.15	2.66	1.7	2.75
hp-pc	11	2	1.04	3.51	6.6	3.96
ami33-pc	33	4	1.66	85.28	4.0	270.7
ami49-pc	49	4	1.21	204.64	6.6	336.9

IV. EXPERIMENTAL RESULTS

We tested our floorplanner on a set of MCNC benchmark data. For each experiment, the starting temperature is decided such that an acceptance ratio is 100% at the beginning. The temperature is lowered at a constant rate and the number of iterations in one temperature step is proportional to the number of modules. All the experiments were carried out on a 300-MHz Pentium II Intel processor.

We carried out two sets of experiments. For the first set, we want to compare with [13] in handling preplaced constraint. We use exactly the

TABLE II
RESULTS OF TESTING THE RANGE CONSTRAINT

Data	n	Dead space (%)	Time (sec)
ami33-rc1	33	0.81	57.03
ami33-rc2	33	1.89	46.32
ami33-rc3	33	2.64	53.66
ami33-rc4	33	0.83	46.82
ami33-rc5	33	1.64	65.40
ami49-rc1	49	1.56	103.51
ami49-rc2	49	2.12	143.99
ami49-rc3	49	3.44	149.98
ami49-rc4	49	3.72	88.10
ami49-rc5	49	4.86	104.52
playout-rc1	62	1.67	236.98
playout-rc2	62	2.72	225.82
playout-rc3	62	4.96	238.28
playout-rc4	62	3.62	244.87
playout-rc5	62	2.01	208.32

same data in [13] and the comparisons are shown in Table I. Columns 4 and 5 are obtained by our floorplanner and columns 6 and 7 are results from [13]. We can see that our floorplanner has improved over [13] in both efficiency and quality when handling preplaced modules. This can be explained by the fact that we have used a better penalty function here. In [13], the packing process will stop once a violation is found and the penalty is computed by estimating the overlapping area between the free modules and the preplaced modules. Now, in case of infeasible packing, we will still pack all the modules as if there is no placement constraint and the penalty is computed as the total distance of the preplaced modules from their desired positions. This penalty term can describe the error more accurately. Besides, the formulas to combine two shape curves with range constraint are much simplier than those in [13] for preplaced modules and this leads to a faster runtime.

In the second set of experiments, we work on the three largest MCNC benchmarks, ami33, ami49, and playout. We selected three modules from each benchmark and derived 15 data by imposing different range constraints on them. In most cases, we picked three different range constraints (may or may not overlap) for the selected modules, so there are actually multiple range constraints which can reflect better the real design situations. The three selected modules are hard modules while all the other modules have shape flexibility that their aspect ratio can range between 0.25 and 4.0. The results are shown in Table II. Fig. 12 is the result packing of ami33-rc5 where module 1, 2, and 3 are constrained to be placed within the dotted line rectangle. Fig. 13 is the result packing of ami49-rc5 where module 1, 2, and 3 are constrained to be placed above the dotted line. We can see from Table II that the performance in both quality and execution time are good.

We can conclude from the above experimental results that the extended slicing floorplanner can handle preplaced constraint and range constraint very well. Although the current estimation of the interconnect cost is very simple, we can always replace it with a more sophisticated one given the efficiency of our algorithm.

REFERENCES

[1] K. Bazargan, S. Kim, and M. Sarrafzadeh, "Nostradamus: A floorplanner of uncertain design," in *Proc. Int. Symp. Physical Design*, 1998, pp. 18–23.
[2] D. P. Lapotin and S. W. Director, "A global floor-planning tool," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1985, pp. 143–145.
[3] H. Murata, K. Fujiyoushi, and M. Kaneko, "VLSI/PCB placement with obstacles based on sequence-pair," in *Proc. Int. Symp. Physical Design*, 1997, pp. 26–31.
[4] H. Murata, K. Fujiyoushi, S. Nakatake, and Y. Kajitani, "Rectangle-packing-based module placement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1995, pp. 472–479.
[5] H. Murata and E. S. Kuh, "Sequence-pair based placement method for hard/soft/preplaced modules," in *Proc. Int. Symp. Physical Design*, 1998, pp. 167–172.
[6] S. Nakatake, K. Fujiyoushi, H. Murata, and Y. Kajitani, "Module placement on BSG-structure and IC layout applications," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1996, pp. 484–491.
[7] R. H. J. M. Otten, "Automatic floorplan design," in *Proc. 19th ACM/IEEE Design Automation Conf.*, 1982, pp. 261–267.
[8] ——, "Efficient floorplan optimization," in *Proc. IEEE Int. Conf. Computer Design*, 1983, pp. 499–502.
[9] L. Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," *Inform. Contr.*, vol. 59, pp. 91–101, 1983.
[10] T. Tamanouchi, K. Tamakashi, and T. Kambe, "Hybrid floorplanning based on partial clustering and module restructuring," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1996, pp. 478–483.
[11] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," in *Proc. 23rd ACM/IEEE Design Automation Conf.*, 1986, pp. 101–107.
[12] F. Y. Young and D. F. Wong, "How good are slicing floorplans," *Integration VLSI J.*, vol. 23, pp. 61–73, 1997; also appeared in ISPD-97.
[13] ——, "Slicing floorplans with preplaced modules," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1998, pp. 252–258.
[14] F. Y. Young, D. F. Wong, and H. H. Yang, "Slicing floorplans with boundary constraints," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 1385–1389, Sept. 1999.