SLICING FLOORPLANS WITH RANGE CONSTRAINT*

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

In floorplanning, it is important to allow users to specify placement constraints. Floorplanning with pre-placed constraint is considered recently in [3, 13]. In this paper, we address a more general 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 because any pre-placed constraint can be written as a range constraint. We extend the Wong-Liu algorithm [11] 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 pre-placed modules.

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), while some may have been completely designed and have a fixed shape (hard modules). There are two kinds of floorplans: slicing and non-slicing. 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 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 computation 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 non-slicing floorplans recently. Two methods, sequence-pair [4] and bound-sliceline-grid (BSG) [6], have been proposed for placement of hard modules. The sequence-pair method has later been extended to handle pre-placed modules [3] and soft modules [5]. In order to handle soft modules, it has to solve mathematical programming problems 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 pre-placed constraint, boundary constraint and range constraint. For pre-placed constraint, we require a module to be placed exactly at certain position in the final floorplan. In fact, the problem of floorplanning with obstacles can be solved by treating the obstacles as pre-placed modules. This problem has been considered in both slicing and non-slicing floorplan [3, 5, 13]. For boundary constraint, we require a module to be placed along one of the four sides in 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 I/O connections. This problem is considered recently in a slicing floorplanner [14]. Pre-placed constraint is however very restrictive and sometimes, it is already sufficient to allow users to restrict the placement of a module to within a certain range. Therefore we consider the range constraint in which we require a module to be placed within a given rectangular region in the final floorplan. This less restrictive constraint is more useful in practice but no previous work is reported on it before. Actually, the range constraint problem is a more general problem because any pre-placed constraint can be written as a range constraint by specifying the rectangular region such that it has the same size as the module itself.

In this paper, we address this more general 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 pre-placed modules, we out-perform the floorplanner in [13]. Note that the floorplanner in [13] is also based on the Wong-Liu algorithm [11] by extending the shape curve computation to understand pre-placed 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 in which only the original set of simple moves in [11] is used. 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 2. Section 3 provides a brief review of the Wong-Liu algorithm. Section 4 presents our method to handle range constraint. Experimental results are shown in Section 5.

*THIS WORK WAS PARTIALLY SUPPORTED BY A GRANT FROM THE INTEL CORPORATION.

ISPD '99 Monterey CA USA

2. PROBLEM DEFINITION

A module A is a rectangle of height $h(A)$ and width $w(A)$. Let $area(A) = h(A) \times w(A)$ denote 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 non-overlapping rectangles such that each rectangle must be large enough to accommodate the module assigned to it. A *supermodule* is a sub-floorplan which contains one or more modules. 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 line or a horizontal line. A non-slicing floorplan is a floorplan which is not 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:

Pre-placed 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.

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 Figure 1. In Figure 1, module A has size 3×2 and it must be placed with its lower left corner at $(3,2)$ (pre-placed 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 Figure 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 pre-placed constraint (P) or range constraint (R). We assume that the pre-placed modules are given as non-overlapping and all of them lie in the first quadrant of the xy-plane. We also assume that they do not form a non-slicing structure. (If they are given as non-slicing, we can pre-process them by cutting their total occupied area into a slicing structure.) Notice that pre-placed constraint is a special kind of range constraint in which the module has no freedom of movement. (For example, in Figure 1, the pre-placed 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 only focus on solving the more general range constraint problem. A *packing* is a non-overlap 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}.

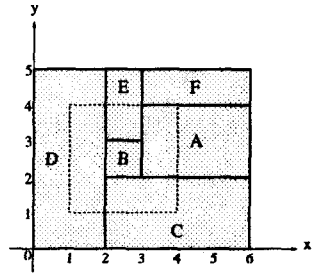

Figure 1. An example of the pre-placed constraint and range constraint

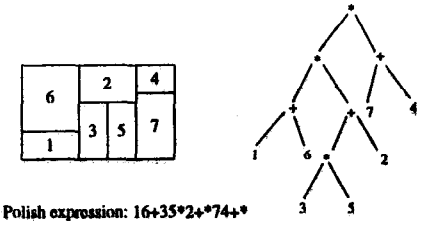

Figure 2. Slicing tree representation and Polish expression representation of a slicing floorplan

3. WONG-LIU ALGORITHM

In this section, we briefly review the Wong-Liu slicing algorithm [11]. 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 $+$ operator, 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 $*$'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.

In [11], Wong and Liu used the simulated annealing method with the set of all normalized Polish expressions as the solution space. 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. (M1 exchanges two adjacent modules in the Polish expression, e.g. $12*3+$ becomes $21*3+$. M2 complements a maximal sequence of operators, e.g. $1234*+*$ becomes $123+*+$. M3 exchanges a module with its adjacent operator, e.g. $123*+$ becomes $12*3+$.) 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 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. This algorithm is very efficient and the performance is very good. (For the same set of benchmark data used by the sequence-pair based algorithm in [5], the slicing floorplanner in [11] can obtain comparable results using only a fraction of the runtime. In fact, it gives less than 1% dead space using no more than 7 seconds for each test problem.) In this paper, we extend the Wong-Liu algorithm to handle range constraint.

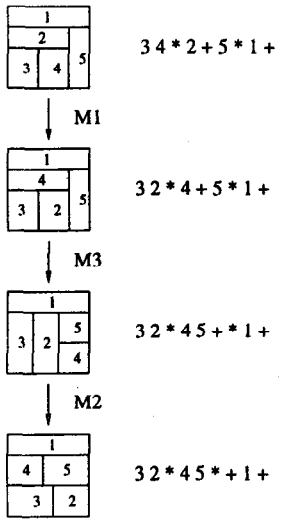

Figure 3. Floorplan transformation in the Wong-Liu algorithm

4. FLOORPLANNING WITH RANGE CONSTRAINT

The Wong-Liu algorithm can fully exploit 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 piecewise linear decreasing curve which represents the tradeoff between the height and the 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 the 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 the 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.

An example is shown in Figure 4. In Figure 4, module X has width w and height h and it is constrainted 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

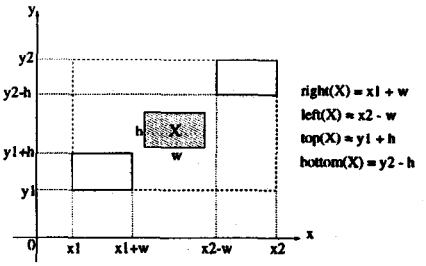

Figure 4. An example of a module with range constraint

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) or the same y values (if v corresponds to a * operation), and there is no degradation in performance. The details of combining two modules with range constraints will be described in the next section.

4.1. Combining Modules with Range Constraints

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

4.1.1. Only A has range constraint

Module B has no placement constraint, so we can put it above wherever module A is (Figure 5). 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 $bottom(X) = bottom(A)$.

4.1.2. Only B has range constraint

There is a condition which must be satisfied in this case, which is, $bottom(B) \geq h(A)$, because otherwise, module A will be placed below the x-axis (Figure 6). If this condition is satisfied, we can put module A below wherever module B is as long as they are both in the first quadrant (Figure 7). Let X be the combined supermodule, i.e. $X = AB+$, then $h(X) = h(A) + h(B)$, $w(X) = \max\{w(A), w(B)\}$, $right(X) = \max\{w(X), right(B)\}$, $left(X) = left(B)$, $top(X) = \max\{h(X), top(B)\}$ and $bottom(X) = bottom(B) - h(A)$.

4.1.3. 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 otherwise, A and B will overlap vertically (Figure 8). If this condition is satisfied, we can compute $X = AB+$ by considering the vertical and the horizontal directions separately:

Vertical Direction We consider two different cases:

Case 1 Assume that the range of A does not overlap with the range of B vertically as shown in Figure 9. We want to put A and B as close to each other as possible, so $h(X) = top(B) - bottom(A)$. The combined supermodule X is fixed in position vertically. $top(X) = top(B)$ and $bottom(X) = bottom(A)$.

Case 2 Assume that the range of A and B overlap vertically as shown in Figure 10. 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 constrainted by either $top(A)$ or $top(B)$, so $top(X) = \max\{top(B), top(A) + h(B)\}$. Similarly, $bottom(X)$ is constrainted by either $bottom(A)$ or $bottom(B)$, so $bottom(X) = \min\{bottom(A), bottom(B) - h(A)\}$.

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 $bottom(X) = \min\{bottom(A), bottom(B) - h(A)\}$.

Horizontal Direction Again, we consider two different cases:

Case 1 Assume that the range of A does not overlap with the range of B horizontally as shown in Figure 11. 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 $left(X) = \min\{left(A), left(B)\}$.

Case 2 Assume that the ranges of A and B overlap horizontally as shown in Figure 12. If the length of the overlap in the x-direction is greater than either $w(A)$ or $w(B)$ (Figure 12(a-c)), $w(X) = \max\{w(A), w(B)\}$. Otherwise (Figure 12(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 constrainted by either $right(A)$ or $right(B)$, so $right(X) = \max\{right(A), right(B)\}$. Similarly $left(X)$ is constrainted by either $left(A)$ or $left(B)$, so $left(X) = \min\{left(A), left(B)\}$.

Combining the above two cases, we obtain $right(X) = \max\{right(A), right(B)\}$, $left(X) = \min\{left(A), left(B)\}$ and $w(X) = \max\{right(A) - left(B), right(B) - left(A), w(A), w(B)\}$.

The following formulae summarize the above discussion for the case when both A and B have range constraints. The formulae for the case of putting module B horizontally on the right of A can be derived similarly:

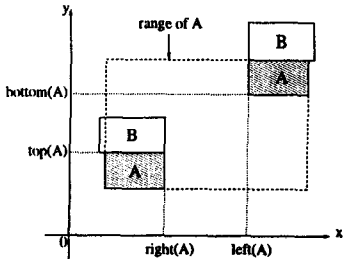

Figure 5. Only module A has range constraint in $AB+$

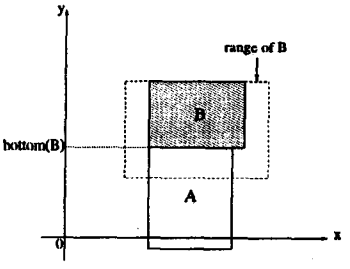

Figure 6. The necessary condition when only module B has range constraint in $AB+$

1. $X = AB+$

   Necessary condition: $bottom(B) \geq top(A)$
   Computations:

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

2. $X = AB*$

   Necessary condition: $left(b) \geq right(A)$
   Computations:

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

4.2. Moves and Cost Function

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 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

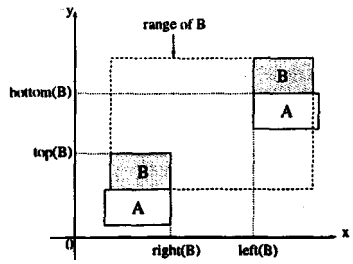

Figure 7. Only module B has range constraint in $AB+$

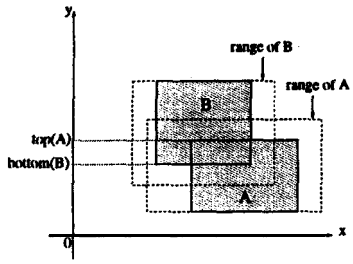

Figure 8. The necessary condition when both A and B have range constraints in $AB+$

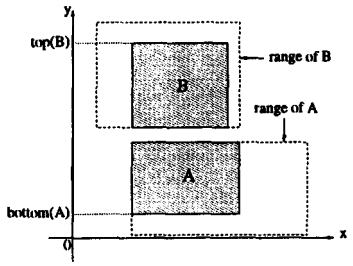

Figure 9. The ranges of A and B do not overlap vertically.

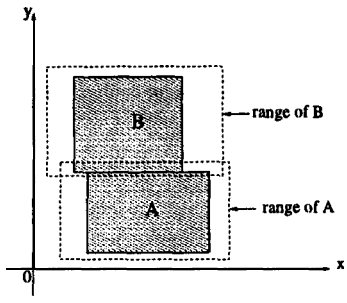

Figure 10. The ranges of A and B overlap vertically.

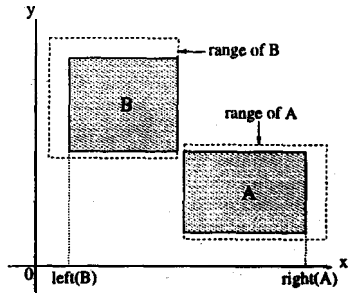

Figure 11. The ranges of A and B do not overlap horizontally.

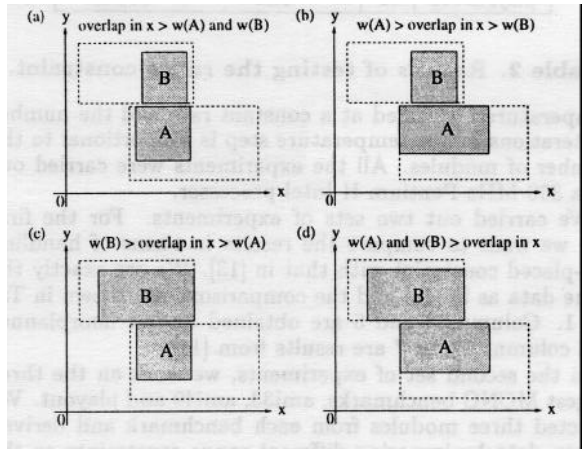

Figure 12. The ranges of A and B overlap horizontally.

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 modules which have range constraints from the centers of the ranges they are constrainted 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. 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.

			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

Table 1. Results of testing the pre-placed constraint. Columns 4 and 5 are obtained by our floorplanner. Columns 6 and 7 are results from [13].

5. EXPERIMENTAL RESULTS

We tested our floorplanner on a set of MCNC benchmarks. For each experiment, the starting temperature is decided such that an acceptance ratio is 100% at the beginning. The

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

Table 2. Results of testing the range constraint.

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 the results in terms of handling pre-placed constraint with that in [13]. We use exactly the same data as in [13] and the comparisons are shown in Table 1. Columns 4 and 5 are obtained by our floorplanner and columns 6 and 7 are results from [13].

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 fifteen data by imposing different range constraints on the selected modules. 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. The results are shown in Table 2. Figure 13 is the result packing of ami33-rc5 where module 1, 2 and 3 are constrainted to be placed within the dotted line rectangle. Figure 14 is the result packing of ami49-rc5 where module 1, 2 and 3 are constrainted to be placed above the dotted line. We can see from Table 2 that the performance in both quality and execution time are very good.

We can conclude from the above experimental results that the extended slicing floorplanner can handle pre-placed 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.

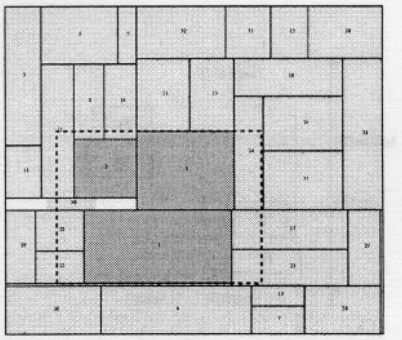

Figure 13. Result packing of ami33-rc5. Module 1, 2 and 3 are constrainted to be placed within the dotted rectangle.

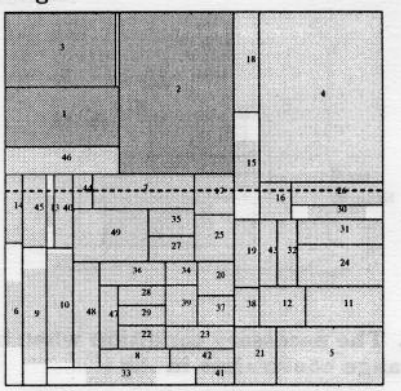

Figure 14. Result packing of ami49-rc5. Module 1, 2 and 3 are constrainted to be placed above the dotted horizontal line.

ACKNOWLEDGEMENT

We want to thank Dr. Hannah Yang at the Intel Corporation for her helpful discussions.

REFERENCES

[1] K. Bazargan, S. Kim, and M. Sarrafzadeh. Nostradamus: A floorplanner of uncertain design. *International Symposium on Physical Design*, pages 18–23, 1998.

[2] D.P. Lapotin and S.W. Director. Mason: A global floorplanning tool. *Proceedings IEEE International Conference on Computer-Aided Design*, pages 143–145, 1985.

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

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

[5] H. Murata and Ernest S. Kuh. Sequence-pair based placement method for hard/soft/pre-placed modules. *International Symposium on Physical Design*, pages 167–172, 1998.

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

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

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

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

[10] 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.

[11] 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.

[12] F.Y. Young and D.F. Wong. How good are slicing floorplans. *Integration, the VLSI journal*, 23:61–73, 1997. Also appeared in ISPD-97.

[13] F.Y. Young and D.F. Wong. Slicing floorplans with pre-placed modules. *Proceedings IEEE International Conference on Computer-Aided Design*, pages 252–258, 1998.

[14] F.Y. Young and D.F. Wong. Slicing floorplans with boundary constraints. *IEEE Asia South Pacific Design Automation Conference*, pages 17–20, 1999.