Handling Soft Modules in General Nonslicing Floorplan Using Lagrangian Relaxation

F. Y. Young, Chris C. N. Chu, W. S. Luk, and Y. C. Wong

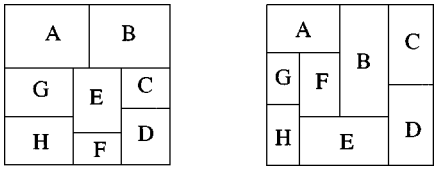
***Abstract*—In the early stage of floorplan design, many modules have large flexibilities in shape (soft modules). Handling soft modules in general nonslicing floorplan is a complicated problem. Many previous works have attempted to tackle this problem using heuristics or numerical methods, but none of them can solve it optimally and efficiently. In this paper, we show how this problem can be solved optimally by geometric programming using the Lagrangian relaxation technique. The resulting Lagrangian relaxation subproblem is so simple that the optimal size of each module can be computed in linear time. We implemented this method in a simulated annealing framework based on the sequence pair representation. The geometric program is invoked in every iteration of the annealing process to compute the optimal size of each module to give the best packing. The execution time is much faster (at least 15 times faster for data sets with more than 50 modules) than that of the most updated previous work by Murata and Kuh (1998). For a benchmark data with 49 modules, we take 3.7 h in total for the whole annealing process using a 600-MHz Pentium III processor while the convex programming approach described by Murata and Koh needs seven days using a 250-MHz DEC Alpha. Our technique will also be applicable to other floorplanning algorithms that use constraint graphs to find module positions in the final packing.**

***Index Terms*—Floorplanning, Lagrangian relaxation, nonslicing, physical design, shaping.**

I. Introduction

FLOORPLANNING has become increasingly important in physical design of very large scale integrated circuits due to the advance in the deep submicrometer technology. Many floorplanning algorithms were proposed in recent years and many of them make use of constraint graphs to compute module positions in the final packing. Unfortunately, it is not known how shape flexibilities of soft modules can be handled efficiently using constraint graphs. This is an important problem since soft modules are common in the floorplanning stage when many designs are not yet done in details. Some previous works [4], [8], [9], [12] have attempted to tackle this problem but none of them succeeded in obtaining the optimal solution efficiently.

There are two types of floorplans: slicing and nonslicing. A slicing floorplan is a floorplan that can be obtained by recursively cutting rectangles horizontally or vertically. A nonslicing floorplan is one that is not restricted to be slicing. Fig. 1 shows an example of each. Nonslicing floorplans are a more general representation that can describe all kinds of packings. However, slicing floorplans have an important advantage over nonslicing: there are efficient algorithms to handle soft modules in slicing floorplans optimally. A well-known approach by Wong *et al.* [13] uses shape curve representation. A shape curve can describe all possible shapes of a module and these shape curves can be added up horizontally or vertically to produce new shape curves for supermodules containing more than one basic modules. Moh *et al.* [5] and Wang *et al.* [11] use numerical optimization methods. Moh *et al.* [5] formulate the problem as a geometric programming and find its global minimum using some standard convex optimization techniques. However, all these methods are limited to placement topology of rectangular dissection only, i.e., slicing.



(a) (b)

Fig. 1. (a) Slicing and (b) nonslicing floorplan.

The problem of handling soft modules becomes more complicated in nonslicing floorplans. Both Pan *et al.* [9] and Wang *et al.* [12] try to generalize Stockmeyer's algorithm [10] to nonslicing floorplan. Kang *et al.* [4] extend the bounded sliceline grid (BSG) method [8] to handle soft modules using heuristics. These methods are either suboptimal or applicable to some specific nonslicing structures only. Murata *et al.* [7] follow the framework of [5] and try to reduce the number of variables and functions when formulating the problem so as to improve the efficiency. However, the execution time of their method to find an exact solution is still very long. It takes seven days to pack a benchmark data with 49 modules.

In this paper, we will present an efficient method to handle shape flexibilities of soft modules in general nonslicing floorplans optimally. The problem is formulated as a geometric program, but we use the Lagrangian relaxation technique [6], a general technique for constrained nonlinear optimization, to solve the problem efficiently. This technique transforms the problem into a sequence of subproblems called Lagrangian relaxation subproblems. Each subproblem can be significantly simplified by the Kuhn–Tucker conditions. The resulting subproblem is so simple that the size of each module can be computed in linear time. This complexity can be further reduced to a constant on average by using a different representation for nonslicing floorplans that supports planar constraint graphs.

We implemented this method in a simulated annealing framework using the sequence pair representation. The objective of the annealing process is to minimize the total packing area and interconnect cost. To evaluate the area in each iteration of the annealing process, we use the geometric program to compute the optimal packing area taking into account the shape flexibilities of all the soft modules simultaneously. Our floorplanner can pack much faster than the most updated previous work [7]. For the benchmark data with 49 modules, we take only 3.7 h in total for the whole annealing process using a 600-MHz Pentium III processor while the convex programming approach in [7] needs seven days using a 250-MHz DEC Alpha. Our method will also be applicable to other floorplanning algorithms that make use of constraint graphs to compute module positions in the final packing.

The rest of this paper is organized as follow. We will formulate the problem in Section II. Section III describes briefly the sequence pair representation. We will formulate the geometric program in Section IV. In Section V, we will explain in details the Lagrangian relaxation technique. Experimental results will be shown in Section VI and some remarks will be given in the last section.

II. Problem Formulation

We consider two kinds of modules: hard modules and soft modules. A hard module is a module whose dimension is fixed. A soft module is one whose area is fixed, but its dimension can be changed as long as its aspect ratio, i.e., the ratio of height to width, is within

Manuscript received November 11, 2000. 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.

C. C. N. Chu is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011 USA.

W. S. Luk and Y. C. Wong are with the Synopsys, Inc., Mountain View, CA 94043 USA.

Publisher Item Identifier S 0278-0070(01)03075-5.

a given range. In this problem, we are given n modules of areas $A_1, A_2, \ldots, A_n$ and their aspect ratio ranges $[r_{1,\min}, r_{1,\max}]$, $[r_{2,\min}, r_{2,\max}]$, $\ldots$, $[r_{n,\min}, r_{n,\max}]$. In case of a hard module, the maximum and minimum aspect ratio will be the same.

A *packing* of a set of modules is a nonoverlap placement of the modules. A *feasible packing* is a packing such that the widths and heights of the modules are consistent with their aspect ratio constraints and area constraints. We measure the area of a packing as the area of the smallest rectangle enclosing all the modules.

We are also given the netlist information: $net_1, net_2, \ldots, net_m$ and the relative positions of the input–output (I/O) pins $p_1, p_2, \ldots, p_q$ along the boundary of the chip. For each net net_i, where $1 \leq i \leq m$, we are given its weight, the I/O pin, and the set of modules to which it is connected. Our objective is to obtain a feasible packing minimizing the total packing area and interconnect cost. We use the simulated annealing technique (based on the sequence pair representation) to search the solution space. For each intermediate solution in the annealing process, we evaluate the packing by computing a linear function of its area and interconnect cost. However, there can be many realizations of the same packing due to the shape flexibilities of the soft modules. The most important contribution of our paper is that we devised an efficient method to compute the shapes of the soft modules to give the optimal packing. The problem is formulated as follows.

Problem Floorpolan Area Minimization (FP/AM): Given a set of hard and soft modules with area and aspect ratio constraints, and a specific packing topology of these modules described by a pair of vertical and horizontal constraint graphs, find the optimal shape of each module so as to produce the smallest possible feasible packing taking into consideration the shape flexibilities of all the soft modules simultaneously.

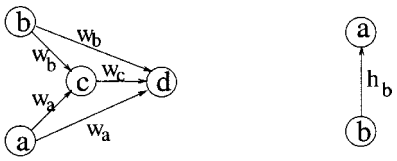
III. Sequence Pair and Constraint Graph

We use sequence pair to represent a general floorplan in the annealing process. A sequence pair of a set of module is a pair of combinations of the module names. For example, $s = (abcd, bacd)$ is a sequence pair of the module set $\{a, b, c, d\}$. We can derive the relative positions between the modules from a sequence pair s by the following rules.

1) *H-constraint:* If $s = (\cdots a \cdots b \cdots, \cdots a \cdots b \cdots)$, module b is on the right hand side of module a.
2) *V-constraint:* If $s = (\cdots a \cdots b \cdots, \cdots b \cdots a \cdots)$, module b is below module a.

We can use constraint graphs to represent these horizontal and vertical placement relationships. A horizontal (vertical) constraint graph G_h (G_v) for a set of n modules is a graph of n vertices with the vertices representing the modules and the edges representing the horizontal (vertical) placement constraints. For example, if module b is on the right-hand side of module a, we will add an edge from a to b in the horizontal constraint graph with a weight equal to the width of a. The reason is that if b is on the right hand side of a, its lower left corner (notice that we always refer the position of a module by the coordinates its lower left corner) should be at a distance of at least the width of a from the lower left corner of a. Similarly, if module b is above module a, we will add an edge from a to b in the vertical constraint graph with a weight equal to the height of a. We can build these graphs directly from a sequence-pair representation.

1) Add an edge from a to b labeled w_a to the horizontal constraint graph G_h where w_a is the width of a iff $s = (\cdots a \cdots b \cdots, \cdots a \cdots b \cdots)$.
2) Add an edge from b to a labeled h_b to the vertical constraint graph G_v where h_b is the height of b iff $s = (\cdots a \cdots b \cdots, \cdots b \cdots a \cdots)$.



(a) (b)

Fig. 2. (a) Horizontal and (b) vertical constraint graphs for the sequence pair $(abcd, bacd)$.

Fig. 2 shows the horizontal and vertical constraint graphs for the sequence pair $s = (abcd, bacd)$. In this example, the orders of a and b in the two sequences are different $(\cdots a \cdots b \cdots, \cdots b \cdots a \cdots)$, so a is above b and there is an edge from b to a labeled h_b in the vertical constraint graph. For modules a and c, their orders are the same in both sequences $(\cdots a \cdots c \cdots, \cdots a \cdots c \cdots)$, so c is on the right-hand side of a and there is an edge from a to c labeled w_a in the horizontal constraint graph. In this way, we can construct the horizontal and vertical constraint graphs by looking at the orders of every pair of modules in the two sequences. In the annealing process, we can modify a sequence pair by two kinds of moves:

1) **M1:** exchange two modules in the first sequence only;
2) **M2:** exchange two modules in both sequences.

These two moves are sufficient to transform any sequence pair α to any other arbitrary sequence pair β in one or more steps.

IV. Formulation of the Geometric Program

We are given n modules $M_1, M_2, \ldots, M_n$ of areas $A_1, A_2, \ldots, A_n$. For each module M_i, where $1 \leq i \leq n$, its minimum and maximum aspect ratios are $r_{i,\min}$ and $r_{i,\max}$, respectively. The minimum and maximum width of M_i are, thus, $L_i = \sqrt{A_i / r_{i,\max}}$ and $U_i = \sqrt{A_i / r_{i,\min}}$, respectively. We are also given the topology of the packing described by a pair of horizontal and vertical constraint graphs. Let x_i denote the smallest x position of the lower left corner of module i satisfying all the horizontal constraints in the horizontal constraint graph G_h. Similarly, y_i denotes the smallest y position of the lower left corner of module i, satisfying all the vertical constraints in the vertical constraint graph G_v. Then, for each edge $e(i, j)$ from module i to module j in G_h, we have the following constraint:

$$x_i + w_i \leq x_j$$

where w_i is the width of module i. Similarly, for each edge $e(i, j)$ from module i to module j in G_v, we have the following constraint:

$$y_i + \frac{A_i}{w_i} \leq y_j.$$

In the horizontal constraint graph G_h, we denote the set of sources and sinks by s_h and t_h, respectively, where a source is a vertex without incoming edge and a sink is a vertex without outgoing edge. Similarly, we use s_v and t_v to denote the set of sources and sinks in G_v, respectively. Then, for each module i in s_h

$$x_i = 0$$

and for each module i in s_v

$$y_i = 0.$$

For simplicity, we add one dummy vertex labeled $n + 1$ to each G_h and G_v. The dummy vertex in G_h and G_v represents the rightmost and the topmost boundary of the chip, respectively. Edge $e(i, n + 1)$ with weight w_i is added to G_h for each $i \in t_h$ because the rightmost

chip boundary should be at a distance of at least w_i from each module $i \in t_h$. Similarly, $e(i, n+1)$ with weight A_i/w_i is added to G_v for each $i \in t_v$. From now onwards, we assume that the constraint graphs G_h and G_v contain these additional vertices and edges. The problem can be formulated as the following geometric programming primal problem (PP):

$$\begin{aligned}
&\text{minimize} && x_{n+1}y_{n+1} && \\
&\text{subject to} && x_i + w_i \le x_j \quad \forall e(i,j) \in G_h && \text{(A)}\\
& && y_i + \frac{A_i}{w_i} \le y_j \quad \forall e(i,j) \in G_v && \text{(B)}\\
& && L_i \le w_i \le U_i \quad \forall 1 \le i \le n. && \text{(C)}
\end{aligned}$$

V. Lagrangian Relaxation

According to the Lagrangian relaxation procedure, we can introduce nonnegative multipliers, called Lagrange multipliers, to the constraints in order to get rid of those difficult constraints and incorporate them into the objective function. Let $\lambda_{i,j}$ denote the multiplier for the constraint $x_i + w_i \le x_j$ in (A) and $\mu_{i,j}$ denote the multiplier for the constraint $y_i + A_i/w_i \le y_j$ in (B). Let $\vec{\lambda}$ and $\vec{\mu}$ be vectors of all the Lagrange multipliers introduced to the constraints in (A) and (B), respectively. Then, the Lagrangian relaxation subproblem associated with the multiplier $\vec{\lambda}$ and $\vec{\mu}$, denoted by $LRS/(\vec{\lambda}, \vec{\mu})$, becomes

$$\begin{aligned}
&\text{minimize} && x_{n+1}y_{n+1} + \\
& && \sum_{e(i,j)\in G_h} \lambda_{i,j}(x_i + w_i - x_j) + \\
& && \sum_{e(i,j)\in G_v} \mu_{i,j}\left(y_i + \frac{A_i}{w_i} - y_j\right)\\
&\text{subject to} && L_i \le w_i \le U_i \qquad \forall 1 \le i \le n.
\end{aligned}$$

Let $Q(\vec{\lambda}, \vec{\mu})$ denote the optimal value of the problem $LRS/(\vec{\lambda}, \vec{\mu})$. We define the Lagrangian dual problem (LDP) of PP as follows:

$$\begin{aligned}
&\text{maximize} && Q(\vec{\lambda}, \vec{\mu})\\
&\text{subject to} && \vec{\lambda} \ge 0 \text{ and } \vec{\mu} \ge 0.
\end{aligned}$$

Since PP can be transformed into a convex problem [7], we can apply [6, theorem 6.2.4] and imply that if $(\vec{\lambda}, \vec{\mu})$ is the optimal solution to LDP, the optimal solution of $LRS/(\vec{\lambda}, \vec{\mu})$ will also optimize PP.

A. Simplification of the Lagrangian Relaxation Subproblem

The Lagrangian relaxation subprogram $LRS/(\vec{\lambda}, \vec{\mu})$ can be greatly simplified by the Kuhn–Tucker conditions. Consider the Lagrangian ζ of PP [6]

$$\begin{aligned}
\zeta &= x_{n+1}y_{n+1} + \sum_{e(i,j)\in G_h} \lambda_{i,j}(x_i + w_i - x_j)\\
&\quad + \sum_{e(i,j)\in G_v} \mu_{i,j}(y_i + \frac{A_i}{w_i} - y_j) + \sum_{1\le i\le n} u_i(L_i - w_i)\\
&\quad + \sum_{1\le i\le n} v_i(w_i - U_i)\\
&= x_{n+1}y_{n+1} - \sum_{e(i,n+1)\in G_h} \lambda_{i,n+1}x_{n+1}\\
&\quad - \sum_{e(i,n+1)\in G_v} \mu_{i,n+1}y_{n+1}\\
&\quad + \sum_{1\le i\le n}\left(\sum_{e(i,j)\in G_h}\lambda_{i,j} - \sum_{e(j,i)\in G_h}\lambda_{j,i}\right)x_i\\
&\quad + \sum_{1\le i\le n}\left(\sum_{e(i,j)\in G_v}\mu_{i,j} - \sum_{e(j,i)\in G_v}\mu_{j,i}\right)y_i\\
&\quad + \sum_{1\le i\le n}\left(\left(\sum_{e(i,j)\in G_h}\lambda_{i,j}\right)w_i + \left(\sum_{e(i,j)\in G_v}\mu_{i,j}\right)\frac{A_i}{w_i}\right)\\
&\quad + \sum_{1\le i\le n} u_i(L_i - w_i) + \sum_{1\le i\le n} v_i(w_i - U_i).
\end{aligned}$$

The Kuhn–Tucker conditions imply that $\partial\zeta/\partial x_i = 0$ and $\partial\zeta/\partial y_i = 0$ for all $1 \le i \le n+1$ at the optimal solution of PP. Therefore, in searching for the $\vec{\lambda}$ and $\vec{\mu}$ to optimize LDP, we only need to consider those multipliers such that these conditions are satisfied. Therefore, for all $1 \le i \le n$

$$\begin{aligned}
\partial\zeta/\partial x_i &= \sum_{e(i,j)\in G_h}\lambda_{i,j} - \sum_{e(j,i)\in G_h}\lambda_{j,i} = 0\\
\partial\zeta/\partial y_i &= \sum_{e(i,j)\in G_v}\mu_{i,j} - \sum_{e(j,i)\in G_v}\mu_{j,i} = 0
\end{aligned}$$

and

$$\begin{aligned}
\partial\zeta/\partial x_{n+1} &= y_{n+1} - \sum_{e(i,n+1)\in G_h}\lambda_{i,n+1} = 0\\
\partial\zeta/\partial y_{n+1} &= x_{n+1} - \sum_{e(i,n+1)\in G_v}\mu_{i,n+1} = 0.
\end{aligned}$$

Rearrange

$$\sum_{e(j,i)\in G_h}\lambda_{j,i} = \sum_{e(i,j)\in G_h}\lambda_{i,j} \tag{1}$$

$$\sum_{e(j,i)\in G_v}\mu_{j,i} = \sum_{e(i,j)\in G_v}\mu_{i,j} \tag{2}$$

and

$$y_{n+1} = \sum_{e(i,n+1)\in G_h}\lambda_{i,n+1} \tag{3}$$

$$x_{n+1} = \sum_{e(i,n+1)\in G_v}\mu_{i,n+1}. \tag{4}$$

We use Ω to denote the set of $(\vec{\lambda}, \vec{\mu})$ satisfying the above relationships (1)–(4) for a given pair of horizontal and vertical constraint graphs. If $(\vec{\lambda}, \vec{\mu}) \in \Omega$, the objective function F of $LRS/(\vec{\lambda}, \vec{\mu})$ becomes

$$\begin{aligned}
F = &\sum_{1\le i\le n}\left(\left(\sum_{e(i,j)\in G_h}\lambda_{i,j}\right)w_i + \left(\sum_{e(i,j)\in G_v}\mu_{i,j}\right)\frac{A_i}{w_i}\right)\\
&- \left(\sum_{e(i,n+1)\in G_h}\lambda_{i,n+1}\right)\left(\sum_{e(i,n+1)\in G_v}\mu_{i,n+1}\right)
\end{aligned}$$

where $(\sum_{e(i,n+1)\in G_h}\lambda_{i,n+1})(\sum_{e(i,n+1)\in G_v}\mu_{i,n+1})$ is a constant for a fixed $(\vec{\lambda}, \vec{\mu})$.

B. Solving $LRS/(\vec{\lambda}, \vec{\mu})$

In this section, we consider solving the Lagrangian relaxation subproblem $LRS/(\vec{\lambda}, \vec{\mu})$ when $(\vec{\lambda}, \vec{\mu}) \in \Omega$, i.e., computing w_i for $1 \leq i \leq n$. F can be written as

$$F = k + \sum_{1 \leq i \leq n} \left(\left(\sum_{e(i,j) \in G_h} \lambda_{i,j} \right) w_i + \left(\sum_{e(i,j) \in G_v} \mu_{i,j} \right) \frac{A_i}{w_i} \right)$$

where

$$k = - \left(\sum_{e(i,n+1) \in G_h} \lambda_{i,n+1} \right) \left(\sum_{e(i,n+1) \in G_v} \mu_{i,n+1} \right)$$

is a constant. Differentiate F with respect to w_i in order to get the optimal value of w_i to minimize F

$$\frac{\partial F}{\partial w_i} = 0$$

$$\sum_{e(i,j) \in G_h} \lambda_{i,j} - \frac{A_i}{w_i^2} \sum_{e(i,j) \in G_v} \mu_{i,j} = 0$$

$$w_i = \sqrt{\frac{A_i \times \sum_{e(i,j) \in G_v} \mu_{i,j}}{\sum_{e(i,j) \in G_h} \lambda_{i,j}}}.$$

Recall that w_i must lie within the range $[L_i, U_i]$. Let w_i^* denote $\sqrt{(A_i \times \sum_{e(i,j) \in G_v} \mu_{i,j}) / \sum_{e(i,j) \in G_h} \lambda_{i,j}}$. Since $\partial F / \partial w_i$ is positive for $w_i < w_i^*$ and negative for $w_i > w_i^*$, the optimal w_i can be computed as

$$w_i = \min\{U_i, \max\{L_i, w_i^*\}\}.$$

The total time to compute the widths of all the modules are $O(|E_h| + |E_v|)$, where $|E_h|$ and $|E_v|$ are the numbers of edges in the horizontal and vertical constraint graphs, respectively. The algorithm *Find-Width* below outlines the steps to solve $LRS/(\vec{\lambda}, \vec{\mu})$.

```
Algorithm Find-Width
/* This algorithm solves LRS/(λ, μ) optimally given (λ, μ) ∈ Ω */
Input: Areas A_1, ..., A_n
       Lower bounds of widths L_1, L_2, ..., L_n
       Upper bounds of widths U_1, U_2, ..., U_n
       Constraint graphs G_v and G_h
       Lagrange multipliers (λ, μ) ∈ Ω
Output: Widths w_1, w_2, ..., w_n
1. For i = 1 to n
2.     sum_1 = sum_2 = 0
3.     For all e(i, j) ∈ G_h
4.       Compute sum_1 = sum_1 + λ_{i,j}
5.     For all e(i, j) ∈ G_v
6.       Compute sum_2 = sum_2 + μ_{i,j}
7.     If (sum_1 ≠ 0) and (sum_2/sum_1 ≥ 0)
7.       Compute w* = sqrt(A_i * sum_2/sum_1)
8.       w_i = min{U_i, max{L_i, w*}}.
```

C. Solving LDP

As explained above, we only need to consider those $(\vec{\lambda}, \vec{\mu}) \in \Omega$ in order to maximize $Q(\vec{\lambda}, \vec{\mu})$ in the LDP problem. We used a subgradient optimization method to search for the optimal $(\vec{\lambda}, \vec{\mu})$. Starting from an arbitrary $(\vec{\lambda}, \vec{\mu}) \in \Omega$ in step k, we will move to a new pair $(\vec{\lambda'}, \vec{\mu'})$ by following the subgradient direction:

$$\lambda'_{i,j} = [\lambda_{i,j} + \rho_k (x_i + w_i - x_j)]^+$$

$$\mu'_{i,j} = \left[\mu_{i,j} + \rho_k \left(y_i + \frac{A_i}{w_i} - y_j \right) \right]^+$$

where

$$[x]^+ = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

and ρ_k is a step size such that $\lim_{k \to \infty} \rho_k = 0$ and $\sum_{k=1}^{\infty} \rho_k = \infty$. After updating $\vec{\lambda}$ and $\vec{\mu}$, we will *project* $(\vec{\lambda'}, \vec{\mu'})$ back to the nearest point $(\vec{\lambda^*}, \vec{\mu^*})$ in Ω and solve the Lagrangian relaxation subproblem $LRS/(\vec{\lambda^*}, \vec{\mu^*})$ using the method described in Section V-B. This procedure is repeated until the solution converges. The following algorithm summarizes the steps to solve LDP.

```
Algorithm Solve-LDP
/* This algorithm solves the LDP problem optimally. Given the placement topology described by a pair of constraint graphs, it computes the optimal values for the widths of the modules to minimize the total packing area. */
Input: Areas A_1, A_2, ..., A_n
       Lower bounds of widths L_1, L_2, ..., L_n
       Upper bounds of widths U_1, U_2, ..., U_n
       Constraint graphs G_v and G_h
Output: Widths w_1, w_2, ..., w_n
1. Initialize (λ, μ) and ρ_1
2. k = 1
3. Repeat
4.     Call Find-width() to solve LRS(λ, μ)
5.     Compute (x_i, y_i) ∀ 1 ≤ i ≤ n+1 using the
       longest path algorithm
6.     Compute λ'_{i,j} = [λ_{i,j} + ρ_k(x_i + w_i −
       x_j)]^+ ∀ e(i, j) ∈ G_h
7.     Compute μ'_{i,j} = [μ_{i,j} + ρ_k(y_i + A_i/w_i −
       y_j)]^+ ∀ e(i, j) ∈ G_v
8.     Project (λ', μ') to (λ*, μ*) such that
       (λ*, μ*) ∈ Ω
9.     k = k + 1
10.    (λ, μ) = (λ*, μ*)
11. Until w_i's converge.
```

D. Projection

As described above, we used subgradient optimization to search for the optimal $(\vec{\lambda}, \vec{\mu})$. Starting from an arbitrary $(\vec{\lambda}, \vec{\mu}) \in \Omega$, we will move to a new pair $(\vec{\lambda'}, \vec{\mu'})$ by following the subgradient direction. $(\vec{\lambda'}, \vec{\mu'})$ will then be projected back to the nearest point $(\vec{\lambda^*}, \vec{\mu^*})$ in Ω based on the two-norm measure. This projection step is done by finding an orthonormal bases $\vec{\lambda_1}, \ldots, \vec{\lambda_p}, \vec{\mu_1}, \ldots, \vec{\mu_q}$ of Ω. Then

$$\vec{\lambda^*} = \sum_{i=1}^{p} (\vec{\lambda'} \cdot \vec{\lambda_i}) \vec{\lambda_i} \tag{5}$$

$$\vec{\mu^*} = \sum_{i=1}^{q} (\vec{\mu'} \cdot \vec{\mu_i}) \vec{\mu_i}. \tag{6}$$

To find the orthonormal bases spanning Ω, we first find a set I of independent vectors spanning Ω using QR decomposition. For simplicity,

we consider λ's only in the following discussion. Let Ω_λ denote the set of $\vec{\lambda}$'s satisfying the relationships (1) and (3) and let

$$Q_\lambda \vec{\lambda} = \vec{y}$$

be the system of equations described by (1) and (3). By QR decomposition, we can write each dependent variable λ_i in $\vec{\lambda}$ as a linear combination of the other independent variables λ_j's in $\vec{\lambda}$

$$\lambda_i = \sum_j \alpha_{i,j} \lambda_j .$$

From these formulae, we can obtain a set of independent vectors I_λ spanning Ω_λ. Notice that in (1)–(4), each variable will appear at most twice and their coefficients are either 1 or -1, so the QR decomposition step takes only $O(n^2)$ time instead of $O(n^3)$, where n is the total number of modules and there is no floating point division throughout the whole process. Then we apply the Gram–Schmidt process [2] to obtain the orthonormal bases from I_λ.

```
Algorithm Gram-Schmidt
Input: An independent set v_1, ..., v_m ∈ R^p
Output: An orthonormal set q_1, ..., q_m ∈ R^p
  such that the set q_1, ..., q_m spans the
  same space as v_1, ..., v_m
1. For k = 1, ..., m
2.     For i = 1, ..., k-1 (skip when k = 1)
3.         r_ik = v_k · v_i
4.         v_k = v_k - r_ik v_i
5.     r_kk = ||v_k||_2
6.     v_k = (1/r_kk) v_k .
```

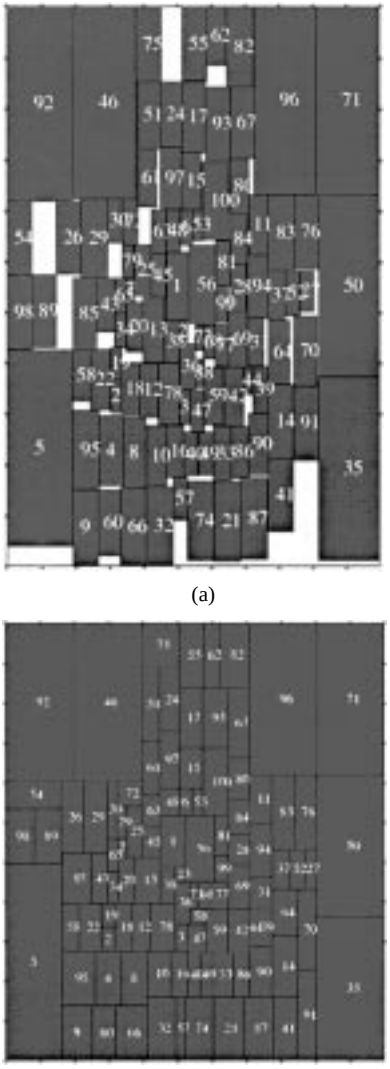
The Gram–Schmidt Algorithm takes $O(|E|^3)$, where $|E|$ is the number of edges in the constraint graph. Fortunately, we only need to do the QR decomposition and Gram–Schmidt process once for each sequence pair. After finding an orthonormal set of vector, we can repeatedly use this set to do projection in searching for an optimal $(\vec{\lambda}, \vec{\mu}) \in \Omega$ according to (5) and (6). Another useful incremental technique to improve the efficiency is due to the observation that the structures of the constraint graphs are unchanged if we just exchange two modules in a move of the annealing process (M2), so we do not need to recompute the orthonormal bases in almost half of the iterations.

VI. Experimental Results

We tested our floorplanner with the MCNC benchmarks and some randomly generated data sets using a 600-MHz Pentium III processor. In all the experiments, the weightings between the area term and the wirelength term in the cost function of the annealing process are approximately balanced. We did three sets of experiments. In the first set, we want to know the speed and quality of sizing all the modules once by the Lagrangian relaxation method. We randomly generated six data sets with 10 to 500 modules each. The aspect ratio of each module can range from 0.1 to 10.0 and the areas of the modules are randomly generated in the range between 0 and 500 000. The sizing procedure is applied only once at the end of the annealing process and the chip aspect ratio can range between 0.5 to 2.0. The result is shown in Table I. Notice that the result for each data set is obtained by repeating the experiment six times and picking the best one. Fig. 3 shows the packings for the data set with 100 modules before and after the sizing procedure. Murata and Kuh [7] have also reported the speed and quality of their method on data set with module size randomly generated in the range between 100^2 to $10\,000^2$ running on a 250-MHz Alpha DEC and their results is shown in Table II.

TABLE I
Speed and Quality of the Sizing Procedure

#Module	Deadspace (%) Before Sizing	Deadspace (%) After Sizing	Time (sec)
10	9	0	0.35
20	7	0	1.38
50	9	0	2.84
100	11	0	8.90
200	11	1	148.8
500	14	3	4697.1



(a)

(b)

Fig. 3. Packings of 100 modules (a) before and (b) after one sizing step.

In the second set of experiments, we apply the sizing procedure in every iteration of the annealing process. We use the same set of parameters as in [7]: the initial temperature is decided such that the acceptance ratio is 95%, the temperature is exponentially lowered in four decades by 20 steps, the number of iterations in one temperature step is ten times the number of modules, and the aspect ratio of the whole chip is approximately one. The temperature drops until it is below a certain threshold (1×10^{-10}). We test our method using the benchmark data sets and the aspect ratio of the modules can range between 0.1 to 10.0. The results is shown in Table III. Note that our experiments are performed on a 600-MHz Pentium III processor while [7] used a 250-MHz DEC Alpha processor. Fig. 4 shows a result packing for ami33.

TABLE II
RESULTS FROM [7]

#Module	Deadspace (%) Before Sizing	Deadspace (%) After Sizing	Time (sec)
10	26	1	0.396
20	11	0	4.93
50	9	1	60.5
100	7	1	937
200	7	1	7140
500	9	2	73834

TABLE III
RESULTS OF APPLYING THE SIZING PROCEDURE IN EVERY ITERATION OF THE ANNEALING PROCESS

		Our Method				[7]
Data	n	Deadspace (%)	Time (sec)	No. of Iterations	Time per Iteration (10^{-3} sec)	Time (sec)
apte	9	0.05	53.8	30872	1.7	1198
xerox	10	0.47	79.0	33802	2.3	789
hp	11	1.3	129.9	36742	3.5	1346
ami33	33	1.6	2622.5	110222	23.8	75684
ami49	49	4.4	13200.9	166602	79.2	612103

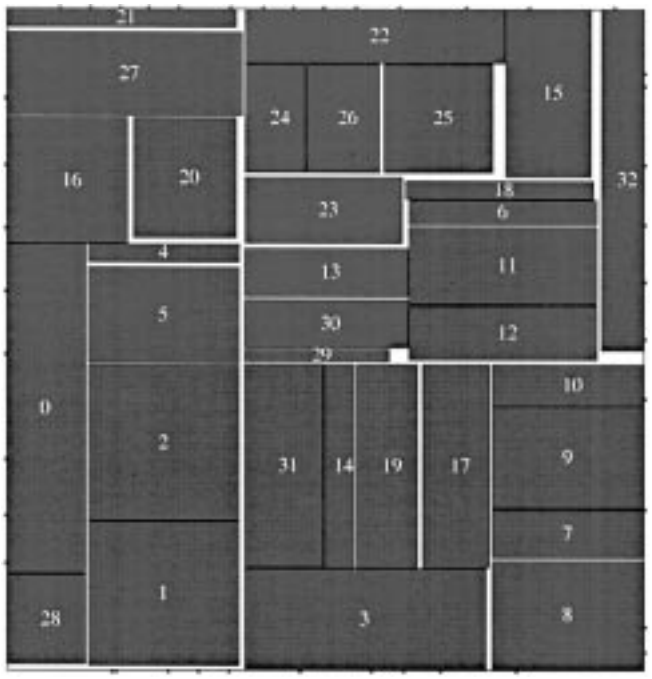


Fig. 4. Result packing of ami33 with aspect ratio bound [0.1, 10.0]. It has 1.6% deadspace.

In the last set of experiments, we also use the benchmark data sets and invoke the sizing procedure in every iteration of the annealing process. However, we allow the aspect ratio of each module to range from 0.5 to 2.0. This is a more reasonable range and it can better demonstrate the speed and quality of the sizing method in practice. In this set of experiments, the initial temperature is decided such that the acceptance ratio is 95%. The aspect ratio of the whole chip is also approximately one. The temperature is lowered at a constant rate of 0.95 until it is below a certain threshold (1×10^{-10}) and the number of iterations at each temperature step is a constant of 30. The results are shown in Table IV.

TABLE IV
RESULTS OF TESTING WITH THE BENCHMARK DATA USING ASPECT RATIO BOUND [0.5, 2.0]

Data	n	Deadspace (%)	Time (sec)	No. of Iterations	Time per Iteration (10^{-3} sec)
apte	9	0.54	53.0	29072	1.8
xerox	10	0.4	71.6	28742	2.5
hp	11	1.4	107.3	28292	3.8
ami33	33	4.3	774.6	28382	27.3
ami49	49	7.7	2354.0	28982	81.2

VII. REMARKS

Our method can also be used in the presence of hard rectilinear blocks. This can be done by partitioning a rectilinear hard block into several rectangular submodules and keeping them together as one piece by inserting additional edges in the constraint graphs. In this way, we can still shape the soft modules optimally in the presence of hard blocks.

In our current implementation, the time taken to compute the width of a module i is linear to the total number of outgoing edges from i in the two constraint graphs. This is $O(n)$ on average for constraint graphs constructed from the sequence pair representation. However, this can be reduced to $O(1)$ by using another representation, e.g., O-tree [1] and B*-tree [3], which supports planar constraint graphs.

ACKNOWLEDGMENT

The authors would like to thank Prof. M. D. F. Wong for his kindness of providing us the source code for the sequence pair floorplanning algorithm.

REFERENCES

[1] Y. C. Chang, Y. W. Chang, G. M. Wu, and S. W. Wu, "B*-Trees: A new representation for nonslicing floorplans," in *Proc. 37th ACM/IEEE Design Automation Conf.*, June 2000, pp. 458–463.
[2] D. S. Watkins, *Fundamentals of Matrix Computations*, 1st ed. New York: Wiley, 1991.
[3] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-tree representation of nonslicing floorplan and its applications," in *Proc. 36th ACM/IEEE Design Automation Conf.*, June 1999, pp. 268–273.
[4] M. Kang and W. W. M. Dai, "General floorplanning with L-shaped, T-shaped and soft blocks based on bounded slicing grid structure," in *Proc. IEEE Asia South Pacific Design Automation Conf.*, Jan. 1997, pp. 265–270.
[5] T.-S. Moh, T.-S. Chang, and S. L. Hakimi, "Globally optimal floorplanning for a layout problem," *IEEE Trans. Circuit Syst. I*, vol. 43, pp. 713–720, Sept. 1996.
[6] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, 2nd ed. New York: Wiley, 1997.
[7] H. Murata and E. S. Kuh, "Sequence-pair based placement method for hard/soft/preplaced modules," in *Proc. Int. Symp. Physical Design*, Apr. 1998, pp. 167–172.
[8] S. Nakatake, K. Fujiyoushi, H. Murata, and Y. Kajitani, "Module placement based on BSG-structure and IC layout applications," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1996, pp. 484–491.
[9] P. Pan and C. L. Liu, "Area minimization for floorplans," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 129–132, Jan. 1995.
[10] L. Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," *Inform. Control*, vol. 59, no. 2, pp. 91–101, May 1983.
[11] T.-C. Wang and D. F. Wong, "An optimal algorithm for floorplan area optimization," in *Proc. ACM/IEEE Design Automation Conf.*, June 1990, pp. 180–186.
[12] ——, "Optimal floorplan area optimization," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 992–1001, Aug. 1992.
[13] 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.