An Optimal Algorithm for Floorplan Area Optimization*

Ting-Chi Wang and D.F. Wong
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

ABSTRACT. In this paper we present an optimal algorithm for the floorplan area optimization problem. Our algorithm is based on an extension of the technique in [5]. Experimental results indicate that our algorithm is efficient and capable of successfully handling large floorplans. We compare our algorithm with the branch-and-bound optimal algorithm in [6]. The running time of our algorithm is substantially less than that of [6]. For several examples where the algorithm in [6] ran for days and did not terminate, our algorithm produced optimal solutions in a few seconds.

1 Introduction

Floorplan design [4,7] is an important step in the physical design of VLSI circuits. A typical approach to floorplan design is to first determine the topology (i.e., relative positions of the modules) of the floorplan primarily using the interconnection information among the modules [3,4]. After the topology of the floorplan is determined, various optimizations are then performed on it to minimize some cost measure. If each module is a rectangle with a finite set of implementations, one optimization problem is to select the optimal implementation for each module such that the total area of the floorplan is minimized. This problem is referred to as the *floorplan area optimization problem*. For slicing floorplans, there is an efficient optimal algorithm to solve the floorplan area optimization problem in polynomial time [5]. However, for general non-slicing floorplans this problem has been proven to be NP-complete [5]. Consequently, recent research efforts have been focused on efficient heuristic method [8] and exponential time branch-and-bound algorithm [6].

In this paper we extend the technique used in [5] to obtain an optimal algorithm for a special class of non-slicing floorplans called *hierarchical floorplans of order 5* [8,9]. Our idea can be extended to the general case. Experimental results indicate that our algorithm is an efficient optimal algorithm and capable of successfully handling very large floorplans. We compare our algorithm with Wimer et al's branch-and-bound optimal algorithm [6]. The running time of our algorithm is substantially less than that of [6]. For several examples where the algorithm in [6] ran for days and did not terminate, our algorithm produced optimal solutions in a few seconds.

2 Problem Description

A *floorplan* for m modules consists of an enveloping rectangle subdivided by horizontal and vertical line segments into m non-overlapping rectangles called *basic rectangles*. Each basic rectangle i must be large enough to accommodate the module assigned to it.

An important class of floorplans is the set of all slicing floorplans [4,7]. 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 *wheel* is a non-slicing floorplan of five modules. (See Figure 1.) There are only two possible wheels and they are the simpliest non-slicing floorplans.

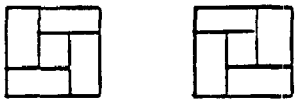

Figure 1: Two possible wheels

A natural extension of slicing floorplans is the set of hierarchical floorplans of order 5 [8,9]. A floorplan is said to be *hierarchical of order* 5 if it can be obtained by recursively partitioning a rectangle into two parts by either a vertical line or a horizontal line or into five parts by a wheel. The hierarchy of the partitioning can be represented by a *floorplan tree*. Figure 2 shows a hierarchical floorplan of order 5 and its floorplan tree. Each leaf in the floorplan tree corresponds to a basic rectangle and each internal node corresponds to a composite rectangle in the floorplan. Note that each internal node in a floorplan tree has degree either two or five.

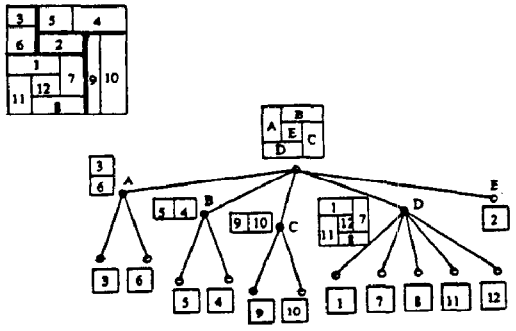

Figure 2

The floorplan area optimization problem considered in this paper can be described as follows. We are given a floorplan tree specifying the topology of a hierarchical floorplan of order 5 for a set of m modules. Each module is a rectangle with a finite set of possible implementations. For each module i, we are given a set of possible implementations of the form $\{w_1 \times h_1, w_2 \times h_2, ..., w_{n_i} \times h_{n_i}\}$, where w_i is the width and h_i is the height of the module. The objective of this problem is to determine the optimal implementation for each module such that the total area of the floorplan is minimized.

*This work was partially supported by the National Science Foundation under grant MIP-8909586, by the Texas Advanced Research Program under grant 4096, and by an IBM Faculty Development Award.

3 Preliminaries

In a floorplan, a *block* is a connected set of basic rectangles. We shall consider two types of blocks: *L-shaped* and *rectangular* blocks. (See Figure 3.) The terms that will be used to denote the various sides of L-shaped and rectangular blocks are shown in Figure 4.

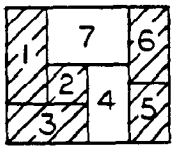

Figure 3: An L-shaped block and a rectangular block

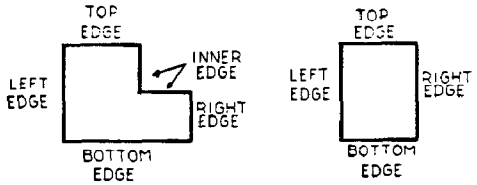

Figure 4

Since each module has a finite number of possible implementations, consequently each block in a floorplan also has different possible implementations. An implementation of an L-shaped block can be represented by a 4-tuple (w^1, w^2, h^1, h^2). Here w^1 and w^2 represent the lengths of the bottom edge and the top edge, respectively, and h^1 and h^2 represent the lengths of the left edge and the right edge, respectively. Besides, the two inequalities: $w^1 \geq w^2$ and $h^1 \geq h^2$ always hold. (See Figure 5.) Clearly, a rectangular block can be considered as a special type of an L-shaped block with $w^1 = w^2 = w$ and $h^1 = h^2 = h$, where w and h are the width and the height of the rectangular block, respectively. (See Figure 5.)

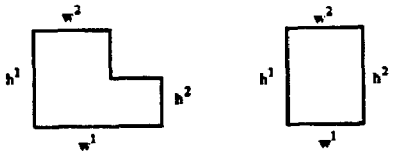

Figure 5

Definition 1 Let $X = (w^1, w^2, h^1, h^2)$ and $Y = (w^{1'}, w^{2'}, h^{1'}, h^{2'})$ be two implementations of a block (L-shaped or rectangular). We say X *dominates* Y if and only if the following 4 inequalities hold:

$$(1)\ w^1 \geq w^{1'};\ (2)\ w^2 \geq w^{2'};\ (3)\ h^1 \geq h^{1'};\ \textit{and}\ (4)\ h^2 \geq h^{2'}$$

If X dominates Y, we say X is a *redundant implementation.* Clearly, redundant implementations of a block are not relevant to the optimal implementation of the floorplan and hence should be pruned as early as possible.

Definition 2 A list $\{(w_1^1, w_1^2, h_1^1, h_1^2), ..., (w_n^1, w_n^2, h_n^1, h_n^2)\}$ is an *R-list* if the following properties $P1$ and $P2$ are satisfied for all j and k, $1 \leq j < k \leq n$.

$$P1:\ w_j^1 = w_j^2 \geq w_k^1 = w_k^2;\quad P2:\ h_j^1 = h_j^2 \leq h_k^1 = h_k^2$$

Definition 3 A list $\{(w_1^1, w_1^2, h_1^1, h_1^2), ..., (w_n^1, w_n^2, h_n^1, h_n^2)\}$ is an *L-list* if the following properties $P3$, $P4$, $P5$ and $P6$ are satisfied for all j and k, $1 \leq j < k \leq n$.

$$P3: w_j^1 \geq w_k^1;\ P4: w_j^2 = w_k^2;\ P5: h_j^1 \leq h_k^1;\ P6: h_j^2 \leq h_k^2$$

Definition 4 An *irreducible R-list* is an R-list in which there are no redundant elements. Similarly, an *irreducible L-list* is an L-list in which there are no redundant elements.

In our algorithm, all non-redundant implementations of a rectangular block are stored in an irreducible R-list; and all non-redundant implementations of an L-shaped block are stored in a set of irreducible L-lists.

4 Algorithm

Let T be a floorplan tree representing a hierarchical floorplan of order 5. Each leaf in T corresponds to a basic rectangle and each internal node in T corresponds to a rectangular block. For each leaf, we can use an irreducible R-list to store all non-redundant implementations of the corresponding basic rectangle. (Note that non-redundant implementations of a basic rectangle are the same as the non-redundant implementations of its corresponding module.) Our algorithm recursively determines an irreducible R-list for each internal node of T in a bottom-up fashion. Each such R-list corresponds to the set of all non-redundant implementations of the block represented by the internal node. After the irreducible R-list associated with the root is obtained, we can scan the list to search for the implementation with minimum area and it will be the optimal implementation of the floorplan.

We now describe how to obtain the irreducible R-list for each internal node of T. Let v be an internal node of T and $v_1, v_2, ..., v_l$ be all the sons of v ($l = 2$ or 5). We can obtain the irreducible R-list of v from the irreducible R-lists of $v_1, v_2, ..., v_l$. According to the number of sons that v has, our algorithm uses different approaches (described in next 2 subsections) to obtain the irreducible R-list for v.

4.1 Slicing

In this case, v corresponds to a slicing cut and has only two sons v_1 and v_2. The algorithm in [5] can be used here to obtain the irreducible R-list for v. Let k_1 and k_2 be the number of elements of the irreducible R-lists associated with v_1 and v_2, respectively. The basic idea in [5] is that it is unnecessary to consider all $k_1 k_2$ possible combinations of implementations for v; instead it was that at most $k_1 + k_2 - 1$ possible combinations are relevant to the optimal floorplan. As a result, the irreducible R-list of non-redundant implementations for v can be obtained by a method similar to merging the two irreducible R-lists associated with v_1 and v_2. The details of this algorithm can be found in [5].

4.2 Wheel

In this case, v corresponds to a wheel and has five sons $v_1, v_2, ..., v_5$. Although there are two possible wheels, we can always think of one as the *reflection* of the other. Hence without loss of generality, we only consider how to obtain non-redundant implementations of the wheel with the form shown in the left part of Figure 1. As for the other wheel shown in Figure 1, we can first reflect it appropriately and then apply the same algorithm.

Let T_v be the subtree rooted in v. We can rearrange T_v into a binary tree T_v' as shown in Figure 6. In T_v', we create three extra nodes u_1, u_2 and u_3; each of them is an internal

node representing an L-shaped block.

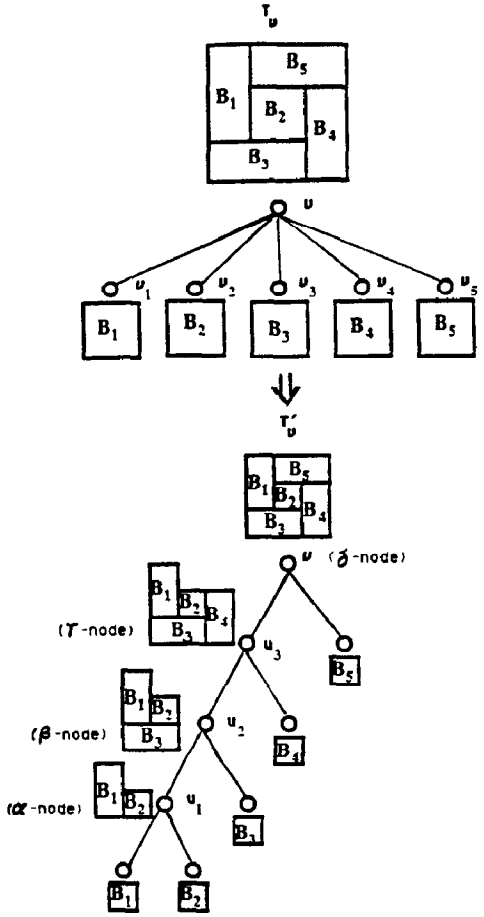

Figure 6

Definition 5 An *α-node* is a node which corresponds to an L-shaped block obtained by combining two rectangular blocks by attaching the left edge of the second rectangular block to the right edge of the first rectangular block.

Definition 6 A *β-node* is a node which corresponds to an L-shaped block obtained by combining an L-shaped block and a rectangular block by attaching the top edge of the rectangular block to the bottom edge of the L-shaped block .

Definition 7 A *γ-node* is a node which corresponds to an L-shaped block obtained by combining an L-shaped block and a rectangular block by attaching the left edge of the rectangular block to the right edge of the L-shaped block .

Definition 8 A *δ-node* is a node which corresponds to a rectangular block obtained by combining an L-shaped block and a rectangular block.

Based on the above definitions, u_1, u_2, u_3 and v are α, β, γ and δ nodes, respectively. (See Figure 6.) We shall construct a set of irreducible L-lists for each nodes u_1, u_2 and u_3, and an irreducible R-list for v. These irreducible lists store all non-redundant implementations for the blocks represented by the nodes. To obtain the irreducible R-list for the root v , we do the following four steps.

Step 1. Obtain a set of irreducible L-lists for u_1.
Step 2. Obtain a set of irreducible L-lists for u_2.
Step 3. Obtain a set of irreducible L-lists for u_3.
Step 4. Obtain an irreducible R-list for v.

As we shall see, during the execution of Steps 2, 3 and 4, our algorithm can prune redundant implementations within the same list by using an extension of the technique for slicing floorplans [5]. Therefore, at the end of Step 4, many implementations that are irrelevant to the optimal floorplan are pruned. Now let us describe the details of the procedures used in Step 1 to Step 4.

There are four basic procedures α-procedure, β-procedure, γ-procedure and δ-procedure. α-procedure, β-procedure and γ-procedure are the procedures to construct a set of irreducible L-lists for an α-node, a β-node and a γ-node, respectively. δ-procedure is a procedure to construct an irreducible R-list for a δ-node. We shall use α-procedure for Step 1, β-procedure for Step 2, γ-procedure for Step 3, and δ-procedure for Step 4. We now briefly describe these four basic procedures.

4.2.1 α-procedure

Suppose d is an α-node. According to Definition 5, its children d_1 and d_2 represent two rectangular blocks. Let L_{d_1} and L_{d_2} be the irreducible R-lists constructed for d_1 and d_2. The α-procedure constructs $|L_{d_1}|$ irreducible L-lists $\{L_d^1, L_d^2, ..., L_d^{|L_{d_1}|}\}$ for d such that each list has at most $|L_{d_2}|$ elements. The construction process for $\{L_d^1, L_d^2, ..., L_d^{|L_{d_1}|}\}$ is described as follows. Let (w^1, w^2, h^1, h^2) be the ith element in L_{d_1}. The α-procedure constructs a list L_d^i such that its elements are obtained by sequentially combining (w^1, w^2, h^1, h^2) and each element $(w^{1'}, w^{2'}, h^{1'}, h^{2'})$ in L_{d_2}, scanned from the beginning to the end, by using the formula: $(w^1 + w^{1'}, w^2, max(h^1, h^{2'}), h^{2'})$. (See Figure 7.) Note that each L_d^i satisfies properties $P3 - P6$ and hence is an L-list. After all the L-lists associated with d are obtained, α-procedure proceeds to prune any redundant element within each of these L-lists by scanning each list from the beginning to the end once. As a result, each L_d^i is an irreducible L-list. Clearly, α-procedure has time complexity of $O(|L_{d_1}||L_{d_2}|)$, and the total number of all non-redundant implementations for d, stored in $\{L_d^1, L_d^2, ..., L_d^{|L_{d_1}|}\}$, is also $O(|L_{d_1}||L_{d_2}|)$.

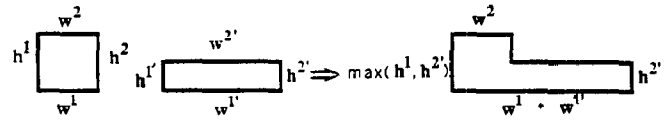

Figure 7

4.2.2 β-procedure

Suppose d is a β-node. According to Definition 6, its children d_1 and d_2 represent an L-shaped block and a rectangular block, respectively. Let $\{L_{d_1}^1, L_{d_1}^2, ..., L_{d_1}^p\}$ be the irreducible L-lists constructed for d_1 and L_{d_2} be the irreducible R-list constructed for d_2. The β-procedure constructs p lists $\{L_d^1, L_d^2, ..., L_d^p\}$ for d and each list L_d^i has at most $(|L_{d_1}^i| + |L_{d_2}| - 1)$ elements. Each L_d^i is obtained by a procedure similar to merging the two lists $L_{d_1}^i$ and L_{d_2} by scanning the two lists from the beginning to the end. Let

$$L_{d_1}^i = \{(w_1^1, w_1^2, h_1^1, h_1^2), ..., (w_k^1, w_k^2, h_k^1, h_k^2)\}$$
$$L_{d_2} = \{(w_1^{1'}, w_1^{2'}, h_1^{1'}, h_1^{2'}), ..., (w_n^{1'}, w_n^{2'}, h_n^{1'}, h_n^{2'})\}$$

where $k = |L_{d_1}^i|$ and $n = |L_{d_2}|$. The formula for obtaining an element in L_d^i is $(max(w_x^1, w_y^{2'}), w_x^2, h_x^1 + h_y^{1'}, h_x^2 + h_y^{2'})$ when considering $(w_x^1, w_x^2, h_x^1, h_x^2) \in L_{d_1}^i$ and $(w_y^{1'}, w_y^{2'}, h_y^{1'}, h_y^{2'}) \in L_{d_2}$, where $1 \le x \le k$ and $1 \le y \le n$. A key observation is that we do not have to consider all kn such new elements since many of them clearly dominate some others. For example, in Figure 8 with $w_x^1 \ge w_y^{2'}$, there is no reason

to consider $(w_x^1, w_x^2, h_x^1, h_x^2)$ and $(w_z^{1'}, w_z^{2'}, h_z^{1'}, h_z^{2'})$ for any $z > y$, since, based on that L_{d_2} satisfies $P1$ and $P2$, we have

$$max(w_x^1, w_z^{2'}) = max(w_x^1, w_y^{2'}) = w_x^1,$$
$$h_x^1 + h_z^{1'} \geq h_x^1 + h_y^{1'},$$
$$h_x^2 + h_z^{2'} \geq h_x^2 + h_y^{2'},$$

and these inequalities imply that $(max(w_x^1, w_z^{2'}), w_x^2, h_x^1 + h_z^{1'}, h_x^2 + h_z^{2'})$ is a redundant implementation for d. Similarly, with $w_x^1 \leq w_y^{2'}$, it is also unnecessary to consider $(w_z^1, w_z^2, h_z^1, h_z^2)$ and $(w_y^{1'}, w_y^{2'}, h_y^{1'}, h_y^{2'})$ for any $z > x$, based on that $L_{d_1}^i$ satisfies properties $P3 - P6$. Thus for each list $L_{d_1}^i$,when combining its elements together with the elements in L_{d_2}, we can consider these operations as similar to the approach used in [5] to combine two blocks vertically. Note that each L_d^i satisfies properties $P3 - P6$ and hence is an L-list. After all the L-lists associated with d are obtained, β-procedure proceeds to prune redundant elements within each of these L-lists by scanning each list from the beginning to the end once. As a result, each L_d^i is an irreducible L-list. The time complexity of the β-procedure and the total number of non-redundant implementations for d, stored in $\{L_d^1, L_d^2, ..., L_d^p\}$, are both $O((\sum_{i=1}^{p} |L_{d_1}^i|)+p|L_{d_2}|)$.

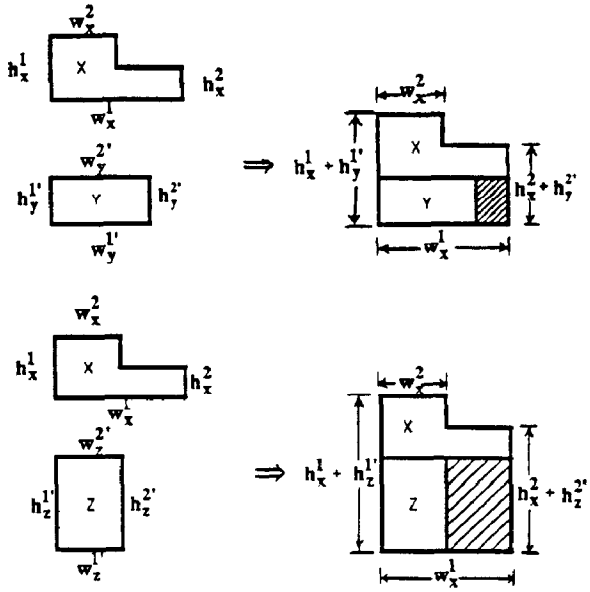

Figure 8

4.2.3 γ-procedure

Suppose d is a γ-node. According to Definition 7, its children d_1 and d_2 represent an L-shaped block and a rectangular block, respectively. Let $\{L_{d_1}^1, L_{d_1}^2, ..., L_{d_1}^p\}$ be the irreducible L-lists constructed for d_1 and L_{d_2} be the irreducible R-list constructed for d_2. The γ-procedure constructs q lists $\{L_d^1, L_d^2, ..., L_d^q\}$ for d, where $p \leq q \leq p|L_{d_2}|$. For each list $L_{d_1}^i$, γ-procedure constructs at most $|L_{d_2}|$ lists for d by combining the elements, scanned from the end to the beginning, between $L_{d_1}^i$ and L_{d_2}. Let

$$L_{d_1}^i = \{(w_1^1, w_1^2, h_1^1, h_1^2), ..., (w_k^1, w_k^2, h_k^1, h_k^2)\}$$
$$L_{d_2} = \{(w_1^{1'}, w_1^{2'}, h_1^{1'}, h_1^{2'}), ..., (w_n^{1'}, w_n^{2'}, h_n^{1'}, h_n^{2'})\}$$

where $k = |L_{d_1}^i|$ and $n = |L_{d_2}|$. The formula for obtaining an implementation for d is:

$$(w_x^1 + w_y^{1'}, w_x^2, max(h_x^1, h_y^{1'}), max(h_x^2, h_y^{1'}))$$

when considering $(w_x^1, w_x^2, h_x^1, h_x^2) \in L_{d_1}^i$ and $(w_y^{1'}, w_y^{2'}, h_y^{1'}, h_y^{2'}) \in L_{d_2}$, where $1 \leq x \leq k$ and $1 \leq y \leq n$. In order to prune redundant elements, we have to consider three different cases.

- Case (1): $h_x^2 > h_y^{1'}$
  In this case, there is no reason to consider $(w_x^1, w_x^2, h_x^1, h_x^2)$ and $(w_z^{1'}, w_z^{2'}, h_z^{1'}, h_z^{2'})$ for any $z < y$, since, based on that L_{d_2} satisfies properties $P1$ and $P2$, we have

  $$w_x^1 + w_z^{1'} \geq w_x^1 + w_y^{1'},$$
  $$max(h_x^1, h_z^{1'}) = max(h_x^1, h_y^{1'}) = h_x^1,$$
  $$max(h_x^2, h_z^{1'}) = max(h_x^2, h_y^{1'}) = h_x^2,$$

  and these inequalities imply that $(w_x^1 + w_z^{1'}, w_x^2, max(h_x^1, h_z^{1'}), max(h_x^2, h_z^{1'}))$ is a redundant implementation for d. (See Figure 9.)

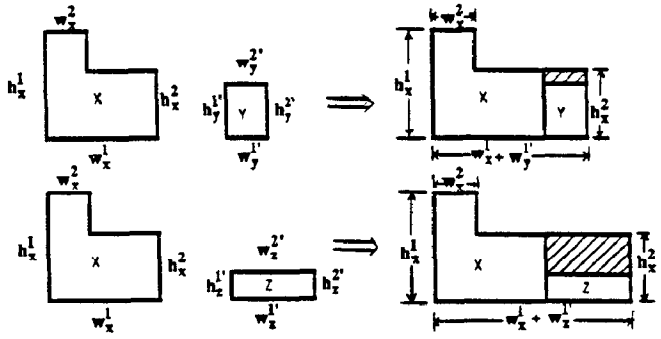

Figure 9

- Case (2): $h_x^1 \leq h_y^{1'}$
  In this case, there is no reason to consider $(w_z^1, w_z^2, h_z^1, h_z^2)$ and $(w_y^{1'}, w_y^{2'}, h_y^{1'}, h_y^{2'})$ for any $z < x$, since, based on that $L_{d_1}^i$ satisfies properties $P3 - P6$, we have

  $$w_z^1 + w_y^{1'} \geq w_x^1 + w_y^{1'},$$
  $$max(h_z^1, h_y^{1'}) = max(h_x^1, h_y^{1'}) = h_y^{1'},$$
  $$max(h_z^2, h_y^{1'}) = max(h_x^2, h_y^{1'}) = h_y^{1'},$$

  and these inequalities imply that $(w_z^1 + w_y^{1'}, w_z^2, max(h_z^1, h_y^{1'}), max(h_z^2, h_y^{1'}))$ is a redundant implementation for d. (See Figure 10.)

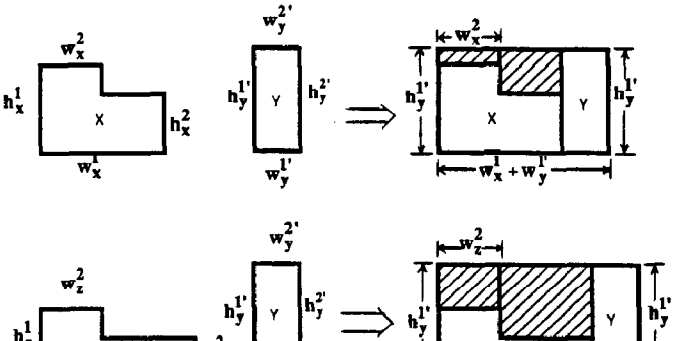

Figure 10

- Case (3): $h_x^2 \leq h_y^{1'} < h_x^1$
  In this case, we have to create a new list to store the combinations between $(w_z^1, w_z^2, h_z^1, h_z^2)$ and $(w_y^{1'}, w_y^{2'}, h_y^{1'}, h_y^{2'})$ for any $z < x$ until $z = 1$ or $h_z^1 \leq h_y^{1'}$.

Thus for each list $L_{d_1}^i$,when combining its elements together with the elements of L_{d_2}, in Cases (1) and (2) we can

consider these operations as similar to the approach used in [5] to combine two blocks horizontally. But in Case (3), it seems that there are less pruning. Note that these lists $\{L_d^1, L_d^2, ..., L_d^q\}$ constructed by the above approach satisfy properties $P3 - P6$ and hence are L-lists. After all the L-lists associated with d are obtained, γ-procedure proceeds to prune redundant elements within each of these L-lists by scanning each list from the beginning to the end once. As a result, each L_d^i, $1 \le i \le q$, is an irreducible L-list. In the worst case, the time complexity of γ-procedure and the total number of non-redundant implementations for d , stored in $\{L_d^1, L_d^2, ..., L_d^q\}$, are both $O((\sum_{i=1}^{p} |L_{d_1}^i|)|L_{d_2}|)$.

4.2.4 δ-procedure

Suppose d is a δ-node. According to Definition 8, its children d_1 and d_2 represent an L-shaped block and a rectangular block, respectively. Let $\{L_{d_1}^1, L_{d_1}^2, ..., L_{d_1}^p\}$ be the irreducible L-lists constructed for d_1 and L_{d_2} be the irreducible R-list constructed for d_2. The δ-procedure first constructs p lists $\{L_d^1, L_d^2, ..., L_d^p\}$ for d and each list L_d^i has at most $(|L_{d_1}^i| + |L_{d_2}| - 1)$ elements. Then it proceeds to merge the p lists into an irreducible R-list since d represents a rectangular block. Each list L_d^i is obtained by a procedure similar to merging the two lists $L_{d_1}^i$ and L_{d_2} by scanning the two lists from the beginning to the end. Let

$$L_{d_1}^i = \{(w_1^1, w_1^2, h_1^1, h_1^2), ..., (w_k^1, w_k^2, h_k^1, h_k^2)\}$$
$$L_{d_2} = \{(w_1^{1'}, w_1^{2'}, h_1^{1'}, h_1^{2'}), ..., (w_n^{1'}, w_n^{2'}, h_n^{1'}, h_n^{2'})\}$$

where $k = |L_{d_1}^i|$ and $n = |L_{d_2}|$. The formula for obtaining an element in L_d^i is:

$$(max(w_x^1, w_x^2 + w_y^{2'}), max(w_x^1, w_x^2 + w_y^{2'}),$$
$$max(h_x^1, h_x^2 + h_y^{2'}), max(h_x^1, h_x^2 + h_y^{2'}))$$

when considering $(w_x^1, w_x^2, h_x^1, h_x^2) \in L_{d_1}^i$ and $(w_y^{1'}, w_y^{2'}, h_y^{1'}, h_y^{2'}) \in L_{d_2}$, where $1 \le x \le k$ and $1 \le y \le n$. Similarly, we do not have to consider all kn such new elements since some of them clearly dominate some others. For example, in Figure 11, with $(w_x^1 - w_x^2) \ge w_y^{2'}$, there is no reason to consider $(w_x^1, w_x^2, h_x^1, h_x^2)$ and $(w_z^{1'}, w_z^{2'}, h_z^{1'}, h_z^{2'})$ for any $z > y$, since, based on that L_{d_2} satisfies properties $P1$ and $P2$, we have

$$max(w_x^1, w_x^2 + w_z^{2'}) = max(w_x^1, w_x^2 + w_y^{2'}) = w_x^1,$$
$$max(h_x^1, h_x^2 + h_z^{2'}) \ge max(h_x^1, h_x^2 + h_y^{2'}),$$

and these inequalities imply that $(max(w_x^1, w_x^2 + w_z^{2'}), max(w_x^1, w_x^2 + w_z^{2'}), max(h_x^1, h_x^2 + h_z^{2'}), max(h_x^1, h_x^2 + h_z^{2'}))$ is a redundant implementation for d. On the other hand, with $(w_x^1 - w_x^2) \le w_y^{2'}$, it is also unnecessary to consider $(w_z^1, w_z^2, h_z^1, h_z^2)$ and $(w_y^{1'}, w_y^{2'}, h_y^{1'}, h_y^{2'})$ for any $z > x$, based on that $L_{d_1}^i$ satisfies properties $P3 - P6$. Thus for each list $L_{d_1}^i$,when combining its elements together with the elements of L_{d_2}, we can consider these operations as similar to the approach used in [5] to combine two blocks vertically. Note each L_d^i satisfies properties $P1$ and $P2$ and hence is an R-list. δ-procedure is quite similar to β-procedure except the following.

- The formula for obtaining an implementation for d needs to be modified.
- The conditions to combine two elements needs to be modified.
- After obtaining the p lists constructed for d, δ-procedure proceeds to merge these p lists into an R-list and then prune redundant implementations after the merging process.

As a result, the list constructed for d is an irreducible R-list. The time complexity of obtaining the p lists is the same as that of β-procedure, i.e., $O((\sum_{i=1}^{p} |L_{d_1}^i|) + p|L_{d_2}|)$. The total number of non-redundant implementations for d is also $O((\sum_{i=1}^{p} |L_{d_1}^i|) + p|L_{d_2}|)$. Since each of the p lists is an R-list, we can easily use the approach similar to mergesort [1] to merge the p lists into one. Therefore the time complexity of the merging process takes $O(\log p((\sum_{i=1}^{p} |L_{d_1}^i|)+p|L_{d_2}|))$.

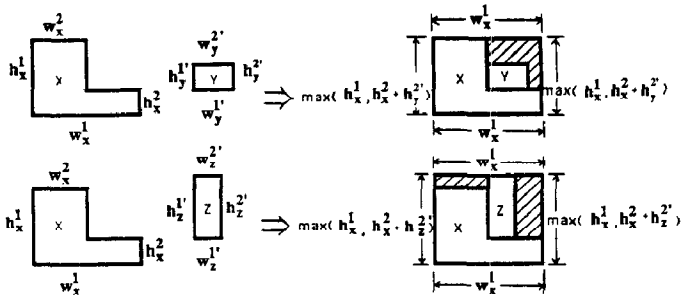

Figure 11

5 Improvement of the Algorithm

In this section, we would like to point out a factor which may affect the efficiency of our algorithm. Again, consider the wheel, say W, shown in figure 6. W is composed of 5 rectangular blocks and let rectangular block B_i have k_i non-redundant implementations. Based on our algorithm, the sequence of forming W will be : (1) combining the rectangular block B_1 and B_2 to get an L-shaped block Q_1; (2) combining the L-shaped block Q_1 and the rectangular block B_3 to get an L-shaped block Q_2; (3) combining the L-shaped block Q_2 and the rectangular block B_4 to get an L-shaped block Q_3; (4) combining the L-shaped block Q_3 and the rectangular block B_5 to get W (i.e., the original wheel). In order to obtain all non-redundant implementations of W, our algorithm sequentially applies α-procedure, β-procedure, γ-procedure and δ-procedure corresponding to the above 4 steps. Therefore the total number of the non-redundant implementations, constructed by our algorithm is $S = O(k_1 k_4 (k_2 + k_3 + k_5))$, and the corresponding time complexity is $T = O(S \log S)$. Note that the time complexity T is an increasing function of S, thus if S can be reduced, T can be reduced too. We will focus on how to improve the efficiency of our algorithm by considering how to reduce S.

In Figure 12 we show three wheels (named W_1, W_2 and W_3) which have the same form as W and are obtained by counterclockwise rotating W by 90°, 180° and 270°, respectively. The corresponding binary trees representing the three wheels are shown in Figure 12. By using our algorithm based on these binary trees, the individual total number of all non-redundant implementations corresponding to W_1, W_2 and W_3 is listed as follows.

$$S_1 = O(k_5 k_3 (k_2 + k_1 + k_4))$$
$$S_2 = O(k_4 k_1 (k_2 + k_5 + k_3))$$
$$S_3 = O(k_3 k_5 (k_2 + k_4 + k_1))$$

Clearly, in the worst case, $S = S_2$ and $S_1 = S_3$. If $k_5 k_3 (k_2 + k_1 + k_4) \ll k_1 k_4 (k_2 + k_3 + k_5)$, we expect $S_1 < S$. In the case, we recommend first rotating W to W_1 and obtain the list of all non-redundant implementations for W_1, and rotate W_1 back to W later. On the other hand, if $k_5 k_3 (k_2 + k_1 + k_4)$ is not much less than $k_1 k_4 (k_2 + k_3 + k_5)$, it is hard to tell whether S_1 is less than S. Also there is a cost associated with rearranging the R-lists for the blocks after rotation to satisfy properties $P1$ and $P2$, which makes using W_1 more costly. Therefore, we recommend to use W_1 instead of W

only if $k_5k_3(k_2+k_1+k_4) < \theta k_1k_4(k_2+k_3+k_5)$, where θ is a user input parameter with $0 < \theta < 1$.

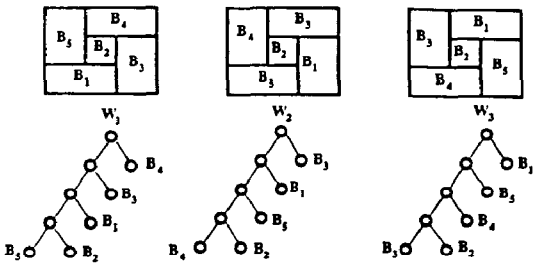

Figure 12

6 Extension to General Floorplans

Every floorplan can be viewed as obtained by recursively partitioning a rectangle into smaller rectangles. The hierarchy of the partitioning can be represented by a tree T. Let u be an internal node of T. In this case, u may have degree more than five and represents a more complicated non-slicing floorplan. As before, we need to construct an irreducible R-list to store all non-redundant implementations of the block represented by u. We can extend the idea for the case of wheels to this general case. There are two cases to consider.

1. If the irreducible R-list associated with u can be obtained by repeatedly using α-procedure, or β-procedure or γ-procedure, and finally using δ-procedure, then we can rearrange the subtree rooted at u into a binary tree and recursively obtain the irreducible L-lists and R-list for the nodes in the binary tree. (See Figure 13 for an example.)

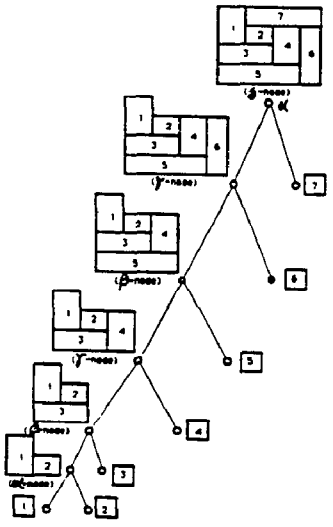

Figure 13

2. If the irreducible R-list associated with u can not be obtained from the previous case, then we have to decompose the rectangular block, which represented by u, into several L-shaped blocks such that the irreducible L-lists of each L-shaped block can be obtained by repeatedly using α-procedure, or β-procedure or γ-procedure. After the irreducible L-lists of each L-shaped block are obtained, based on the topology of the rectangular block represented by u, the lists of each L-shaped block can be combined together by trying all combinations. Finally, rearrange the list of u such that the resulting R-list remains irreducible.

7 Experimental Results and Concluding Remarks

We have implemented our algorithm in C language on a Sun 3/50 workstation running Unix 4.2BSD. The experimental results are very encouraging. The 7 test examples we used are described as follows.

- **EX1:** The 25-module floorplan shown in Figure 14 with each module having 3 possible implementations $\{4 \times 1, 2 \times 2, 1 \times 4\}$;
- **EX2:** The 25-module floorplan shown in Figure 14 with each module having 4 possible implementations $\{6 \times 1, 3 \times 2, 2 \times 3, 1 \times 6\}$;
- **EX3:** The 25-module floorplan shown in Figure 14 with each module having 5 possible implementations $\{16 \times 1, 8 \times 2, 4 \times 4, 2 \times 8, 1 \times 16\}$;
- **EX4:** The 25-module floorplan shown in Figure 14 with each module having 6 possible implementations $\{12 \times 1, 6 \times 2, 4 \times 3, 3 \times 4, 2 \times 6, 1 \times 12\}$;
- **EX5:** The 25-module floorplan shown in Figure 14 with each module having 8 possible implementations $\{24 \times 1, 12 \times 2, 8 \times 3, 6 \times 4, 4 \times 6, 3 \times 8, 2 \times 12, 1 \times 24\}$;
- **EX6:** The 24-module floorplan in [6] (See Figure 15.);
- **EX7:** The 120-module floorplan shown in Figure 16 which is a wheel with each rectangular block being a copy of EX6;

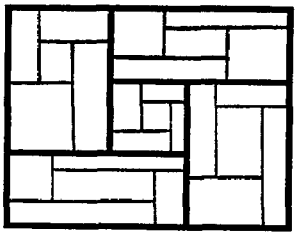

Figure 14

First, we compared our algorithm with the branch-and-bound optimal algorithm in [6] on EX1 to EX5. In fact, the branch-and-bound program we used is an implementation of [6] by Arvindam et al [2]. Because the current sequential version of the branch-and-bound program in [2] does not incorporate the slicing floorplan algorithm [5], in order to make a fair comparison, we choose the floorplan shown in Figure 14, which can not be partially processed by slicing floorplan algorithm, to be our test examples EX1 to EX5. The experimental results of EX1 to EX5 between our program and the branch-and-bound program in [2] are listed in Table 1. In Table 1, BB denotes the branch-and-bound algorithm and OPT denotes our algorithm. The running time of our algorithm on each test example among EX1 to EX5 is substantially less than that of the branch-and-bound algorithm [6] implemented by [2]. Since the branch-and-bound algorithm uses an enumeration tree to represent the states of the search space, we also made a comparison on the number of visited nodes. Note that in our algorithm, the number of the visited nodes is equal to the sum of the number of the implementations ever generated for each internal node in the binary tree representing the given floorplan. Again the number of nodes visited by our algorithm is substantially less than that by [2]. For test examples EX4 and EX5,since

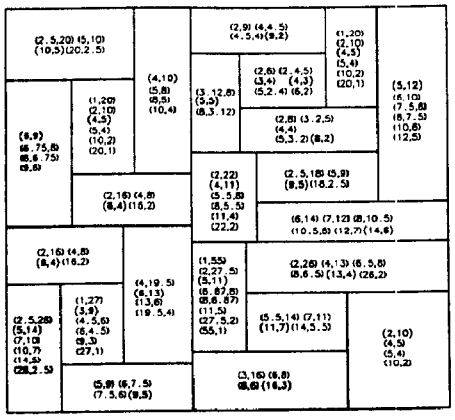

Figure 15

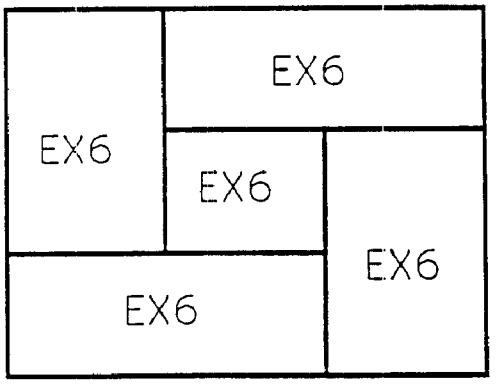

Figure 16

Test Example	Optimal Area	Total # of Implementations	# of Visited Nodes		Running Time (sec)	
			BB	OPT	BB	OPT
EX1	121	3^{25}	1620	348	59	0.3
EX2	176	4^{25}	37706	653	1506	0.7
EX3	484	5^{25}	321985	1921	10903	1.8
EX4	352	6^{25}	-	2202	> 2 days	2.0
EX5	660	8^{25}	-	6256	> 2 days	4.3
EX6	1024	2.03×10^{16}	$\approx 10^4$	966	-	0.9
EX7	6970	3.45×10^{81}	-	17242	-	14.8

Table 1: Experimental Results of EX1-EX7

the branch-and-bound program in [2] ran for 2 days and did not terminate, we did not get the data about the number of visited nodes and hence put a '-' in the corresponding entry of Table 1 to indicate that the data is unavailable.

Then we compared the experimental results of EX6 with that of [6]. Because Wimer et al [6] only presented their experimental results of EX6 in terms of the ratio of the number of visited nodes to the total number of leaves in the search tree, the running time of EX6 in [6] is unavailable and hence '-' is put in the corresponding entry of Table 1. Again our algorithm checked less number of nodes than that of [6]. Finally, in order to show that our program can handle large floorplans, we ran our algorithm on EX7. The floorplan of EX7 is primarily composed of 5 rectangular blocks and each rectangular block is a copy of the floorplan of EX6. As a result, EX7 is a 120-module floorplan. The total number of possible implementations of EX7 is about 3.45×10^{81} and our algorithm only ran for 14.8 seconds to obtain the optimal solution. The experimental results of EX7 are listed in Table 1. Note that the optimal areas for the 7 test examples are also listed in Table 1.

The above experimental results indicate that our algorithm, based on an extension of the technique in [5], did prune very large number of redundant implementations. In addition, since our algorithm basically exploits the geometric property of the topology of the given floorplan, it does not need to depend on the polar dual graphs [6] to calculate the longest paths. Consequently, our algorithm is able to run more efficiently than the branch-and-bound algorithm in [6].

References

[1] Aho, A.V., J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974.

[2] Arvindam, S., V. Kumar and V.N. Rao "Floorplan Optimization on Multiprocessors," *ICCD-89.*

[3] Lauther, U., "A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation," *Journal of Digital Systems, Vol. IV, Issue 1 (1980), 21-34.*

[4] Otten, R.H.J.M., "Automatic Floorplan Design," *Proc. 19th ACM/IEEE Design Automation Conf. (1982), 261-267.*

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

[6] Wimer, S., I. Koren and I. Cederbaum, "Optimal Aspect Ratios of Building Blocks in VLSI," *IEEE Trans. on CAD, Vol. CAD-8, No. 2 (1989), 139-145.*

[7] Wong, D.F. and C.L. Liu, "A New Algorithm for Floorplan Design," *Proc. 23rd ACM/IEEE Design Automation Conf. (1986), 101-107.*

[8] Wong, D.F. and P. Sakhamuri, "Efficient Floorplan Area Optimization," *Proc. 26th ACM/IEEE Design Automation Conf. (1989), 586-589.*

[9] Wong, D.F. and K.S. The, "An Algorithm for Hierarchical Floorplan Design," *Proc. IEEE International Conf. on Computer-Aided Design (1989), 434-487.*