Optimal Floorplan Area Optimization

Ting-Chi Wang, Student Member, IEEE, and D. F. Wong, Member, IEEE

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 [7]. The running time of our algorithm is substantially less than that of [7]. For several examples where the algorithm in [7] ran for days and did not terminate, our algorithm produced optimal solutions in a few seconds.

I. INTRODUCTION

FLOORPLAN design [4], [8] is an important step in the physical design of VLSI circuits. A typical approach to floorplan design is first to 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 this optimization problem in polynomial time [5]. However, for general nonslicing floorplans this problem has been proved to be NP-complete [5]. Consequently, recent research efforts have focused on the efficient heuristic method [9] and the exponential time branch-and-bound algorithm [7].

In this paper we extend the technique used in [5] to obtain an optimal algorithm for a special class of nonslicing floorplans, called hierarchical floorplans of order 5 [9], [10]. Our idea can be extended to the general case. Experimental results indicate that our optimal algorithm is efficient and capable of successfully handling very large floorplans. We compare our algorithm with the branch-and-bound optimal algorithm of Wimer et al. [7]. The running time of our algorithm is substantially less than that of [7]. For several examples where the algorithm in [7] ran for days and did not terminate, our algorithm produced optimal solutions in a few seconds.

The rest of this paper is organized as follows. In Section II, we give a formal description of the floorplan area optimization problem. In Section III, some necessary terms and definitions are introduced. In Section IV, we present our algorithm. In Section V, the issue of further improving our algorithm is addressed. In Section VI, we describe the idea of how to extend our algorithm to handle general floorplans. In Section VII, experimental results are presented. Finally, we conclude this paper in Section VIII with some remarks.

II. PROBLEM DESCRIPTION

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

There are two kinds of floorplans: slicing and nonslicing. A slicing floorplan is a floorplan which can be obtained by recursively cutting a rectangle into two parts by either a vertical or a horizontal line [4], [8]. A nonslicing floorplan is a floorplan which is not a slicing floorplan. A wheel is a nonslicing floorplan of five modules. There are only two possible wheels and they are the simplest nonslicing floorplans (see Fig. 1.)

A natural extension of slicing floorplans is the set of hierarchical floorplans of order 5 [9], [10]. 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 or a horizontal line or into five parts by a wheel. (Note that we do not consider partitioning a rectangle into a slicing floorplan with three, four, or five parts because these partitions can be obtained by recursively partitioning the rectangle into two parts.) The hierarchy of the partitioning can be represented by a floorplan tree. Fig. 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 the floorplan tree has a degree of either 2 or 5, corresponding to partitioning a rectangle into two or five parts, respectively.

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 $\{w_1 \times h_1, w_2 \times h_2, \ldots, w_n \times h_n\}$, 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.

III. PRELIMINARIES

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

Since each module has a finite number of possible implementations, 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 Fig. 5). Clearly, a rectangular block can be considered as a special type of 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 Fig. 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 four inequalities hold:

1. $w^1 \geq w'^1$;
2. $w^2 \geq w'^2$;
3. $h^1 \geq h'^1$; 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 $\{(w^1_1, w^2_1, h^1_1, h^2_1), \ldots, (w^1_n, w^2_n, h^1_n, h^2_n)\}$ is an R-list if the following properties, $P1$ and $P2$, are satisfied for all $j$ and $k$, $1 \leq j < k \leq n$:

1. $w^j_1 = w^j_2 \geq w^j_k = w^j_1$
2. $h^j_1 = h^j_2 \leq h^j_k = h^j_2$.

By Definition 2, an R-list represents a list of implementations of a rectangular block in which the implementations are arranged in order of decreasing width and increasing height.

**Definition 3:** A list $\{(w^1_1, w^2_1, h^1_1, h^2_1), \ldots, (w^1_n, w^2_n, h^1_n, h^2_n)\}$ is an L-list if the following properties, $P3$, $P4$, $P5$, and $P6$, are satisfied for all $j$ and $k$, $1 \leq j < k \leq n$:

1. $w^j_1 \geq w^j_1$;
2. $w^j_1 = w^j_2$; $P5$: $h^j_1 \leq h^j_k$;
3. $P6$: $h^j_1 \leq h^j_2$.

By Definition 3, an L-list represents a list of implementations of an L-shaped block in which the implementations are all of equal top edge width and are arranged in order of decreasing bottom edge width and increasing left edge and right edge heights.

**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 nonredundant implementations of a rectangular block are stored in an irreducible R-list, and
all nonredundant implementations of an L-shaped block are stored in a set of irreducible L-lists.

IV. An Optimal 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 nonredundant implementations of the corresponding basic rectangle. (Note that nonredundant implementations of a basic rectangle are the same as the nonredundant 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 irreducible R-list corresponds to the set of all nonredundant implementations of the block represented by the internal node. After the irreducible R-list associated with the root is obtained, since it corresponds to all nonredundant implementations of the floorplan, we can scan it once to search for the implementation with minimum area which definitely is the optimal implementation of the floorplan. Therefore our algorithm is an optimal algorithm.

We now describe how to obtain the irreducible R-list for each internal node of $T$. Let $v$ be an internal node of $T$. We can obtain the irreducible R-list of $v$ from the irreducible R-list of its sons. According to the degree of $v$, our algorithm uses different approaches to obtain the irreducible R-list for $v$. There are only two cases to be considered: the degree of $v$ is 2 (Section IV-A) or the degree of $v$ is 5 (Section IV-B).

A. Slicing

Suppose the degree of $v$ is 2. 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-list associated with $v_1$ and $v_2$, respectively. The basic idea in [5] is that it is unnecessary to consider all $k_1k_2$ possible implementations of $v$; instead, there are at most $k_1 + k_2 - 1$ implementations that are nonredundant and relevant to the optimal floorplan. As a result, to obtain the irreducible R-list for $v$, we first generate an R-list storing the candidates of nonredundant implementations of $v$ by scanning the two irreducible R-lists associated with $v_1$ and $v_2$ once, and then scan the resulting R-list once to prune any redundant element in it. The time to obtain the irreducible R-list for $v$ is $O(k_1 + k_2)$. The details of this algorithm can be found in [5].

B. Wheel

Suppose the degree of $v$ is 5. In this case, $v$ corresponds to a wheel and has five sons, $v_1$, $v_2$, $v_3$, $v_4$, and $v_5$. Although there are two possible wheels, we can always treat one as the reflection of the other. Hence without loss of generality, we only consider how to obtain nonredundant implementations of the wheel in the form shown in the left part of Fig. 1. As for the other wheel shown in Fig. 1, we can first reflect it appropriately and then apply the same algorithm.

Let $T_v$ be the subtree rooted at $v$. We can restructure $T_v$ into a binary tree, $T_v'$, as shown in Fig. 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.

Definition 5: An $\alpha$ 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 $\beta$ 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 $\gamma$ 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 $\delta$ node is a node which corresponds to a rectangular block obtained by combining an L-shaped block and a rectangular block.

By the above definitions, $u_1$, $u_2$, $u_3$, and $v$ are $\alpha$, $\beta$, $\gamma$, and $\delta$ nodes, respectively (see Fig. 6). Based on the irreducible R-lists of $v_1$, $v_2$, $v_3$, $v_4$, and $v_5$, we will construct a set of irreducible L-lists for each of the nodes $u_1$, $u_2$, and $u_3$ and an irreducible R-list for $v$. All nonredundant implementation of the blocks represented by the nodes are stored in these irreducible lists. To obtain the irreducible R-list for $v$, we perform 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 will 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 in [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, called the $\alpha$ procedure, the $\beta$ procedure, the $\gamma$ procedure, and the $\delta$ procedure. The $\alpha$ procedure, the $\beta$ procedure and the $\gamma$ procedure are the procedures for constructing a set of irreducible L-lists for an $\alpha$ node, a $\beta$ node, and a $\gamma$ node, respectively. The $\delta$ procedure is the procedure for constructing an irreducible R-list for a $\delta$ node. We will use the $\alpha$ procedure for step 1, the $\beta$ procedure for step 2, the $\gamma$ procedure for step 3, and the $\delta$ procedure for step 4.
The α Procedure

**input:** two irreducible R-lists, $L_{d_1}$ and $L_{d_2}$ associated with $d_1$ and $d_2$, respectively.

**output:** a set of irreducible L-lists \( \{L_{d_1}, L_{d_2}, \ldots, L_{d_{|L_{d_1}|}}\} \) associated with $d$. 

**begin**

for $i := 1$ to $|L_{d_1}|$ do 

\((w^{i}, w^{2i}, h^{i}, h^{2i}) := \text{the } i\text{th element of } L_{d_1}\)

Create an empty list $L_{d_1}^i$.

for $j := 1$ to $|L_{d_2}|$ do 

\((w^{1j}, w^{2j}, h^{1j}, h^{2j}) := \text{the } j\text{th element of } L_{d_2}\)

Insert the implementation \( (w^{i} + w^{1j}, w^{2i}, \max(h^{i}, h^{1j}), h^{2i}) \) to the end of $L_{d_1}^i$.

end for

end for

**end**

Clearly, the α procedure considers all possible combinations between $L_{d_1}$ and $L_{d_2}$, and hence has a time complexity of $O(|L_{d_1}||L_{d_2}|)$. The total number of implementations of $d$, stored in \( \{L_{d_1}, L_{d_2}, \ldots, L_{d_{|L_{d_1}|}}\} \), is $|L_{d_1}||L_{d_2}|$.

2) The β Procedure: Let $d$ be 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 \( \{L_{d_1}, L_{d_2}, \ldots, L_{d_{|L_{d_1}|}}\} \) be the set of 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 \( \{L_{d_1}^1, L_{d_2}^1, \ldots, L_{d_p}^1\} \) for $d$ and each list, $L_{d_j}$, has at most $|L_{d_1}| + |L_{d_2}| - 1$ elements. Each $L_{d_j}$ is obtained by a procedure similar to merging the two lists $L_{d_1}$ and $L_{d_2}$ by scanning the two lists from the beginning to the end. Let

\[
L_{d_1} = \{(w^1, w^{21}, h^1, h^{21}), \ldots, (w_k, w^{2k}, h_k, h^{2k})\}
\]

\[
L_{d_2} = \{(w_1^1, w_2^1, h_1^1, h_2^1), \ldots, (w_1^n, w_2^n, h_1^n, h_2^n)\}
\]

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

\[
(w^{i1}, w^{i2}, h^{i1}, h^{i2}) = \max(w^{i1}, w^{i2}, h^{i1}, h^{i2})
\]

when considering \( (w_1^1, w_2^1, h_1^1, h_2^1) \in L_{d_1} \) and \( (w_1^1, w_2^1, h_1^1, h_2^1) \in L_{d_2} \), where $1 \leq x \leq k$ and $1 \leq y \leq n$. A key observation is that we do not have to consider all $kn$ such new elements since many of them clearly dominate certain others. For example, in Fig. 8 with $w^1_1 \geq w_2^1$, there is no reason to consider \( (w^{i1}, w_2^1, h^{i1}, h_2^1) \) and \( (w_1^1, w^{i2}, h_1^1, h_2^2) \) for any $z > y$, since, from the fact that $L_{d_2}$ satisfies P1 and P2, we have

\[
\max(w^1_1, w_2^1) = w^1_1
\]

\[
h^{i1} + h_1^1 \geq h^{i1} + h_1^1
\]

\[
h^{i2} + h_2^2 \geq h^{i2} + h_2^2
\]

and these inequalities imply that \( \max(w^{i1}, w_2^1, h^{i1}, h_2^1, h_1^1, h^{i2}, h_2^2) \) is a redundant implementation for $d$. Simi-
Figure 8. Illustration of the β procedure.

literally, with \( w_1 \leq w_0 \), it is also unnecessary to consider \((w'_1, w'_2, h'_1, h'_2)\) and \((w'_2, w'_3, h'_3, h'_4)\) for any \( z > x \), based on the fact that \( L_{d_0} \) satisfies properties P3–P6. Thus for each \( L_{d_0} \), when combining its elements with the elements in \( L_{d_0} \), we can regard these operations as similar to the approach used in [5] to combine two blocks vertically. Note that each \( L_d \) satisfies properties P3–P6 and hence is an L-list. After each \( L_d \) associated with \( d \) is obtained, the β procedure proceeds to prune redundant elements within each \( L_d \). This is accomplished by scanning each \( L_d \) from the beginning to the end once by the procedure Prune_Redundant_Elements, shown below. Procedure Prune_Redundant_Elements is applicable to any L-list and clearly or R-list runs in linear time.

Prune_Redundant_Elements

**input:** \( L = \{L(1), L(2), \ldots, L(k)\} \), where \( L \) is an L-list or an R-list.

**output:** An irreducible L-list or R-list obtained by removing all redundant elements from \( L \).

begin

\( i = 1; \) (* \( i \) is the index to a candidate of redundant elements. *)

for \( j = 2 \) to \( k \) do

if \((L(i)\) and \(L(j)\) do not dominate each other\) then

(* \( L(i) \) is nonredundant. *)

\( i = j; \)

else if \((L(i)\) dominates \(L(j)\) then (* \( L(i) \) is redundant. *)

Remove \( L(i) \) from \( L; \)

else (* \( L(j) \) is redundant. *)

Remove \( L(j) \) from \( L; \)

end for

end

As a result, each \( L_d \) is an irreducible L-list. Note that for these irreducible L-list constructed by the β procedure, it is possible that for two elements in different lists one dominates the other, but we guarantee that all non-redundant implementations of \( d \) are completely stored in these irreducible L-lists. An outline of the β procedure is as follows.

The β Procedure

**input:** a set of irreducible L-lists \( \{L_{d_1}, L_{d_2}, \ldots, L_{d_p}\} \), associated with \( d_1 \), and an irreducible R-list, \( L_{d_0} \), associated with \( d_0 \).

**output:** a set of irreducible L-list, \( \{L_{d_1}, L_{d_2}, \ldots, L_{d_p}\} \), associated with \( d \).

begin

for \( i = 1 \) to \( p \) do

\( x := 1; \ y := 1; \)

while \((x \leq |L_{d_i}| \) and \(y \leq |L_{d_0}|\) do

\( (w'_1, w'_2, h'_1, h'_2) := \) the \( x \)th element of \( L_{d_i}; \)

\( (w'_1, w'_2, h'_3, h'_4) := \) the \( y \)th element of \( L_{d_0}; \)

Insert the implementation \((\max \{w'_1, w'_2\}, w'_3, h'_3 + h'_4)\) to the end of \( L_{d_0}; \)

if \( (w'_1 \geq w'_2) \) then \( x := x + 1; \)

if \( (w'_1 \leq w'_2) \) then \( y := y + 1; \)

end while

Apply Prune_Redundant_Elements to prune redundant elements within \( L_{d_i}; \)

end for

end

At the \( i \)th iteration of the “for” loop, the “while” loop is executed at most \(|L_{d_i}| + |L_{d_0}| - 1 \) times; hence the number of elements of \( L_{d_i} \) is also at most \(|L_{d_i}| + |L_{d_0}| - 1 \). The time to prune redundant elements within \( L_d \) is bounded by \( O(|L_{d_i}| + |L_{d_0}|) \). Thus the \( i \)th iteration of the “for” loop takes time \( O(|L_{d_i}| + |L_{d_0}|) \). Since the “for” loop iterates \( p \) times, the total time complexity of the β procedure is \( O(\Sigma_{i=1}^{p} (|L_{d_i}| + |L_{d_0}|)) \), and the total number of implementations of \( d \) stored in \( \{L_{d_1}, L_{d_2}, \ldots, L_{d_p}\} \), is bounded by \( \Sigma_{i=1}^{p} (|L_{d_i}| + |L_{d_0}|) - 1 \).

3) The γ Procedure: Let \( d \) be 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 \( \{L_{d_1}, L_{d_2}, \ldots, L_{d_p}\} \) be the set of irreducible L-lists constructed for \( d_1 \) and \( L_{d_0} \) be the irreducible R-list constructed for \( d_2 \). The γ procedure constructs \( q \) lists \( \{L_{d_1}, L_{d_2}, \ldots, L_{d_q}\} \) for \( d \), where \( p \leq q \leq p(|L_{d_0}| + 1) \). For each list \( L_{d_i} \), the γ procedure constructs at most \(|L_{d_i}| + 1 \) lists for \( d \) by combining the elements, scanned from the end to the beginning, between \( |L_{d_0}| \) and \( |L_{d_0}| \). Let

\[ L_{d_i} = \{(w'_1, w'_2, h'_1, h'_2), \ldots, (w'_1, w'_2, h'_2, h'_3)\} \]

\[ L_{d_k} = \{(w'_1, w'_2, h'_1, h'_2), \ldots, (w'_1, w'_2, h'_3, h'_4)\} \]

where \( k = |L_{d_i}| \) and \( n = |L_{d_0}| \). The formula for obtaining an implementation for \( d \) is

\[ (w'_1 + w'_2, \max \{h'_1, h'_2\}, \max \{h'_3, h'_4\}) \]

when considering \((w'_1, w'_2, h'_1, h'_2) \in L_{d_i} \) and \((w'_1, w'_2, h'_3, h'_4) \in L_{d_k} \).
$h^{1'}_i, h^{2'}_i \in L_{d2}$, 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:** $h^{1'}_i \geq h^{2'}_i$. In this case, there is no reason to consider $(w^{1'}_i, w^{2'}_i, h^{1'}_i, h^{2'}_i)$ and $(w^{1'}_i, w^{2'}_i, h^{1'}_i, h^{2'}_i)$ for any $z < y$, since, based on the fact that $L_{d2}$ satisfies properties P1 and P2, we have

$$w^{1'}_i + w^{2'}_i \geq w^{1'}_i + w^{2'}_i$$
$$\max(h^{1'}_i, h^{1'}_i) = \max(h^{1'}_i, h^{2'}_i) = h^{1'}_i$$
$$\max(h^{2'}_i, h^{2'}_i) = \max(h^{2'}_i, h^{2'}_i) = h^{2'}_i$$

and these inequalities imply that $(w^{1'}_i + w^{2'}_i, w^{1'}_i, max(h^{1'}_i, h^{2'}_i), max(h^{2'}_i, h^{2'}_i))$ is a redundant implementation for $d$ (see Fig. 9).

- **Case 2:** $h^{1'}_i \leq h^{2'}_i$. In this case, there is a reason to consider $(w^{1'}_i, w^{2'}_i, h^{1'}_i, h^{2'}_i)$ and $(w^{1'}_i, w^{2'}_i, h^{1'}_i, h^{2'}_i)$ for any $z < x$, since, based on the fact that $L_{d2}$ satisfies properties P3–P6, we have

$$w^{1'}_i + w^{2'}_i \geq w^{1'}_i + w^{2'}_i$$
$$w^{2} = w^{2}$$
$$\max(h^{1'}_i, h^{1'}_i) = \max(h^{1'}_i, h^{2'}_i) = h^{1'}_i$$
$$\max(h^{2'}_i, h^{2'}_i) = \max(h^{2'}_i, h^{2'}_i) = h^{2'}_i$$

and these inequalities imply that $(w^{1'}_i + w^{2'}_i, w^{2}, max(h^{1'}_i, h^{2'}_i), max(h^{2'}_i, h^{2'}_i))$ is a redundant implementation for $d$ (see Fig. 10).

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

Thus for each list $L_{d2}$, when combining its elements with the elements of $L_{d1}$, 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 is less pruning. Note that the lists $\{L^{1}_d, L^{2}_d, \ldots, L^{n}_d\}$ constructed by the approach satisfy properties P3–P6 and hence are L-lists. After each L-list associated with $d$ is obtained, the $\gamma$ procedure proceeds to prune redundant elements within each list by the procedure Prune Redundant Elements described in Section IV-B-2. As a result, each $L^{i}_d$, $1 \leq i \leq q$, is an irreducible L-list. Note that for these irreducible L-lists constructed by the $\gamma$ procedure, it is possible that for two elements in different lists one dominates the other, but we guarantee that all nonredundant implementations of $d$ are completely stored in these irreducible L-lists. An outline of the $\gamma$ procedure follows.

**The $\gamma$ Procedure**

**input:** a set of irreducible L-lists, $\{L^{1}_d, L^{2}_d, \ldots, L^{n}_d\}$, associated with $d_1$, and an irreducible R-list, $L_{d2}$, associated with $d_2$.

**output:** a set of irreducible L-lists, $\{L^{1}_d, L^{2}_d, \ldots, L^{n}_d\}$, associated with $d$, where $p \leq q \leq p(|L_{d2}| + 1)$.

begin

$q := p$;

for $i := 1$ to $p$ do

Create an empty list $L^{i}_d$;

$x := |L^{i}_d|; y := |L^{i}_d|;

while ($x \geq 1$ and $y \geq 1$) do

$(w^{1'}_i, w^{2}, h^{1'}_i, h^{2'}_i) :=$ the $x$th element of $L^{i}_d$;

$(w^{1'}_i, w^{2}, h^{1'}_i, h^{2'}_i) :=$ the $y$th element of $L^{i}_d$;

Insert the implementation $(w^{1'}_i + w^{2}, w^{2}, max(h^{1'}_i, h^{2'}_i), max(h^{2'}_i, h^{2'}_i))$ to the beginning of $L^{i}_d$;

if $(h^{1'}_i \geq h^{2'}_i)$ then $x := x - 1$; (*Case 1*)

else if $(h^{1'}_i \leq h^{2'}_i)$ then $y := y - 1$, (*Case 2*)

else if $(h^{2'}_i < h^{2'}_i < h^{1'}_i)$ and $x > 1$ then (*Case 3*)

begin

$q := q + 1; z := x$;

Create an empty list $L^{z}_d$;

repeat

$z := z - 1$;

$(w^{1'}_i, w^{2}, h^{1'}_i, h^{2'}_i) :=$ the $z$th element of $L^{i}_d$;

Insert the implementation $(w^{1'}_i + w^{2}, w^{2}, max(h^{1'}_i, h^{2'}_i), max(h^{2'}_i, h^{2'}_i))$ to the beginning of $L^{z}_d$;

until ($z = 1$ or $h^{1'}_i \leq h^{2'}_i$);

Apply Prune Redundant Elements to prune redundant elements within $L^{z}_d$;

$y := y - 1$;

end

end
end while

Apply Prune_Redundant_Elements to prune redundant elements within \( L_d \);

end for

end

At the \( i \)th iteration of the "for" loop, the "repeat-until" loop is executed at most \((|L_{d_j}| - 1) |L_{d_j}|)\) times; hence at most \(|L_{d_j}| + 1\) lists are constructed to store \(|L_{d_j}| \) implementations of \( d \). (The worst case happens when case 3 always remains true. Of these \(|L_{d_j}| + 1\) lists, the list \( L_d \) at the beginning of the \( i \)th iteration of the "for" loop stores \(|L_{d_j}|\) elements, and each of the remaining \(|L_{d_j}|\) lists, \( L_{d_j} \), where \( p + (i - 1) |L_{d_j}| + 1 \leq f \leq p + i |L_{d_j}| \), is created in case 3 and stores \(|L_{d_j}| - 1\) elements.)

The total time to prune redundant elements within each of the lists at the \( i \)th iteration of the "for" loop is also bounded by \( O(|L_{d_j}|) \). Thus the total time of the "for" loop takes time \( O(|L_{d_j}|) \). Since the "for" loop iterates \( p \) times, the total time complexity of \( \gamma \) procedure is \( O(E_p+1(|L_{d_j}|) |L_{d_j}|) \), and the total number of implementations of \( d \), stored in \( \{L_{d_1}, L_{d_2}, \ldots, L_{d_p}\} \), is bounded by \( E_p+1(|L_{d_j}|) |L_{d_j}| \).

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

\[
L_{d_i} = \{(w_i', w_i', h_i', h_i'), \ldots, (w_i', w_i', h_i', h_i')\}
\]

\[
L_{d_1} = \{(w_i^{1'}, w_i^{2'}, h_i^{1'}, h_i^{1'}), \ldots, (w_i^{1'}, w_i^{2'}, h_i^{1'}, h_i^{1'})\},
\]

where \( k = |L_{d_1}'| \) and \( n = |L_{d_2}'| \). The formula for obtaining an element in \( L_{d_1} \) is

\[
\begin{align*}
\max (w_i^{1'}, w_i^{2'}, + w_i^{3'}), & \max (w_i^{1'}, w_i^{2'}, + w_i^{3'}), \\
\max (h_i^{1'}, h_i^{1'}, + h_i^{1'}), & \max (h_i^{1'}, h_i^{1'}, + h_i^{1'})
\end{align*}
\]

when considering \((w_i^{1'}, w_i^{2'}, h_i^{1'}, h_i^{1'}) \in L_{d_1}'\) and \((w_i^{1'}, w_i^{2'}, h_i^{1'}, h_i^{1'}) \in L_{d_2}'\), where \( 1 \leq x \leq k \) and \( 1 \leq y \leq n \). Similarly, we do not have to consider all \( kn \) such new elements since some of them clearly dominate certain others. For example, in Fig. 11, with \((w_i^{1'} - w_i^{2'}) \leq w_i^{3'}\), there is no reason to consider \((w_i^{1'}, w_i^{2'}, h_i^{1'}, h_i^{1'})\) and \((w_i^{1'}, w_i^{2'}, h_i^{1'}, h_i^{1'})\) for any \( z > y \), since based on the fact that \( L_{d_2} \) satisfies properties \( P1 \) and \( P2 \), we have

\[
\begin{align*}
\max (w_i^{1'}, w_i^{2'}, + w_i^{3'}) = \max (w_i^{1'}, w_i^{2'}, + w_i^{3'}) = w_i^{3'} \\
\max (h_i^{1'}, h_i^{1'}, + h_i^{1'}) \geq \max (h_i^{1'}, h_i^{1'}, + h_i^{1'})
\end{align*}
\]

and these inequalities imply that \( \max (w_i^{1'}, w_i^{2'}, + w_i^{3'}) \), \( \max (w_i^{1'}, w_i^{2'}, + w_i^{3'}) \), \( \max (h_i^{1'}, h_i^{1'}, + h_i^{1'}) \), \( \max (h_i^{1'}, h_i^{1'}, + h_i^{1'}) \)

Fig. 11. Illustration of the \( \delta \) procedure.

The \( \delta \) Procedure

input: a set of irreducible L-lists, \( \{L_{d_1}', L_{d_2}', \ldots, L_{d_p}'\} \), associated with \( d_1 \), and an irreducible R-list, \( L_{d_2} \), associated with \( d_2 \).

output: an irreducible R-list, \( L_d \), associated with \( d \).

begin

for \( i := 1 \) to \( p \) do

Create an empty list \( L_{d_i}' \);

\( x := 1; y := 1; \)

while \((x \leq |L_{d_i}'| \) and \( y \leq |L_{d_i}'| \)) do

\( (w_i^{1'}, w_i^{2'}, h_i^{1'}, h_i^{1'}) := \) the \( x \)th element of \( L_{d_i}' \);

\( (w_i^{1'}, w_i^{2'}, h_i^{1'}, h_i^{1'}) := \) the \( y \)th element of \( L_{d_i}' \);

Insert the implementation

\( \max (w_i^{1'}, w_i^{2'}, + w_i^{3'}) \), \( \max (w_i^{1'}, w_i^{2'}, + w_i^{3'}) \), \( \max (h_i^{1'}, h_i^{1'}, + h_i^{1'}) \), \( \max (h_i^{1'}, h_i^{1'}, + h_i^{1'}) \)

\( \max (h_i^{1'}, h_i^{1'}, + h_i^{1'}) \) to the end of \( L_{d_i}' \);

if \((w_i^{1'} - w_i^{2'} \geq w_i^{3'}) \) then \( x := x + 1; \)

if \((w_i^{1'} - w_i^{2'} \leq w_i^{3'}) \) then \( y := y + 1; \)

end while
end for

Merge \( L_{d_1}', L_{d_2}', \ldots, L_{d_p}' \) into an R-list, \( L_{d_2} \);

Apply Prune_Redundant_Elements to prune redundant elements within \( L_{d_2} \);

end
By the same analysis as that in the β procedure, the
time taken in the "for" loop is \(O(\Sigma_{i=1}^{\delta} (|L_{d_i}^\delta| + |L_{d_i}|))\),
and the total number of implementations of \(d\) stored in
\(L_{d_1}^\delta, L_{d_2}^\delta, \ldots, L_{d_n}^\delta\) is bounded by \(\Sigma_{i=1}^{\delta} (|L_{d_i}^\delta| + |L_{d_i}| - 1)\).
The time to merge \(L_{d_1}^\delta, L_{d_2}^\delta, \ldots, L_{d_n}^\delta\) into \(L_d\) is \(O(\Sigma_{i=1}^{\delta} (|L_{d_i}^\delta| + |L_{d_i}|) \log p)\).
Since the number of elements stored in \(L_d\) is bounded by the total number of elements
stored in \(L_{d_1}^\delta, L_{d_2}^\delta, \ldots, L_{d_n}^\delta\), i.e., \(\Sigma_{i=1}^{\delta} (|L_{d_i}^\delta| + |L_{d_i}| - 1)\),
the time to prune redundant elements within \(L_d\) is
\(O(\Sigma_{i=1}^{\delta} (|L_{d_i}^\delta| + |L_{d_i}|))\). Therefore, the total time complexity of the δ procedure is
\(O(\Sigma_{i=1}^{\delta} (|L_{d_i}^\delta| + |L_{d_i}|) \log p)\),
and the number of all nonredundant implementations of \(d\), stored in \(L_d\), is bounded by \(\Sigma_{i=1}^{\delta} (|L_{d_i}^\delta| + |L_{d_i}| - 1)\).

5) Remarks: Suppose each rectangular block, \(B_i\), in the
wheel (see Fig. 6) has \(k_i\) nonredundant implementations.
In step 1, the inputs to the α procedure are two irreducible
R-lists of length \(k_1\) and \(k_2\) (for \(B_1\) and \(B_2\), respectively). It
follows from Section IV-B-1 that the α procedure constructs
\(k_1\) irreducible L-lists for \(u_1\), and each such irreducible L-list stores \(k_2\) nonredundant elements. The
time taken in step 1 is \(O(k_1 k_2)\). In step 2, the inputs to the β procedure are
\(k_1\) irreducible L-lists (each with \(k_2\) elements)
and an irreducible R-list of \(k_1\) elements (for \(u_1\) and
\(B_1\), respectively). It follows from Section IV-B-2 that
the β procedure constructs \(k_1\) irreducible L-lists for \(u_2\), and
each such irreducible L-list stores at most \(k_2 + k_3 - 1\)
nonredundant elements. The time taken in step 2 is
\(O(k_1 (k_2 + k_3))\). In step 3, the inputs to the γ procedure
are \(k_1\) irreducible L-lists (each with at most \(k_2 + k_3 - 1\)
elements) and an irreducible R-list of \(k_3\) elements (for \(u_2\)
and \(B_2\), respectively). It follows from Section IV-B-3 that
the γ procedure in the worst case constructs at most \(k_1 (k_4 + 1)\)
irreducible L-lists for \(u_3\), and these irreducible L-lists store a total of at most \(k_1 k_2 (k_2 + k_3 - 1)\)
nonredundant elements. (Note that these \(k_1 (k_4 + 1)\)
irreducible L-lists, \(k_1\) lists each store at most \(k_4\) elements, and
each of the remaining \(k_1 k_2 k_3\) lists stores at most \(k_2 + k_3 - 2\)
elements.) The time taken in step 3 is \(O(k_1 k_2 (k_2 + k_3))\).
In step 4, the inputs to the δ procedure are \(k_1 (k_4 + 1)\)
irreducible L-lists (of these irreducible L-lists, \(k_1\) lists each
store at most \(k_4\) elements, and each of the remaining \(k_1 k_2 k_3\)
lists stores at most \(k_2 + k_3 - 2\) elements) and an irreducible
R-list of \(k_3\) elements (for \(u_3\) and \(B_3\), respectively).
It follows from Section IV-B-4 that the δ procedure first constructs \(k_4 (k_4 + 1)\) R-lists for \(v_i\),
and these R-lists in the worst case store a total of at most \(k_1 k_2 (k_2 + k_3 - 3) + k_4 (k_4 + 1)\)
nonredundant elements. Next the δ procedure proceeds to merge these R-lists into one, and
then prunes redundant elements within the resulting R-list.
The time taken in step 4 is dominated by the merging process
and hence is \(O(k_1 k_2 (k_2 + k_3 + k_4)) \log (k_1 k_2)\).
By summing up the time taken in steps 1–4, we get the
total time complexity for obtaining the irreducible R-list
for the wheel at \(O(k_1 k_2 (k_2 + k_3)) \log (k_1 k_2)\). (Note
that this is substantially better than the "brute force" method
of first forming all \(k_1 k_2 k_3 k_4 k_5\) possible implementations
and then pruning redundant elements, which takes time
\(O(k_1 k_2 k_3 k_4 k_5 \log (k_1 k_2 k_3 k_4 k_5))\).

V. IMPROVEMENT

In this section, we would like to point out a technique
which may improve the efficiency of our algorithm.
Again, consider the wheel, say \(W\), shown in Fig. 6, and
let rectangular block \(B_i\) have \(k_i\) nonredundant implementations.
By using our algorithm, the time to obtain all nonredundant
implementations of \(W\) is \(T = O(k_1 k_2 (k_2 + k_3 + k_4 + k_5) \log (k_1 k_2))\). In Fig. 12 we show three wheels (named
\(W_1\), \(W_2\) and \(W_3\)) which have the same form as \(W\) and are
obtained by rotating \(W\) counterclockwise by 90°, 180°,
and 270°, respectively. The corresponding binary trees
representing the three wheels are also shown in Fig. 12.
By using our algorithm based on these binary trees, the
individual time to obtain all nonredundant implementations
corresponding to \(W_1\), \(W_2\), and \(W_3\) is as follows:

\[
T_1 = O(k_5 k_3 (k_2 + k_3 + k_4) \log (k_1 k_2)),
\]

\[
T_2 = O(k_4 k_1 (k_2 + k_3 + k_5) \log (k_1 k_2)),
\]

\[
T_3 = O(k_5 k_2 (k_2 + k_4 + k_5) \log (k_1 k_2)).
\]

Clearly, in the worst case, \(T = T_2 = T_3\). If \(k_5 k_3 (k_2 + k_1 + k_4) \log (k_1 k_5) << k_1 k_4 (k_2 + k_3 + k_5) \log (k_1 k_4),\)
we expect \(T_1 < T\). In this case, we recommend first rotating \(W\) to \(W_1\) and obtaining the list of all nonredundant
implementations for \(W_1\), and rotating \(W_1\) back to \(W\) later.
On the other hand, if \(k_5 k_3 (k_2 + k_1 + k_4) \log (k_1 k_5)\) is not
much less than \(k_1 k_4 (k_2 + k_3 + k_5) \log (k_1 k_4),\) it is hard
to tell whether \(T_1\) is less than \(T\). Also, there is a cost
associated with rearranging the R-lists for the blocks after
each rotation to satisfy properties \(P_1\) and \(P_2\), which makes
using \(W_1\) more costly. Therefore, we recommend
using \(W_1\) instead of \(W\) only if \(k_5 k_3 (k_2 + k_1 + k_4) \log (k_1 k_5) < \theta_k (k_4(k_2 + k_3 + k_5) \log (k_1 k_4),\) where \(\theta\) is a user
input parameter with \(0 < \theta < 1\).

VI. EXTENSION TO GENERAL FLOORPLANS

Every floorplan can be viewed as having been 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 5 and represents
a more complicated nonslicing floorplan. As before, we
need to construct an irreducible R-list to store all nonre-
Fig. 13. A general example.

dundant 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 be considered:

- Case 1: If the irreducible \( R \)-list associated with \( u \) can be obtained by repeatedly using the \( \alpha \) procedure, \( \beta \) procedure, or \( \gamma \) procedure and finally using the \( \delta \) 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 Fig. 13 for an example).
- Case 2: If the irreducible \( R \)-list associated with \( u \) cannot be obtained from the previous case, then we have to decompose the rectangular block, which is 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 the \( \alpha \) procedure, \( \beta \) procedure, or \( \gamma \) 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 by trying all combinations. Finally, rearrange the list of \( u \) such that the resulting \( R \)-list remains irreducible.

VII. EXPERIMENTAL RESULTS

We have implemented our algorithm in C language on a Sun 3/50 workstation running Unix 4.2BSD. Our current implementation does not incorporate the improvement technique introduced in Section V. The experimental results are very encouraging. The ten test examples we used are described as follows.

- EX1–EX5: The 25-module floorplan shown in Fig. 14, where each module has three possible implementations: \{4 \times 1, \ 2 \times 2, \ 1 \times 4\}, four possible implementations: \{6 \times 1, \ 3 \times 2, \ 2 \times 3, \ 1 \times 6\}, five possible implementations: \{16 \times 1, \ 8 \times 2, \ 4 \times 4, \ 2 \times 8, \ 1 \times 16\}, six possible implementations: \{12 \times 1, \ 6 \times 2, \ 4 \times 3, \ 3 \times 4, \ 2 \times 6, \ 1 \times 12\} and eight 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\}, respectively.
- EX6: The 24-module floorplan in [7] shown in Fig. 15.
- EX7: The 120-module floorplan shown in Fig. 16, where each rectangular block represents the floorplan used in EX6.
- EX8–EX10: The 40-module floorplan shown in Fig. 17, where each module in EX8, EX9, and EX10 has 10, 20, and 30 possible implementations (randomly generated with aspect ratio between 0.5 and 2), respectively.

We first compared our algorithm with the branch-and-bound optimal algorithm in [7] on EX1–EX5. In fact, the branch-and-bound program we used is an implementation of [7] 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 Fig. 14, which cannot be partially processed by the slicing floorplan algorithm, to be our test examples EX1–EX5. The experimental results of EX1–EX5 between our program and the branch-and-bound program in [2] are reported in Table I. Here BB denotes the branch-and-bound algorithm and OPT denotes our algorithm. The running time of our algorithm on each of the test examples EX1–EX5 is substantially less than that of the branch-and-bound algorithm. Since the branch-and-bound algorithm uses an enumeration tree to represent the states of the search space, each node visited by the algorithm corresponds to a possible implementation of a partial floorplan. Here we also compared the number of nodes visited by our algorithm and the branch-and-bound algorithm. Note that the number of nodes visited by our algorithm is defined as the total number of implementations ever generated for all internal nodes in the binary floorplan tree since each such implementation also corresponds to a partial floorplan. Again, our algorithm visited substantially fewer nodes than the branch-and-bound algorithm. For EX4 and EX5, since the branch-and-bound program in [2] ran for two days and did not terminate, we could not obtain the data for the number of visited nodes.
and hence put a dash in the corresponding entry of Table I to indicate that the data are unavailable.

We next compared the experimental results of EX6 with those of [7]. Because Wimer et al. presented their experimental results of EX6 only in terms of the ratio of the number of visited nodes to the total number of leaves in the enumeration tree, the running time of EX6 in [7] is unavailable; hence a dash is put in the corresponding entry of Table I. However, our algorithm checked fewer nodes than the branch-and-bound algorithm [7]. In order to demonstrate that our algorithm is able to handle large floorplans, we ran our program on EX7. The floorplan used in EX7 is primarily a wheel where each rectangular block corresponds to the 24-module floorplan used in EX6; hence it is a 120-module floorplan. The total number of possible implementations of EX7 is about $3.45 \times 10^9$ and our algorithm ran for only 14.8 s to obtain the optimal solution. The experimental results of EX6 and EX7 are reported in Table I.

Finally, we considered examples of hierarchical floorplans of order higher than 5 in EX8–EX10. Since the corresponding floorplan tree of this floorplan has all its slicing nodes at lower levels, we first used the slicing floorplan algorithm [5] to generate all nonredundant implementations for each of the slicing nodes, and then applied our algorithm and the branch-and-bound algorithm, respectively, to the resulting floorplan. Note that in these three examples, each module has a larger number of possible implementations. The final experimental results of EX8–EX10 are also reported in Table I. Again, our algorithm performed much better than the branch-and-bound algorithm. In particular, for EX10, where the branch-and-bound algorithm ran for two days and did not terminate, our algorithm obtained the optimal solution in 41.5 s.

The above experimental results indicate that our algorithm, based on an extension of the technique in [5], did prune very large numbers 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 dual polar graphs [7] to calculate the longest paths. Consequently, our algorithm is able to run more efficiently than the branch-and-bound algorithm in [7].

VIII. CONCLUDING REMARKS

In this paper we have presented an optimal algorithm for the floorplan area optimization problem. Our algorithm is based on an extension of the technique in [5] and can be applied to generate the optimal floorplan with prespecified aspect ratio. Experimental results presented in Section VII indicate that our algorithm is very efficient and can successfully handle very large floorplans. Instead of considering all possible implementations, our algorithm eliminates a large number of implementations which are not relevant to the optimal solution. However, we would like to point out that the time complexity of our algorithm in the worst case is still exponential. In fact,
we showed in [6] that any floorplan area optimization algorithm based on extensions of the technique in [5] has exponential time complexity in the worst case. The proof is based on the fact that for a very special type of floorplan we can carefully design the implementations of the modules such that the total number of nonredundant implementations for the entire floorplan is exponential in the number of modules. (The special floorplans which we considered are of the form of a nested set of wheels obtained by initially partitioning a rectangle into a wheel and then recursively partitioning the rectangle in the middle of the wheel into another smaller wheel.) Of course this kind of worst-case example does not represent typical floorplans encountered in practice. We want to emphasize that our algorithm performs well although it is of exponential time complexity in the worst case.

In Section IV-B-5, we showed that if the number of nonredundant implementations of the five blocks in a wheel are \(k_1, k_2, k_3, k_4, k_5\), respectively, the total number of nonredundant implementations of the wheel is bounded by \(k_1 k_2 (k_2 + k_3 + k_4 - 3) + k_1 (k_4 + k_5 - 1)\). In practice, this bound is substantially less, which is the reason why our algorithm has the short running times reported. We observe experimentally that in almost all cases the number of nonredundant implementations is at most \((1 + \epsilon)(k_1 + k_2 + k_3 + k_4 + k_5)\), where \(\epsilon < 1\) is a small number, and in many cases it is less than \(k_1 + k_2 + k_3 + k_4 + k_5\). This means that in practice the number of nonredundant implementations for a wheel grows at most only a little bit faster than the sum of the numbers of nonredundant implementations of the modules. In fact, if \(\epsilon \leq 0\) for every internal node of degree 5 (corresponding to a wheel) in the floorplan tree, it is easy to prove that the number of nonredundant implementations of the floorplan is bounded by the sum of the numbers of nonredundant implementations of the modules; hence our algorithm runs in polynomial time in this case.

REFERENCES