Trees, Slices, and Wheels: On the Floorplan Area Minimization Problem

Ana Maria de Almeida
Dep. de Matemática, Fac. Ciências e Tecnologia, Universidade de Coimbra, Coimbra Portugal

Rosália Rodrigues
Dep. de Matemática, Universidade de Aveiro, Aveiro, Portugal

Hierarchical approaches to *floorplan design* usually use trees to represent hierarchical floorplans, defining topological relations between a set of components to be placed on a (circuit) board. This placement is then optimized to minimize chip area, perimeter, or other such cost measurements. This work places the problem of minimizing hierarchical floorplans of order *k* within the appropriate theoretical framework and presents an algorithm that uses exclusively the *k*-ary representation tree to determine the best or minimal solution to a conflicting bicriteria problem. The algorithm does not need predefined partitioning schemes and determines the orientation of each individual component. It can deal both with five-module wheels, as well as slices, and can handle an arbitrary number of realizations for each individual component. © 2003 Wiley Periodicals, Inc.

Keywords: multicriteria problem; K-ary trees; nondomination; floorplan optimization

1. INTRODUCTION

The problem to be dealt with can be informally described as follows: Given a set of rectangular blocks (components) obeying a predefined positioning scheme (topological design), proceed to position them on a rectangular board so that the overall area is minimized.

This means that, since the topology of the placement has already been decided, we simply want to minimize the area for positioning the modules: in other words, to find the minimal floorplan. A *floorplan* can be viewed as an enclosing rectangle subdivided, by horizontal and vertical line segments, into nonoverlapping rectangles (see Fig. 1) that define the relative positions of the components in the plane. Although this minimization problem is classified as strongly NP-complete for general (nonhierarchical) floorplans [4], this classification does not hold for hierarchical floorplans, that is, floorplans obeying a recursive partitioning process.

A floorplan is then called a *hierarchical floorplan of order k* if its enveloping rectangle can be partitioned into at most k subrectangles, each one being either a simple rectangle (without any division lines) or itself a hierarchical floorplan of order k. The particular case of $k = 2$ is called a *slicing floorplan* since the only allowed partition patterns are vertical or horizontal slices.

The cases $k = 3$ and $k = 4$ can easily be reduced to slicing floorplans [5]. Therefore, the first nonsliceable motif occurs for $k = 5$, where floorplans may include patterns resembling "wheels" with five or more components, as the ones in Figure 2.

A rectangle without internal line segments will be called a *basic rectangle* and, from now on, will be identified with the rectangular component to be placed in it. A *partitioning scheme* is a sequence of partitioning patterns needed to produce a realization (or implementation) of a floorplan (see Fig. 4 for an illustration).

One of the most popular representations for a hierarchical floorplan of order k is the *floorplan tree,* that is, a *rooted k-ary tree* that describes the topological relations (relative positions) between components (Fig. 3). Each leaf node represents a basic rectangle and each nonleaf node, corresponding to a set of components, represents an enclosing rectangle (floorplan). This, of course, implies that the root of the tree is identified with the overall placement area.

For each tree T, there is an associated set of pairs of real numbers that represent the feasible dimensions of possible rectangular regions (or layouts) for T. Therefore, if T is rooted at r, we define $\mathcal{F}(T_r)$ as the set of all pairs of dimensions for possible floorplans that represent the rectangular regions associated with node r. In general, if T_v is the subtree of T rooted at v, then $\mathcal{F}(T_v)$ represents all the

Received October 2001; accepted February 2003
Correspondence to: A. M. de Almeida
Contract grant sponsor: MCT
This work was carried out at the CISUC–Centro de Informática e Sistemas da Universidade de Coimbra Research Center

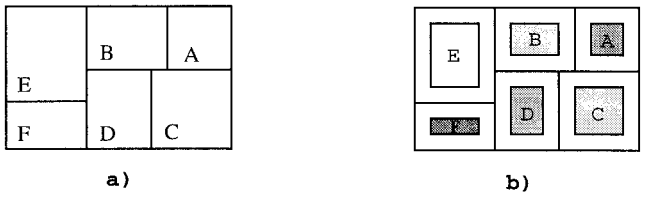

FIG. 1. Slicing floorplan: (a) without components; (b) with components.

feasible pairs of rectangular dimensions for floorplans associated with v.

2. AREA MINIMIZATION

Since the floorplan tree is predefined and the relative positions of the components must be maintained, the optimization of the placement area should be performed using this representation. This means that all the necessary optimization procedures should use and preserve the structure of the tree, not only for efficiency reasons but also because it will still be needed to carry out the actual placement.

Associated with each rectangle is a pair of values, $(h, w) \in \mathbb{R}^2$, that represent its dimensions, height, and width. The problem to be dealt with can then be formally stated as, given the floorplan tree T rooted at r, and a nondecreasing* function $\phi : \mathbb{R}_+^2 \to \mathbb{R}$, minimize $\phi(h, w)$ over all possible orientations of the components for floorplans associated with T:

$$\min_{(h,w)\in\mathcal{F}(T_r)} \phi(h, w).$$

This is a well-known bicriteria problem with conflicting objectives† commonly called *compaction.* The most popular objective functions within this framework are the *area,* $\phi(h, w) = hw$, or, as a means to produce a more balanced aspect ratio, the *semiperimeter,* $\phi(h, w) = h^2 + w^2$. Nevertheless, any other suitable nondecreasing function can be used.

For a leaf node, that is, for a basic rectangle, a *feasible solution* is no more than a pair (h, w) from the list of all possible realizations for the component it represents. In the case of a (sub)floorplan, that is, an intermediate node or the root of the tree, the corresponding solution is still a pair of side dimensions for the rectangle, but now it also represents the orientations for the basic rectangles (leaf nodes) enclosed within the represented rectangular region. Therefore, it implicitly defines the partitioning sequence that leads to that particular floorplan.

Slicing floorplans are represented by binary trees, where each node **can** be labeled (either before or after minimization) with the direction—horizontal or vertical—of the slice needed to position its children, thus specifying the partitioning scheme to be used upon effective placement of the layout, as shown in Figure 4.

Although the area minimization problem for hierarchical floorplans is classified as (weakly) NP-complete [2], this classification does not hold when only slicing patterns are used [1]. In this particular case, under very natural and even desirable restrictions, there exist polynomial algorithms for minimizing the occupied area.

2.1. Slicing Floorplans: Binary Trees

In [1], the authors prove that the optimal solution for the area minimization problem of slicing floorplans is a nondominated solution for the minimization of $\phi(h, w)$ over **all** orientations of components for the floorplans associated with a binary tree T.

Let T be rooted at r. We define the set of *dominated* solutions for this problem as the set of real pairs:

$$\mathcal{D}(T_r) = \{(a, b) \in \mathcal{F}(T_r) \mid \exists\, (c, d) \in \mathcal{F}(T_r) : (c \le a \wedge d < b) \vee (c < a \wedge d \le b)\}.$$

This means that a nondominated solution will be a pair for which there is no other pair that is **strictly better in both dimensions,** that is, a *nondominated* solution will then be one that belongs to

$$\mathcal{N}(T_r) = \mathcal{F}(T_r) - \mathcal{D}(T_r).$$

Thus, minimizing ϕ over all possible orientations is equivalent to minimizing ϕ over all nondominated solutions and, therefore, the optimal solution lies in the set $\mathcal{N}(T_r)$ [1].

In 1983, Stockmeyer [4] presented a polynomial algorithm that, given a sequence of slicing cuts, that is, **using a predefined partitioning sequence,** determines the corresponding nondominated solution set in which the optimal solution lies. Naturally, this optimal solution is only optimal in terms of the given slicing sequence and so is only **locally optimal.** The algorithm has a worst-case space and time complexity of $O(dn)$, where n is the number of components of the placement and d is the depth of the tree. This means that we will have $O(n \log n)$ complexity for balanced trees and $O(n^2)$ otherwise.

Shi [3], in 1996, described a new algorithm that, under the same conditions and using a new data structure for storing and updating nondominated solutions, has a worst-case space and time complexity of $O(m \log m)$, where m is

FIG. 2. Examples of nonsliceable partitioning patterns: five- and seven-module wheels.

* A function, $\phi : \mathbb{D} \subset \mathbb{R}^2 \to \mathbb{R}$, will be called *nondecreasing* if $(a < c \wedge b \le d) \vee (a \le c \wedge b < d) \Rightarrow \phi(a, b) \le \phi(c, d)$, $\forall (a, b), (c, d) \in \mathbb{D}$.

† When you consider that you want to minimize both dimensions, the best solution for one of them (almost) always increases the other dimension.

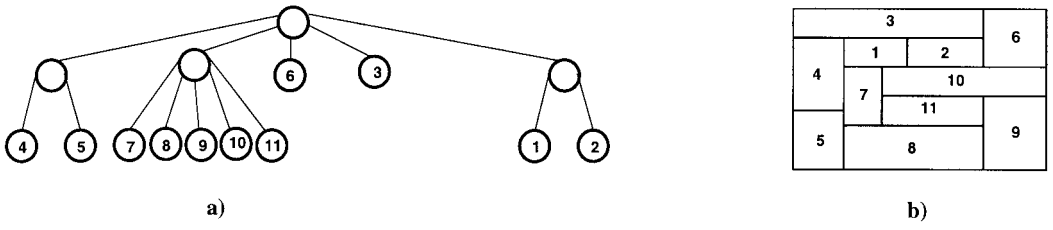

FIG. 3. (a) Hierarchical model: floorplan tree T; (b) board geometry: a floorplan for T.

the total number of realizations for the components and, therefore, m is a function of n. This means that there are different possible pairs to implement each component in a rectangular shape, while Stockmeyer's algorithm only allowed for the use of one rectangular shape for each component, positioned up, (a, b), or down, (b, a). Even though Shi's algorithm is an extension of Stockmeyer's algorithm, it turns out to be faster than the previous one although it handles an arbitrary number of realizations. Moreover, the author proves that $\Omega(m \log m)$ is a lower bound on the running time of any area optimization algorithm for slicing floorplans with fixed partitioning sequences.

The problem of finding the optimal (or best) partitioning sequence for floorplan trees is in itself a complex problem but Almeida et al. [1] presented a new approach that, although based on Stockmeyer's algorithm, extends it, achieving **global optimization,** namely, it needs **no predefined partitioning** and the algorithm decides which one produces the minimal value for the area. This means that the optimal solution for this algorithm, besides defining the best orientation for the basic rectangles, also *implicitly defines the sequence of cuts* that leads to the minimization of the placement area. In [1], it is also proved that the optimal overall solution for the minimization of the placement area is a nondominated one and also *that only nondominated solutions are needed, whatever the tree level, to achieve minimization at the root of the tree,* thus presenting a theoretical framework that ensures optimality.

Because there is no predefined sequence of slicing cuts, to assure that no nondominated solution is left out, both cutting directions must be considered. Since Stockmeyer's algorithm only uses the predefined slicing direction, at first it would appear that the computational effort of the global algorithm, when compared with Stockmeyer's, should be much greater. In reality, it is easy to show that, at the root of a tree with n leaves (representing a positioning with n components), there can be no more than nM nondominated solutions [2], where M is the largest dimension for all the possible realizations for components. The fact that only binary trees are used and this latter result make it simple to prove that, given a slicing tree with n leaves, each representing a component with, at most, k realizations, the algorithm described in [1] has worst-case space complexity of $O(d(k + nM))$ and worst-case time complexity of $O(dMn + k)$, where, again, d stands for the depth of the tree.

Since we have, for a balanced tree, $d = \log n$, these complexities present, in fact, good bounds when compared with Shi's. Also, of course, this also means that, although solving a more general instance for area minimization, if only one rectangular shape is used, the global optimal algorithm has the same complexity order as Stockmeyer's algorithm. Moreover, although in the more general case the worst-case complexity turns this algorithm into a pseudo-polynomial one, this is not as serious as it might seem, for these algorithms are generally used within VLSI layouts, where $k \ll n$ and M is not dependent on n.

2.2. Hierarchical Nonsliceable Floorplans: K-ary Trees

To proceed for more elaborate patterns, we must use nonslicing floorplans. The simplest ones of these are called *hierarchical floorplans of order 5,* where the only allowed

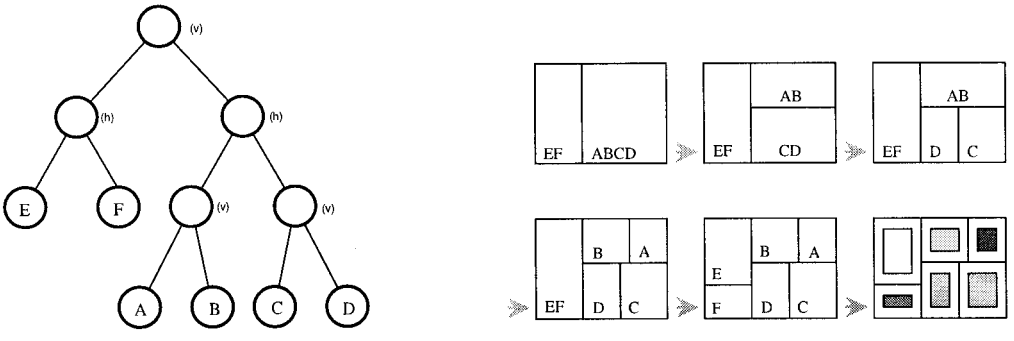

FIG. 4. Slicing tree and sequence of slicing cuts that realize the floorplan it represents.

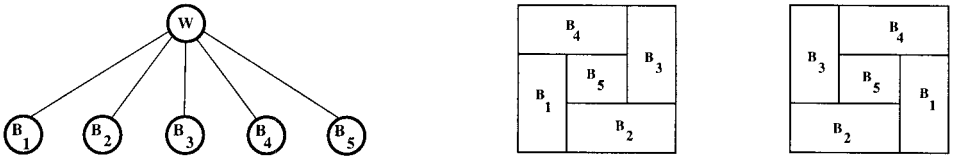

FIG. 5. 5-ary node and possible realizations for a wheel.

partitioning patterns are the two vertical and horizontal slices or the five-module wheels.[‡]

This problem generalizes differently from the slicing problem since the wheels are, to some extent, previously fixed partitionings. These patterns are represented by 5-ary nodes in the trees, which means that the trees used are no longer binary. In a *5-ary tree* for the representation of hierarchical floorplans, a node can only be one of three types: a leaf node, representing a component; a binary node, representing a rectangular region with a slice; or a complete 5-ary node, representing a wheel.

Since the nondominance theory applied for the slicing floorplans is based on the existence of real pairs representing feasible rectangular dimensions, and wheels are nothing more than rectangular regions only presenting a special inner arrangement, this theory can still be applied for nonsliceable hierarchical floorplans. What remains to be done is to determine how to construct wheels and to choose only the nondominated ones, to extend our algorithm to hierarchical floorplans of order 5.

In 1990, Wang and Wong [6] presented an extension of Stockmeyer's algorithm that works with hierarchical floorplans of order 5, which, of course, means that it needs a predefined partitioning scheme. To optimize the area occupied by wheels, the corresponding hierarchical representation (the 5-ary node[§]) is transformed into a special binary tree, where each internal node represents an L-shaped arrangement of modules within the wheel. Once this transformation is complete, several procedures, *specialized in each of the new different nodes thus obtained,* are used to handle the compaction. For normal binary nodes, that is, for slices, Stockmeyer's algorithm is used.

Using predefined partitioning schemes, Pan et al. [2] proposed a new approach to handle five-module wheels that does not need rearrangements of the representation tree and has time complexity of $O((nM)^2\log(nM))$ and space complexity of $O(n^2M)$, where M stands for the largest dimension for all possible realizations. Based on Stockmeyer's idea, they introduced a new concept of "nonredundant solutions" and "support sets" of nonredundant solutions for wheels and used Stockmeyer's algorithm when slices are involved. This redundancy concept follows closely the notion of dominance, but, unfortunately, the procedures that build their support sets are not able to find all nondominated solutions, getting stuck in local optima. Consequently, this yields only a heuristic method for the more general problem that we seek to solve. Nevertheless, as the strategy found in [2] does not imply rearranging the floorplan tree, nor is it necessary to use secondary representation structures, and since their reasoning had a strong resemblance to nondominance theory, we tried to overcome those deficiencies. We present here a new algorithm that **ensures that the complete nondominated sets are built,** thus achieving global optimization.

2.3. Construction of Nondominated Sets for 5-ary Wheels

First, notice that, in the following algorithms, each $\mathcal{N}(T_r)$ set will be represented as *a list of pairs,* $\{(h_i, w_i), i = 1, \ldots, m\}$, satisfying the following order [4]:

$$(h_i > h_{i+1}) \wedge (w_i < w_{i+1}),\ 1 \le i < m. \qquad (1)$$

This ordering is always possible since all the pairs in the list are nondominated.

Figure 5 shows that there are only two different rectangular shapes for a 5-ary node. It can also be easily checked that each one of this possible wheel is just the mirror image of the other. We can therefore, and without loss of generality, use only the left wheel of Figure 5.

Due to the special configuration of a wheel W, it can have both width and height given either by two or three inner modules, depending on the ones that determine the dimension large enough to accommodate the wheel. For simplicity, let us denote by b_i a nondominated solution in $\mathcal{N}(T_{B_i})$. Then, a feasible solution for W can be represented as $r = [b_1; b_2; b_3; b_4; b_5]$, where each b_i is a given solution in $\mathcal{N}(T_{B_i})$ and the sequence in r mimics that shown in the rectangle wheel on the left of Figure 5.

To define the width $w(r)$ and the height $h(r)$, there are essentially 10 different possibilities[#]:

[‡] In fact, to reduce the size of search space during floorplan design, and since the cost of the partitioning process increases rapidly with p, typically, $k \le 5$ [2].

[§] This approach can be extended for wheels of higher order and the authors do present another extension for general hierarchical floorplans, illustrated with a seven-module wheel. Of course, the number of special nodes and associated procedures grows with the order of the wheel!

[#] These groupings resemble closely the classes 1, 5, and 9 found in [2]. Although equivalent, the ordering used here is different but it will be necessary to assure that no feasible nondominated solution is left out!

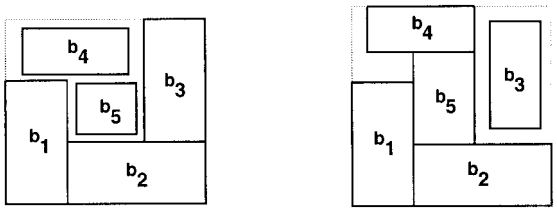

FIG. 6. $w(r) = w(b_1) + w(b_2)$ and $h(r) = h(b_2) + h(b_3)$ or $w(r) = w(b_1) + w(b_2)$ and $h(r) = h(b_2) + h(b_5) + h(b_4)$.

- $w(r)$ and $h(r)$ are both determined by two blocks—**Case 1**

  1. $w(r) = w(b_1) + w(b_2)$, $h(r) = h(b_2) + h(b_3)$;
  2. $w(r) = w(b_4) + w(b_1)$, $h(r) = h(b_1) + h(b_2)$;
  3. $w(r) = w(b_3) + w(b_4)$, $h(r) = h(b_4) + h(b_1)$;
  4. $w(r) = w(b_2) + w(b_3)$, $h(r) = h(b_3) + h(b_4)$;

- $w(r)$ is determined by two blocks while $h(r)$ is determined by three blocks—**Case 2:**

  5. $w(b_4) + w(b_1)$, $h(r) = h(b_1) + h(b_5) + h(b_3)$;
  6. $w(b_2) + w(b_3)$, $h(r) = h(b_1) + h(b_5) + h(b_3)$;
  7. $w(b_1) + w(b_2)$, $h(r) = h(b_2) + h(b_5) + h(b_4)$;
  8. $w(b_3) + w(b_4)$, $h(r) = h(b_2) + h(b_5) + h(b_4)$;

- $w(r)$ and $h(r)$ are both determined by three blocks—**Case 3:**

  9. $w(r) = w(b_1) + w(b_5) + w(b_3)$, $h(r) = h(b_2) + h(b_5) + h(b_4)$;
  10. $w(r) = w(b_2) + w(b_5) + w(b_4)$, $h(r) = h(b_1) + h(b_5) + h(b_3)$.

Each one of these possibilities does, in fact, turn out to be a class where the different arrangements between modules used in the definition of $w(r)$ and $h(r)$ are no more than different 90° rotations for the same wheel. Of special importance is the fact that, when searching for nondominated wheels, and since we only have to keep the wheel with smaller height, we can merge Cases 1 and 2:

- $w(r)$ is determined by two blocks while $h(r)$ is either determined by two or three blocks:

  1. $w(r) = w(b_1) + w(b_2)$, $h(r) = h(b_2) + h(b_3)$ **or** $h(r) = h(b_2) + h(b_5) + h(b_4)$;
  2. $w(r) = w(b_4) + w(b_1)$, $h(r) = h(b_1) + h(b_2)$ **or** $h(r) = h(b_1) + h(b_5) + h(b_3)$;
  3. $w(r) = w(b_3) + w(b_4)$, $h(r) = h(b_4) + h(b_1)$ **or** $h(r) = h(b_2) + h(b_5) + h(b_4)$;
  4. $w(r) = w(b_2) + w(b_3)$, $h(r) = h(b_3) + h(b_4)$ **or** $h(r) = h(b_1) + h(b_5) + h(b_3)$;

Moreover, possibilities 1–4 still correspond to rotations for the same wheel. This then means that we are only going to need two different kinds of procedures for the construction of wheels: one to build wheels where the width is determined by two blocks and another to build wheels where the total width is determined by three blocks.

Note that we elect, as representatives for each of the cases, the examples shown in Figures 6 and 7, just for the sake of procedure construction and explanatory simplicity. The final algorithm must, of course, consider all the different possibilities, that is, the different rotations. Note also that block b_5 can never change position with respect to the other four, otherwise, we would be interfering within the restriction of the wheel design which is not allowed. The other blocks are the ones that can rotate around b_5 since it will not change the requirements of the design.

2.3.1. Nondominated Wheels: Cases 1 and 2. To assure that no nondominated wheel is left out, for the construction of r, we will test all possible pairs b_1; b_2, that is, all $\mathcal{N}(T_{B_1})$ and $\mathcal{N}(T_{B_2})$ lists will be scanned for combinations r that might produce a nondominated wheel. Starting with the lists of nondominated solutions for B_1 and B_2 (being scanned in decreasing order of their heights), we then have a first procedure to begin building the list $\mathcal{N}(T_W)$ of nondominated wheels for a 5-ary node W (and where the *stopi, i* = 3, 4, 5, are Boolean variables initialized with a *false* value):

Procedure 1

for each $b_1 \in \mathcal{N}(T_{B_1})$ and **for each** $b_2 \in \mathcal{N}(T_{B_2})$
  $hmax = \infty$
  while not stop3 **do** next b_3
    if $w(b_3) < w(b_2)$* **then**
      if $h(b_1) + h(b_2) < hmax$ **then**
        while not stop5 **do** next b_5
        if $w(b_5) \leq w(b_2) - w(b_3)$** **then**
          stop5 = true
        while not stop4 **do** next b_4
        if $w(b_4) \leq w(b_1) + w(b_2) - w(b_3)$*** **then**
          stop4 = true
          if $h(b_4) + h(b_5) \leq h(b_3)$ **and** $h(b_4) \leq h(b_2) + h(b_3) - h(b_1)$ **then**
{a case 1 type wheel was found}
            add $r = b_1b_2b_3b_4b_5$ to $\mathcal{N}(T_W)$
            stop3 = true
          otherwise if $h(b_1) \leq h(b_2) + h(b_5)$ **then**
{a case 2 type wheel was found}
            add $r = b_1b_2b_3b_4b_5$ to $\mathcal{N}(T_W)$
            $hmax = h(b_1) + h(b_4) + h(b_5)$
    otherwise stop3 = *true*

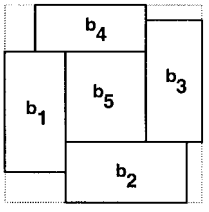

FIG. 7. $w(r) = w(b_1) + w(b_5) + w(b_3)$ and $h(r) = h(b_2) + h(b_5) + h(b_4)$.

First, notice that the nondominated lists for B_i, $i = 3, 4, 5$, *are ordered by increasing height.* Then, for each possible b_2; b_1 that is, for each possible width $w(b_1) + w(b_2)$, this procedure *linearly* tries to find the first b_3 that minimizes height. Some structural and design rules were embedded in the algorithm through the use of the *if* tests, thus avoiding the construction of deficient or obviously dominated wheels. Therefore, the (*) test serves to assure that a b_3 is only used if there is enough width to insert the fifth module. The ordered $\mathcal{N}(T_{B_3})$ list is *linearly* scanned, seeking the first pair whose height meets this condition. Only when such a b_3 is found, does it start to look for a b_5, again using the (**) test to ensure that only the first one that fills in the hole left between b_1 and b_3 is chosen. This means that, because of the ordering of the lists, if such a b_5 is found, it will be the one for which

$$w(b_5) = \max\{w(b) : b \in \mathcal{N}(T_{B_5}) \ \wedge \ w(b) \leq w(b_2) - w(b_3)\}.$$

But, according to (1), this also means that this b_5 is the one where

$$h(b_5) = \min\{h(b) : b \in \mathcal{N}(T_{B_5}) \ \wedge \ w(b) \leq w(b_2) - w(b_3)\}.$$

Having found such a pair, we can stop the scanning of list B_5 and start looking for the next block to fill in the wheel. The (***) test stops in the first or, which is the same, the b_4 with smallest height that fits in the hole left by the other four blocks. Again, if it exists, the scanning of the B_4 list is stopped.

If all the necessary conditions are fulfilled, a feasible wheel is built and we have now two possibilities: Either it is a Case 1 wheel or a Case 2 wheel. In the first case, there is no need to advance in the $\mathcal{N}(T_{B_3})$ list, for all the remaining pairs will have greater height; thus, since the width remains constant, any feasible wheel would be a dominated one. Since this solution corresponds with the wheel r such that

$$\begin{aligned} h(r) &= \min\{h(b_2) + h(b_3) : b \in \mathcal{N}(T_{B_3}) \ \wedge \ w(r) \\ &\quad = w(b_1) + w(b_2)\}, \end{aligned}$$

just *add* this new wheel to $\mathcal{N}(T_W)$ and stop, choosing a new pair b_1b_2.

In the second case, there might still be a possible Case 1 wheel with smaller height; thus, *add* this new wheel to the $\mathcal{N}(T_W)$ list and continue scanning the $\mathcal{N}(T_{B_3})$ list. To prevent an unnecessary scanning, the variable *hmax* is updated to the value $h(r) = h(b_1) + h(b_4) + h(b_5)$ of the Case 2 solution that was just added to the list. If any new b_3 has height such that $h(b_2) + h(b_3) \geq hmax$, then the scanning of $\mathcal{N}(T_{B_3})$ can stop since any possible Case 1 wheel would be a dominated solution.

2.3.2. Nondominated Wheels: Case 3. The central role is now played by b_5 and so the corresponding procedure to construct nondominated wheels is built around the scanning of the entire $\mathcal{N}(T_{B_5})$ list and can be simply described as follows:

Procedure 2

for each $b_5 \in \mathcal{N}(T_{B_5})$
  for each $b_2 \in \mathcal{N}(T_{B_2})$
    if $Search(b_2, b_5; b_1) =$ found
      if $Search^*(b_1, b_5; b_4) =$ found
        if $Search(b_4, b_5; b_3) =$ found **and** $w(b_2) \leq w(b_3) + w(b_4)^*$
          add $r = b_1b_2b_3b_4b_5$ to $\mathcal{N}(T_W)$

The (*) test is necessary to avoid producing degenerate wheels by this procedure. The order of search used in $\mathcal{N}(T_{B_1})$ and $\mathcal{N}(T_{B_3})$ lists is by *decreasing order of heights.* The $Search(b_k, b_j; b_i)$ procedure linearly scans the nondominated list for B_i, using once more a *decreasing order of heights,* so that it finds the first b_i that satisfies $h(b_i) \leq h(b_k) + h(b_j)$, which is also the one such that satisfies

$$h(b_i) = \max\{h(b) : b \in \mathcal{N}(T_{B_i}), h(b) \leq h(b_k) + h(b_j)\},$$

and so this b_i is the one that minimizes width. There is an exception for $Search^*$ where this routine finds the first b_i that satisfies

$$h(b_4) = \min\{h(b) : b \in \mathcal{N}(T_{B_4}), w(b) \leq w(b_1) + w(b_5)\},$$

thus being the one with smallest height to fit in with the fixed b_1b_5.

As a last remark on these procedures, we should note that the *add* instruction that joins a new wheel to the $\mathcal{N}(T_W)$ list only does so if it finds that it constitutes, in fact, a nondominated solution with respect to the ones already in the list. Only once it is known that it still is nondominated is it inserted in the list, according to the order (1) proposed by Stockmeyer [4], thus preventing the insertion of dominated solutions. As this test for nondominance is really a test about ordering, it works both ways, that is, if, during the previously described test, it becomes apparent that this new solution dominates any of the solutions already in the list, the dominated solutions are deleted and only the new one (or nondominated one) is included.

Finally, it is easy to see that the worst-case complexities for both the procedures presented here are $O(k^2)$ for space and $O(k^3)$ for time complexity, where $k = \max|\mathcal{N}(T_{B_i})|$, $i = 1, \ldots, 5$.

2.4. Hierarchical Floorplan Compaction Algorithm

The combination of procedures 1 and 2 together with the algorithm in [1] for slicing patterns produces a general algorithm to perform the optimal compaction of hierarchical floorplans of order 5, for an arbitrary number of possible realizations for each component. This area minimization

TABLE 1. Comparison of algorithms using the examples found in [2].

		No. realizations					
			Examined			Time (seconds)	
Example	Blocks	Total	*PSL0*	*Beta0*	Area	*PSL0*	*Beta0*
1	25	3^{25}	168	88	121	0.002	0.001
2	25	4^{25}	365	148	176	0.003	0.001
3	25	5^{25}	776	304	484	0.004	0.003
4	25	6^{25}	994	398	352	0.007	0.004
5	25	8^{25}	2031	783	660	0.021	0.007
6	24	2.03×10^{16}	441	232	1024	0.003	0.003
7	125	3^{125}	865	465	841	0.007	0.006
8	125	4^{125}	1930	809	1044	0.012	0.009
9	125	5^{125}	4730	1799	2500	0.030	0.017
10	125	6^{125}	5723	2324	1800	0.053	0.023
11	125	8^{125}	12785	4769	3477	0.182	0.060
12	120	3.45×10^{81}	2382	2111	5842	0.042	0.042

algorithm, called *Beta,* is applied in a *bottom-up* fashion to a floorplan tree, T-5, to obtain the **complete** list of nondominated solutions for each and every node in T-5.

Beta

◦ *For each node V* of T-5:

- Initialize an empty list for $\mathcal{N}(T_V)$;
- **If** V is **binary:**
  Orderly *add* nondominated solutions to the list, according to the procedures found in [1];
- **Else if** V is **5-ary:**
  –**While** there are still rotations to explore **do**
  Turn the node 90° to the left
  Use **procedure 1** to orderly *add* nondominated solutions for Cases 1 and 2;
  –Use **procedure 2** to orderly *add* nondominated solutions for Case 3;
  –Turn the node 90° to the left and use **procedure 2** again.

As has already been pointed out, since the procedures are built in terms of a representative for each class, we must now consider each and every possibility, that is, all the possible rotations of the wheel, which explains the 90° turns.

The optimal solution can be found in the nondominated list associated with the root of the tree, through a search made using the desired minimization function, be it area, semiperimeter, or any similar objective function. This solution not only defines the best orientation for the basic rectangles but also implicitly defines the optimal sequence of partitions that leads to the overall minimization of the occupied placement area. This chosen solution can then be embedded in the geometry of the board.

The worst-case orders of complexity for the new algorithm are $\mathbb{O}(n(k + Mn))$ for space and $\mathbb{O}(n^4M^3)$ for time. These orders are obtained only when we have a totally unbalanced (degenerate) 5-ary tree, which is, of course, mostly unlikely.

In fact, we have that, for a totally unbalanced tree, each node has four leaf nodes and the last child is either a 5-ary node or is another leaf node. Then, supposing that it has n leaf nodes, at the root of this tree, it must use $S(n) \leq 4S(1) + S(n - 4) + S(C(n))$ space units, where $C(i)$ is the size of the nondominated list for a tree with i leaf nodes. Since $S(1) \leq k$ and $C(n) \leq nM$, substituting above, we have

$$S(n) \leq 4k + S(n - 4) + nM. \tag{2}$$

Recursively applying (2) in a top-down fashion, we get

$$S(n) \leq 4tk + S(n - 4t) + t(n - 4)M + 4M. \tag{3}$$

The recursion stops when $n - 4t = 5 \Leftrightarrow t = (n - 5)/4$, and since $S(5) \leq 5k + 5M$, substituting in (3), we have

$$S(n) \leq (n - 5)k + 5k + 5M + \frac{n - 5}{4}(n - 4)M + 4M$$

$$= nk + \frac{n - 5}{4}(n - 4)M + 9M,$$

which proves that the used space is, in the worst case, $\mathbb{O}(nk + n^2M)$. Similarly, we can prove the stated bounds for time complexity.

As we shall see in the next section, the performance of this algorithm is quite good, especially if we keep in mind that this is an NP-complete problem that arises within VLSI design.

3. EXPERIMENTAL RESULTS

This approach was used to solve essentially two types of instances for this problem. The first one consists of a known

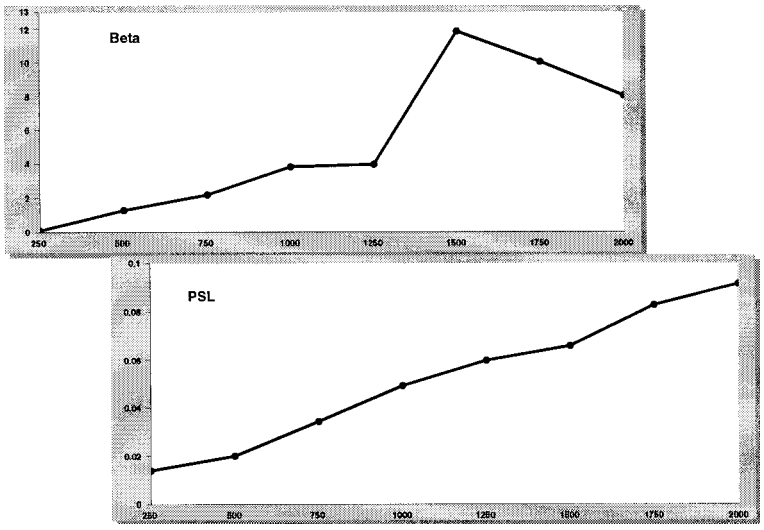

FIG. 8. Time spent on construction of wheels (in seconds).

set of test examples that appear in related literature and are the only ones evaluated in [2]. We then used another set of instances to produce a more systematic and statistical evaluation for *Beta.* This second set of instances was built using several samples of randomly generated floorplan trees. This section therefore begins by presenting a table with direct comparison results between the *Beta* algorithm and the algorithm presented in [2], which we call *PSL.* We then give a graphical presentation of some results obtained using (pseudo)random construction of floorplan 5-ary trees with different dimensions (i.e., number of leaves/components).

All the implementations were coded in C and run on a 433-MHz Digital Beta computer with 64M RAM.

3.1. Comparison Between Beta and PSL

Table 1 shows a direct comparison between the performance of algorithms *Beta* and *PSL.* Due to the artificial nature of these examples, somewhat simpler algorithms were used, since it is not necessary to rotate the wheels to achieve optimality. Therefore, we denote these simpler versions by *Beta0* and *PSL0.*

Since the global optimum was obtained by both versions, we can only compare the number of examined feasible solutions, that is, the wheels constructed and tested for nondominance. It is seen that *Beta0* does, in fact, examine substantially fewer feasible solutions than does *PSL0,* and this fact also has obvious consequences in terms of running times. This dominance is clearest in pure 5-ary trees, that is, trees that are formed exclusively with 5-ary nodes, where the number of examinations made by *Beta0* is, on average, less than half of the number made by *PSL0.*

There seems to be absolutely no doubt that it is the tests done, both for producing a feasible solution and for nondominance, that influence the time complexity levels achieved by these algorithms. Also, of course, this com-

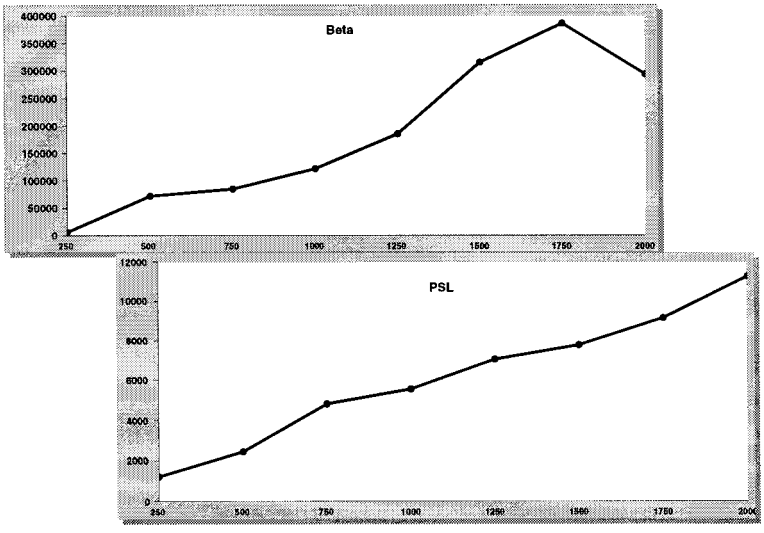

FIG. 9. Total number of feasible wheel solutions examined.

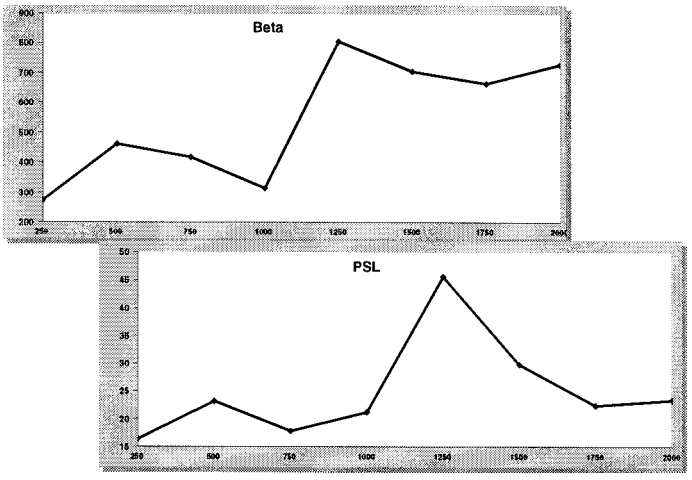

FIG. 10. Number of nondominated solutions at the root of the tree.

plexity is highly dependent on the number of pairs present in the lists associated with the nodes of the tree.

Notice that the results are similar when "mixed" trees are used. In fact, this seems to be the most natural case to occur for real problems. Moreover, the "pure" wheel procedures in [2] do not need any previously decided partitioning sequence. Therefore, we decided to produce a randomly generated set of instances for this kind of tree and use Pan et al. procedures, to have a better indicator for the performance of these algorithms. We were especially interested in evaluating their performance when using the exact algorithm described in [1], that, hereafter, we will call *Opt.*

3.2. Beta Performance

Next, we present the results of the application of both the *Beta* and *PSL* algorithms to floorplan 5-ary trees with different dimensions, that is, different numbers of leaf nodes or components. Here, *PSL* also uses *Opt* for the construction of the nondominated list associated with a binary node. These algorithms include the rotations provision to produce the complete list of nondominated solutions. The trees were generated using a (pseudo)random 5-ary tree generator based on the linear congruential algorithm and 48-bit integer arithmetic. These trees are randomly "mixed," that is, they can use both kinds of nodes (which will be the most probable structure) but can also be pure binary or pure 5-ary.

In the following figures, the x-axis always represents the number of components or leaf nodes for each set of trees. Each of the results, hereafter, is an average of runs on 20 different floorplan trees for each number of components.

The overall time spent in the compaction of placement area is roughly the sum of the time spent on wheel construction plus the time spent on slices. Due to the relative scales involved, the time used for slices is, in fact, insignificant regarding the total running time,[‖] that is, the overall time is directly related to the time spent constructing wheels.

As seen in Figure 8, the time *Beta* now spends building feasible wheel-type solutions is much different from *PSL,* but this time is directly related to the number of constructed and examined solutions, as is clear from examination of Figure 9.

As we can see, *Beta* always constructs far more candidates for nondominance. Moreover, the number of constructed and examined wheels is important since, as was expected, the best result in terms of minimization of area was not achieved by *PSL.* In fact, the number of nondominated solutions in the lists associated with the root of the tree are again substantially different between *Beta* and *PSL.*

As seen in Figure 10, the total number of nondominated solutions for *Beta* is again far greater than is the number obtained with *PSL,* which emphasizes that *PSL* is not able to determine the complete nondominated solution set.

In terms of the minimization of the occupied area, Figure 11 shows a complete domination by *Beta.* Since we have shown that this is an optimal strategy, the *PSL* algorithm can only be considered as a heuristic approach.

Finally, we mentioned previously the upper bound of nM for the number of nondominated solutions in the list associated with the root of the tree. The only sound conclusion is that, in general, this number seems to be independent of n. Indeed, Figure 10 consistently presents integer values far below the number n.

4. CONCLUSIONS

We have presented an overall approach to the minimization of hierarchical floorplans that uses nondominance in

[‖] This time is on the order of milliseconds, regardless of the number of components involved.

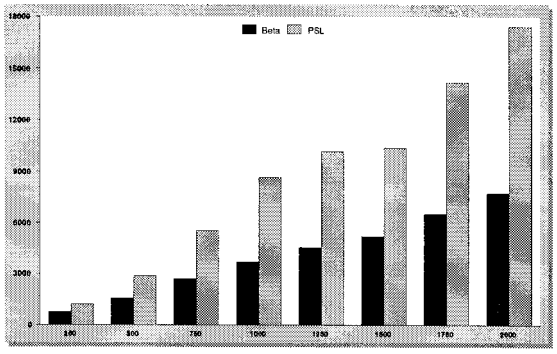

FIG. 11. Area sizes obtained by choosing the best solution at the root of the tree.

all stages of compaction to ensure that the best solution for the optimization of an appropriate nondecreasing function can be found at the root of the floorplan tree. From this theoretical framework, it was possible to derive an algorithm that uses the tree representation without modifying it. This algorithm also solves a more general case than those found in related literature.

A more concise and formal study of the construction of feasible solutions seems to be the appropriate road to follow to develop a greater insight into the structure of these solutions. This may enable a reduction in the number of constructed feasible dominated solutions.

Acknowledgments

The authors would like to thank the referees for the valuable help they gave with their corrections and thank also D. Shier for his constant encouragement. The authors are deeply indebted to the late Prof. Ernesto Q. V. Martins for having pointed out to us the relation between our work and nondominance theory and for all the fruitful talks and discussions.

REFERENCES

[1] A.M. de Almeida, E.Q.V. Martins, and R. Rodrigues, Optimal cutting directions and rectangle orientation algorithm, Eur J Oper Res 109 (1998), 660–671.

[2] P. Pan, W. Shi, and C.L. Liu, Area minimization for hierarchical floorplans, Algorithmica 15 (1996), 550–571.

[3] W. Shi, A fast algorithm for area minimization of slicing floorplans, IEEE Trans Comput-Aided Des Integr Circuits Syst 15 (1996), 1525–1532.

[4] L. Stockmeyer, Optimal orientations of cells in slicing floorplan designs, Info Contr 57 (1983), 91–101.

[5] K.-S. The and D.F. Wong, Graph techniques for hierarchical floorplan area optimization, Technical report TR-93-1, Dept. of Computer Sciences, University of Texas, Austin, 1993.

[6] T.-C. Wang and D.F. Wong, An optimal algorithm for floorplan area optimization, 27th ACM/IEEE Design Automation Conf., ACM Press, New York, 1990, pp. 180–186.