ELSEVIER

European Journal of Operational Research 109 (1998) 660–671

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

Optimal cutting directions and rectangle orientation algorithm [1]

A.M.C. Almeida *, Ernesto Q.V. Martins, Rosália D. Rodrigues

Dep. de Matematica, F.C.T. Universidade de Coimbra, 3000 Coimbra, Portugal

Received 3 October 1995; accepted 18 February 1997

Abstract

The first stage in hierarchical approaches to Floorplan Design defines topological relations between components that intend to optimize a given objective in a circuit board. These relations determine a placement that is subsequently optimized in order to minimize a cost measurement (that will probably be one between chip area or perimeter). The board optimization gives rise to multiple subproblems that need to be answered in order to obtain a good solution. Among the most relevant ones we find the problem of defining the optimal orientation of cells and the definition of the optimal cutting sequence that minimize the placement board area. We will present a generalization of an algorithm due to Stockmeyer so that it obtains a solution that not only defines the optimal cell orientation but also the slicing cuts sequence that will lead to this optimal orientation and overall area minimization. © 1998 Elsevier Science B.V.

Keywords: Packing; Optimization; Compaction; Cutting direction; Rectangle orientation; Nondominated solution

1. Introduction

In most hierarchical approaches to Floorplan Design the first stage produces a particular kind of binary tree as a model for topological relations between the given set of components. This model is subsequently embedded on a board by associating to each tree leaf, representing a single component, Cartesian coordinates that maintain the relations previously found. This is done by recursively using a predefined sequence of slicing cuts (where recursion becomes naturally available due to the hierarchical nature of the tree).

In the next stage, commonly known as Compaction, the board area will be minimized, which can be done by way of rotating, swapping or taking the mirror image of components.

In general, the problem of finding an orientation of the components that optimizes the placement area belongs to the class of NP-complete problems [4,6]. However, if the components are rectangular shaped and the layouts to be obtained are of the so-called *slicing* type, this optimization problem can be solved in polynomial time [6]. Under these restrictions, the binary tree that stands for the topological model of the circuit can also represent a sequence of slicing cuts, which is needed to define rectangular regions in order to place the individual components.

In 1983, Stockmeyer presented a polynomial time algorithm which, using this binary tree, solves the orientation problem arising in Slicing Floorplan Designs, *provided the sequence of slicing cuts is given* [6]. The predefined sequence of cuts is used to label each nonleaf node so that it specifies a horizontal or

* Corresponding author. E-mail: amca@pantera.mat.uc.pt

[1] This work was partially supported by the Centro de Informatica e Sistemas da Universidade de Coimbra.

PII S0377-2217(97)00085-4

vertical cut and only then Stockmeyer's optimization takes place.

In 1990, a work due to Wang and Wong [7] presented an extension of the previous one, where Stockmeyer's algorithm is extended to solve a slightly more general problem, involving the use of a special kind of floorplans, *hierarchical floorplans of order 5,* that have nonslicing subfloorplans with five modules called *wheels.* In order to optimize the area occupied by wheels, they reduce the corresponding hierarchical representation to a binary tree and apply to this tree special procedures to handle its compaction (and which can be extended for wheels of higher order), while using the algorithm of Stockmeyer for normal slicing floorplans.

The slicing cuts determination is, in itself, a very complex problem, for which, until now, there was no known algorithm that would provide an optimal solution. It can be easily inferred that the problem of finding the optimal rectangle orientation and the problem of finding the optimal cutting directions are related, for the optimal rectangle orientation is not the same for different sequences of cuts and vice versa. In the presence of a polynomial algorithm that obtains the optimal solution for one of the problems, the next obvious step is to let this optimum define the cutting sequence that best serves its purpose.

The problem can be informally described as:

From a set of rectangular shaped components, which are to be placed in a rectangular board in a specified arrangement, what is the best orientation for each component – as well as the directions of the corresponding slicing cuts – so that the placement area is minimized?

In this paper, we present the mathematical support for the algorithmic resolution of this problem, that generalizes Stockmeyer's algorithm and so we show that not only the minimal solution in terms of component orientation but also the optimal slicing sequence can be determined.

2. Definitions

The following definitions are in accordance with most of the ones used by Stockmeyer and so a *floorplan* is an enclosing rectangle subdivided, by horizontal and vertical line segments, into nonoverlapping rectangles (see Fig. 1).

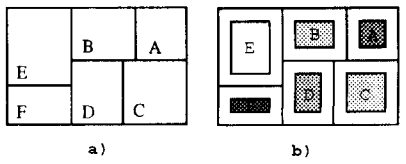

Fig. 1. Slicing floorplan: (a) without components; (b) with components.

A rectangle without internal line segments will be called a *basic rectangle* and will be, from now on, identified with the rectangular component to be placed in it.

A pair of real values (h, w) is associated with each rectangle standing for its side dimensions (height and width).

A *slicing tree* is a rooted binary tree that describes the topological relations between components and whose nodes can be labeled in order to specify the sequence of cuts that will produce the placement regions (basic rectangles) for each component. Each leaf node represents a basic rectangle and each nonleaf node, corresponding to a set of components, stands for an enclosing rectangle (floorplan), which of course implies that the root is identified with all of the placement area (see Fig. 2).

As stated by Stockmeyer, the requirement for binary trees causes no loss of generality since a nonbinary tree can be easily replaced by a binary one, thus simplifying the algorithmic approach to the problem.

The problem to be dealt with can be stated as:

Let $\mathcal{F}(T)$ be the set of real pairs that can stand for feasible dimensions of a floorplan for a given slicing tree T. Given a function $\varphi: \mathbb{R}^2 \to \mathbb{R}$, we

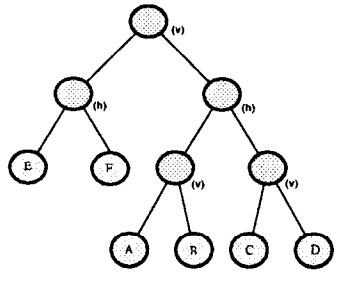

Fig. 2. Slicing tree. Labels: (v): vertical cut; (h): horizontal cut.

intend to minimize $\varphi(h,w)$ over all possible orientations of the components and floorplans associated with T, that is, minimize $\varphi(h,w)$ such that $(h,w) \in \mathcal{F}(T)$.

For this kind of problem, and throughout the related literature, we find that the most commonly used objective functions in this type of problem are the *area*,

$$\varphi(h,w) = hw,$$

or the *semi-perimeter*,

$$\varphi(h,w) = h^2 + w^2.$$

A solution for a basic rectangle is identified with the associated pair (h,w). It corresponds to one of the two possible orientations of the rectangle (if $h > w$, the rectangle is said to be 'up', and 'down' if $h < w$). In case a floorplan is to be considered, the corresponding solution comprehends not only the side dimensions but also all the basic rectangles orientations enclosed within this floorplan. We will use the words *general orientation* or *solution* in case of floorplans and simply *orientation* if we are referring to basic rectangles. $\mathcal{F}(T)$ is therefore the set of all possible solutions for T, or orientations if T is a leaf node.

Finally, a function, $\varphi: \mathcal{D} \subset \mathbb{R}^2 \to \mathbb{R}$ will be called *nondecreasing in both arguments* (or just *nondecreasing*) if

$$(a < c \wedge b \leq d) \vee (a \leq c \wedge b < d) \Rightarrow \varphi(a,b) \leq \varphi(c,d) \quad \forall (a,b),(c,d) \in \mathbb{D}.$$

3. Stockmeyer's algorithm for general slicing floorplan orientation

A brief description of the algorithm found in [6] will now be presented so as to proceed with the presentation of our algorithm.

Given a slicing tree T, where each nonleaf node is labelled by way of specifying a vertical or an horizontal cut, and a nondecreasing function $\varphi(h,w)$ on $\mathcal{F}(T)$, the algorithm begins by associating to each leaf node in T a list that represents no more than the two possible orientations of the cell to be placed in the corresponding basic rectangle. Not to forget that the two pairs in the list must be ordered according to the second item in the table of properties presented next (if the corresponding component has square shape, then there will be only one possible orientation and therefore, only one possible pair). It then proceeds by associating with each nonleaf node v a list of s pairs,

$$\{(h_1,w_1),\ldots,(h_s,w_s)\},$$

satisfying the following properties, where $L(v)$ stands for the set of leaves of the subtree rooted at v:

1. $s \leq |L(v)| + 1$.
2. $h_i > h_{i+1}$ and $w_i < w_{i+1}$, $1 \leq i < s$.
3. For each of these pairs there is a possible orientation, ρ, of $L(v)$ in terms of the defined slicing cuts.
4. For each orientation ρ of $L(v)$ (in terms of the given slicing sequence) there is a pair (h_i,w_i) in the list with $h_i \leq h(\rho)$ and $w_i \leq w(\rho)$.

Let

$$\{(h_1,w_1),\ldots,(h_m,w_m)\} \text{ and } \{(h'_1,w'_1),\ldots,(h'_k,w'_k)\}$$

be the sorted (according to item 2) lists associated with the two children of v. The list associated with v is then constructed by using the following version of Stockmeyer's procedure, which is described in terms of a vertical cut and that can be easily changed for a horizontal cut.

Stockmeyer's Procedure.

- Initialize $i \leftarrow 1$, $j \leftarrow 1$.
- **while** $i \leq m$ and $j \leq k$:
  - Add $(\max\{h_i, h'_j\}, w_i + w'_j)$ to the list of v with pointers to (h_i,w_i) and (h'_j,w'_j).
  - **If** $h_i > h'_j$, **then** $i \leftarrow i + 1$;
  - **Else** if $h_i < h'_j$, **then** $j \leftarrow j + 1$;
  - **Otherwise** $i \leftarrow i + 1$ and $j \leftarrow j + 1$.

With this procedure, and under the previously stated restrictions, each list comprehends only pairs (corresponding to possible solutions) such that there is no other one that is strictly better in both dimensions and also no other one strictly better in one dimension and no worse with respect to the other, since this type of pair would correspond to suboptimal orientations and can therefore be eliminated.

By using a bottom-up approach, the previous algorithm constructs the lists of pairs (possible solu-

tions) for all the nonleaf nodes in T until the root of the tree is reached. Since the objective function is nondecreasing, the minimization of φ over all possible orientations ρ for a given slicing sequence can then be obtained by minimizing φ over all pairs (h_i, w_i) present in the list constructed for the root of T. This list is thus subsequently searched for the pair that minimizes the given cost function φ.

The running time and storage requirements are both $\mathcal{O}(dn)$, where n is the number of leaves and d the depth of the tree.

4. Optimal cutting direction and orientation algorithm

This section begins by pointing out that each of the pairs in a list constructed for a node v by Stockmeyer's algorithm is no more than a nondominated solution for the minimization of $\varphi(h(\rho), w(\rho))$ over all orientations ρ for the floorplan represented by v.

In fact, define a *dominated* solution for this problem as one in the set

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

written as $(c, d)D(a, b)$ and read as (a, b) *is dominated by* (c, d).

A *nondominated* solution will then be one in the set $\mathcal{N}(\mathcal{T})$, where

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

Minimizing φ over all possible orientations ρ is equivalent to minimizing φ over all nondominated solutions and it is not difficult to prove that the optimal solution lies in the set $\mathcal{N}(\mathcal{T})$.

Naturally, the optimal solution thus obtained is only *optimal in terms of a given slicing sequence* and thus only locally optimal. As previously stated, the problem of finding this optimal sequence is in itself a very complex problem. Usually, the sequence of cuts is defined beginning with virtual dimensions for the enclosing rectangle associated with the root of the tree (which, of course, must be big enough to accommodate all the components) and recursively define the sequence whether by specifying that the cut should occur in the direction of the greatest dimension, or alternating the cutting direction in each level of the tree, or even generating a random sequence of cuts.

Since both problems are interrelated and Stockmeyer's algorithm is, in practice, very fast, what would happen if the cutting direction is not specified?

In the following pages it will be shown that the optimal overall solution for the minimization of placement area is still a nondominated one. Beginning with a necessary lemma that will be used to show the correctness of the previous statement, we will then prove that, again, we only need nondominated solutions in each tree level to achieve the entire $\mathcal{N}(\mathcal{T})$ set at upper levels.

Lemma 1. *Any dominated solution is dominated by at least one nondominated solution, i.e.,*

$$(a,b) \in \mathcal{D}(\mathcal{T}) \Rightarrow \exists (a',b') \in \mathcal{N}(\mathcal{T}) : (a',b')\, D(a,b).$$

Proof. In fact, let us assume that $(a, b) \in \mathcal{D}(\mathcal{T})$ is dominated by $(a_1, b_1) \in \mathcal{D}(\mathcal{T})$. Since (a_1, b_1) is also a dominated solution, there exists some solution (a_2, b_2) that dominates it. If $(a_2, b_2) \in \mathcal{D}(\mathcal{T})$, with analogous reasoning application we will end up with a sequence

$$(a_n,b_n)\, D \cdots D(a_2,b_2)\, D(a_1,b_1)\, D(a,b).$$

As $\mathcal{F}(T)$ is a finite set, n must be finite and thereafter there cannot be any other solution that dominates (a_n, b_n), that is $(a_n, b_n) \in \mathcal{N}(\mathcal{T})$. Since D is a transitive relation (which easily follows by the definition of a dominated solution) we have $(a_n, b_n)D(a, b)$, which proves that there is at least one element of $\mathcal{N}(\mathcal{T})$ that dominates (a, b). □

The next result proves that, in fact, the optimal solution for our problem is nothing more than a nondominated pair (h, w) in $\mathcal{F}(T)$.

Theorem 1. *Given a slicing tree T, the optimal solution to the problem of minimizing $\varphi(h_{F,\rho}, w_{F,\rho})$ over all possible orientations of components, $\rho \equiv \rho(T)$, and floorplans, $\mathcal{F} \equiv \mathcal{F}(T)$, lies in the set $\mathcal{N}(\mathcal{T})$.*

Proof. Let $(a^*, b^*) \in \mathscr{F}(T)$ be the optimal solution for the minimization of φ. Then

$$\varphi(a^*,b^*) \le \varphi(\bar{a},\bar{b}) \quad \forall(\bar{a},\bar{b}) \in \mathscr{F}(T). \tag{1}$$

Let us assume that $(a^*, b^*) \mathscr{D}(\mathscr{T})$. If (a^*, b^*) is a dominated solution and using Lemma 1, there must be at least one element of $\mathscr{N}(\mathscr{T})$ that dominates this pair, that is,

$$\exists(a,b) \in \mathscr{N}(\mathscr{T}): (a \le a^* \wedge b < b^*) \vee (a < a^* \vee b \le b^*).$$

But as φ is nondecreasing in both arguments, this implies that

$$\varphi(a,b) \le \varphi(a^*,b^*). \tag{2}$$

Eqs. (1) and (2) imply $\varphi(a^*,b^*) = \varphi(a,b)$, which also means that

$$\varphi(a,b) \le \varphi(\bar{a},\bar{b}) \quad \forall(\bar{a},\bar{b}) \in \mathscr{F}(T).$$

Therefore, there is at least one nondominated solution that is optimal for the minimization of φ.

As $\mathscr{F}(T) = \mathscr{D}(\mathscr{T}) \cup \mathscr{N}(\mathscr{T})$, all has been proven. □

We now need a result that can lead to the expedite construction of the desired nondominated set. The following theorem proves that, regardless of the level of the tree, we only need nondominated solutions to obtain, at the root of the tree, the set $\mathscr{N}(\mathscr{T})$.

Theorem 2. *If $(a, b) \in \mathscr{N}(\mathscr{T})$ and $(a', b') \in \mathscr{D}(\mathscr{T})$, then the solution that encloses both the previous ones in a floorplan is either a dominated one or can be obtained by using only nondominated solutions, regardless of the cutting direction involved.*

Proof. Let $(a, b) \in \mathscr{N}(\mathscr{T})$ and $(a', b') \in \mathscr{D}(\mathscr{T})$ be the solutions which we want to enclose in a new rectangle. Since (a', b') is dominated there is $(c, d) \in \mathscr{N}(\mathscr{T})$ such that

$$(c \le a' \wedge d < b') \vee (c < a' \wedge d \le b'). \tag{3}$$

Using a vertical cut the resulting solution would be $(\max\{a,a'\}, b + b')$.

First of all notice that (3) implies that one of the following inequalities holds:

$$b + b' > b + d, \tag{4}$$

$$b + b' = b + d. \tag{5}$$

(i) If $a' \le a$, then $\max\{a,a'\} = a$.

(i.1) If inequality (4) holds, using (3) we have $c \le a'$. But then $c \le a' < a$ which means that also $\max\{a,c\} = a$. The following relation will then hold:

$$\max\{a,c\} = \max\{a,a'\} \wedge b + d < b + b',$$

and therefore

$$(\max\{a,c\}, b + d)\, D(\max\{a,a'\}, b + b'),$$

which proves that the new solution is dominated.

(i.2) If (5) holds, then together with (3) it yields $c < a'$. Therefore $c < a' \le a$, meaning that in both pairs the maximum is again the same and this implies that

$$(\max\{a,c\}, b + d) = (\max\{a,a'\}, b + b'),$$

which means that we can obtain the same pair only using nondominated solutions.

(ii) If $a < a'$, then $\max\{a,a'\} = a'$.

(ii.1) If inequality (4) holds, then either $c < a < a'$ or $a < c < a'$ hold. In both cases we have

$$\max\{a,c\} \le \max\{a,a'\} \wedge b + d < b + b',$$

and therefore

$$(\max\{a,c\}, b + d)\, D(\max\{a,a'\}, b + b').$$

(ii.2) If (5) holds, we have that either $a \le c < a'$ or $c \le a < a'$ is true and this will always mean that

$$\max\{a,c\} < \max\{a,a'\} \wedge b + d = b + b',$$

and again we prove that the new solution is a dominated one with which we conclude the proof for a vertical cut.

Similarly it can be shown that the same result holds if an horizontal cut is used. □

According to the previous theorems, to obtain the optimal solution we need to construct the entire set $\mathscr{N}(\mathscr{T})$ and, for that, we need only the set of nondominated solutions at each node of the tree. This leads to the following strategy: for a given node v of $\mathscr{T}$,

- Construct the list of nondominated solutions for v, assuming that a vertical cut should take place (we can of course use Stockmeyer's module for vertical cuts).
- Next, the list corresponding to the possible existence of an horizontal cut is constructed (and, again, Stockmeyer's module for horizontal cuts can be used).

- Finally, the two lists are *merged in order, deleting every dominated solution* that might have occurred with the merge.

A procedure that constructs the list of all nondominated solutions with no predefined sequence of cutting directions could then be:

New Procedure.

- Initialize $i \leftarrow 1$, $j \leftarrow 1$.
- **while** $i \leq m$ and $j \leq k$:
  - Add $(\max\{h_i, h'_j\}, w_i + w'_j)$ to the list of v with pointers to (h_i, w_i) and (h'_j, w'_j).
  - **If** $h_i > h'_j$, **then** $i \leftarrow i + 1$;
  - **Else if** $h_i < h'_j$, **then** $j \leftarrow j + 1$;
  - **Otherwise** $i \leftarrow i + 1$ and $j \leftarrow j + 1$.
- Initialize $i \leftarrow m$, $j \leftarrow k$.
- **while** $i \geq 1$ and $j \geq 1$:
  - *Actualize* $(h_i + h'_j, \max\{w_i, w'_j\})$ to the list of v with pointers to (h_i, w_i) and (h'_j, w'_j).
  - **If** $w_i > w'_j$, **then** $i \leftarrow i - 1$;
  - **Else if** $w_i < w'_j$, **then** $j \leftarrow j - 1$;
  - **Otherwise** $i \leftarrow i - 1$ and $j \leftarrow j - 1$.

Note that in the second loop, the operation *Actualize* produces a slight but most important alteration to the corresponding Stockmeyer module for horizontal cuts: it first tests any recently constructed solution to find out if it constitutes, in fact, a nondominated solution with respect to the solutions already in the list. Not until it is known that it still is nondominated does it join the list (in accordance with the order proposed by Stockmeyer), thus avoiding merging both lists and preventing the insertion of dominated solutions. As this test for nondominance is really a test about ordering, it works both ways, which also means that 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 in the list.

Having obtained the set of all nondominated solutions, for all possible cutting sequences, now present in the list associated with the root of the tree, the following step will naturally be the search for the minimal solution for the general floorplan. This optimal solution not only defines the best orientation of the basic rectangles but *also implicitly defines the needed sequence of cuts* that leads to the minimization of the occupied area of placement. This solution can then be embedded in the geometry of the board using a top-down procedure.

As stated above, due to the fact that there is no predefined sequence of slicing cuts, all the nondominated solution set for both cutting directions must be obtained to attain aimed optimization. This will, evidently, cause an increase in the number of possible solutions, when compared with Stockmeyer's algorithm, and this growth is studied next.

Theorem 3. *In the worst case, for the root of a balanced tree with n leaves, there can be an* $\mathcal{O}(n^2)$ *number of nondominated solutions, and for degenerated trees* $\mathcal{O}(2^n)$.

Proof. In fact, the number, $C(n)$, of possible pairs in the list of the root of a balanced tree with n leaves (and therefore with depth $\log_2 n$) will be at most

$$C(n) \leq 2\left(2C\left(\tfrac{1}{2}n\right) - 1\right) = 2^2 C\left(\tfrac{1}{2}n\right) - 2. \qquad (6)$$

For any kind of binary tree, the number of possible pairs (solutions) associated with a nonleaf node v with two leaf-node children, $C(2)$, is at most

$$C(2) \leq 2(|L(v)| + 1) = 2^2 + 2 = 6,$$

since it will correspond to executing Stockmeyer's algorithm twice (see Fig. 3).

If formula (3) is applied recursively, we will get

$$C(n) \leq 2^{2k} C\left(\frac{n}{2^k}\right) - \sum_{i=1}^{k} 2^{2i-1}$$

$$= 2^{2k} C\left(\frac{n}{2^k}\right) + \frac{2(1 - 2^{2k})}{3}.$$

The final level of recursion will occur when $n/2^k = 2$, that is, $k = \log_2 n - 1$. Substituting k in the previous formula we get

$$C(n) \leq \tfrac{1}{3}(4n^2 + 2),$$

which means that the number of nondominated solutions at the root of the tree will be $\mathcal{O}(n^2)$.

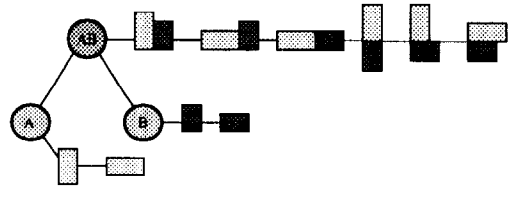

Fig. 3.

For degenerated trees with n leaves (depth n) we have

$$C_n \leq 2(2 + C(n-2) - 1) = 2C(n-2) + 2. \quad (7)$$

Again, by recursive application of formula (2) we get

$$C(n) \leq 2^k C(n-2k) + \sum_{i=1}^{k} 2^i$$

$$= 2^k C(n-2k) + 2^{k+1} - 2,$$

until we reach $n - 2k = 2$, that is, $k = \frac{1}{2}n - 1$ which, when k is substituted, gives

$$C(n) \leq 2^{n/2+2} - 2$$

i.e. the number of pairs in the list of the root is $\mathcal{O}(2^n)$. □

As for running time, by an analogous proof, the worst case is also $\mathcal{O}(n^2)$ for balanced trees and $\mathcal{O}(2^n)$ for degenerated trees. This only stresses the fact that one should choose balanced trees to be used in this kind of problem, which was already referred to by Stockmeyer, amongst other authors.

It should also be pointed out that, with this procedure, we will tend to include, in the list of solutions associated with a given node, pairs that are no more than the rotation of another already in the list (rotation in terms of the corresponding rectangles).

Theorem 4. *If* $(a, b) \in \mathcal{N}(\mathcal{T})$, *then* $(b, a) \in \mathcal{N}(\mathcal{T})$.

Proof. In fact, given a nondominated pair (a, b), suppose that (b, a) is dominated, that is, $(b, a) \notin \mathcal{N}(\mathcal{T})$. This will mean that there is a pair (c, d) such that

$$(c \leq b \wedge d < a) \vee (c < b \wedge d \leq a).$$

But then, there is a pair (d, c) that dominates (a, b) and therefore $(a, b) \notin \mathcal{N}(\mathcal{T})$, which is absurd. □

One way to prevent such an increase in the number of items in the lists would be to include only one of these pairs and therefore each pair in the final list would represent not one but two possible solutions, one of which would be obtained by exchanging the corresponding dimensions (thus reducing the number of pairs in the lists by approximately half of the expected). This fact must be accounted for when building the various lists of solutions (so that no nondominated solution is lost, which will naturally imply a small growth in running time). In order to accomplish this, the list associated with a node v in the tree will be built after four different traversals of the lists of its children nodes v_1 and v_2:

1. one done just like the one corresponding to a vertical cut in Stockmeyer's algorithm;
2. another one as in the referred vertical cut but rotating the pairs in the lists associated with v_1 and v_2;
3. a new traversal rotating only the pairs in the list of v_1; and finally
4. one rotating only the pairs in the list v_2.

Note that each new pair is only included in the list of v if and only if it represents a nondominated solution with respect to the pairs already included in the list and again this can be done by using the subprocedure *Actualize* already referred to.

5. Computational experiments: instance application

We applied both Stockmeyer's algorithm and the new optimal algorithm in one of the resolution stages of a broader problem where a rectangular area must be compacted so as to minimize a cost measure in order to accomplish a given objective.

This problem arises within VLSI Circuit Design where many of the problems defined are very complex, being classified as NP-complete or NP-hard. Therefore the design is divided into various phases where it is possible to define simpler problems. The general phases of design that are of interest within this paper are the first one, called *Placement*, and the one where the area occupied by the placement of the components of the circuit is minimized, *Compaction*. Given a structural model of a circuit, specifying the number and shapes of the components and all their interconnections, we must place the components optimizing not only the total wire length but also the occupied area of placement as a means to

provide a highly routable circuit for one of the next phases of the design (*Routing*).

The cost function used here is a linearization of a multiobjective function on total wire length and final placement area, which are believed to be the most important issues to provide a good routable solution [5]:

$$\min F = A + \lambda W, \tag{8}$$

where W stands for the total wire length and A for final placement area.

The parameter λ is used to increase or decrease the relative importance given to the function W and it will be assumed for the purpose of this paper that $\lambda = 1$.

The general philosophy of a Placement System can be summarized in the following manner:

The structural model for the circuit is a graph, where each node represents a component and each edge, with an associated weight and connecting two different nodes in the graph, represents some measurement of the connections that involve the corresponding components. Using Graph Partition we obtain a topological model of the circuit, a slicing tree, that defines 'neighborhood' relations between components and that optimizes the total wire length. This topological model will then be embedded in the geometry of the placement board.

The cost function normally used to optimize the wire length is

$$W = \sum_i \sum_j c_{ij} \mathrm{dist}\big((x_i, y_i), (x_j, y_j)\big), \tag{9}$$

where i and j are components or, which is the same, the nodes that represent them in the graph, c_{ij} is the weight associated with edge $\{i, j\}$ connecting nodes i and j, and the remaining function, dist, is the distance between the geometrical centers of the locations where the components will be placed.

As for area, the cost measures most used are the *area*, $A(h,w) = hw$, or the *semi-perimeter*, $A(h,w) = h^2 + w^2$, where h stands for the total height and w for the total width of the occupied placement area.

In [1] it is proved that the minimization of (9) is an NP-complete problem and so heuristics must be used to approximate the optimal solution. An heuristic, first presented by Kernighan and Lin in [3], that uses a top-down iterative procedure based on graph bipartition to built the slicing tree was implemented. Due to the fact that it must start with a first feasible solution we have implemented the *3-OPT* version of this heuristic with two different initial solutions:

- KL, where a randomly generated first feasible solution is used;
- Mis, using a bottom-up clustering constructive algorithm that can be found in [2] to obtain the first solution.

These two approaches have been used in order to compare the compaction algorithms not only for one but also for two different approximations for the final solution of the placement of a given circuit. In fact, the slicing trees obtained with each of these versions are different from one another which means that the placement of a given circuit solved using KL will be different from the one obtained using the Mis version.

Due to restrictions in available memory space, the new algorithm was implemented in a slightly different way, which will only affect the execution time (in the sense that it will be worse than expected). It was implemented in C code, using the last version presented (with the four traversals of the lists associated with the nodes of the tree). We will not keep all the lists in memory simultaneously, which means that we must reconstruct the lists when needed. These modifications cause a growth in the executed time whilst saving memory space allocations during execution.

When implementing Stockmeyer's algorithm we had to decide how the needed sequence of slicing cuts should be given. After various tests using the three possible approaches:

- at each node of the tree the direction of the slice is randomly generated;
- starting with virtual dimensions big enough to accommodate all the placement, always cut in the direction of the higher dimension; and
- starting with the same conditions as in the previous approach, the first cut (associated with the root of the tree) is in the direction of the biggest dimension and the directions of the subsequent cuts are alternated;

we observed that the third approach always turned out to be the best of all in the sense that it is the one that obtained best results for the cost measures used (*Area* or *Semi-Perimeter*) and it was therefore the

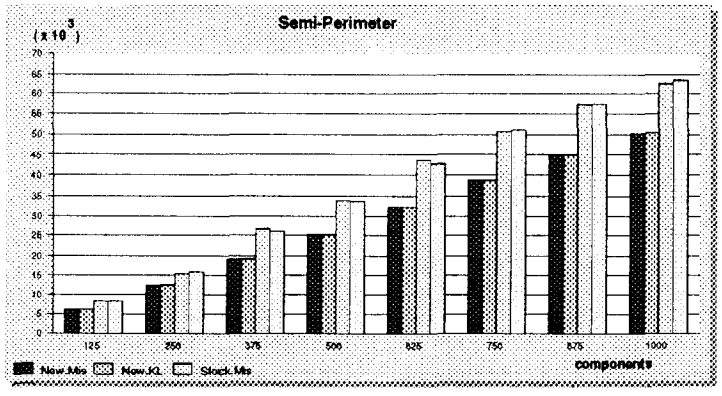

Fig. 4. Semi-Perimeter: $\varphi(h,w) = h^2 + w^2$.

one that we chose to use for comparison with our new algorithm.

6. Computational results

All the necessary implementations were done using C code in a DECSYSTEM 3100 with 16 Mb of RAM and a computational speed of 14 MIPS. Each of the presented results is the average of 10 different problems (circuits) with the same number of components but with different specifications randomly generated.

Looking at the first diagram we can compare the performance, in terms of minimization of the area occupied with the placement, for both versions, KL and Mis, as well as for both algorithms, Stockmeyer's and New.

The compaction algorithms were used with *Semi-Perimeter*, $\varphi(h,w) = h^2 + w^2$ (where h is the height and w the width of the floorplan enclosing all the placement), as the objective function to be minimized.

Fig. 4 shows that, regardless of the number of components involved and the versions used to built the tree, the new algorithm does in fact minimize the semi-perimeter. Moreover it can now be shown that Stockmeyer's approach to the problem did provide a good approximation for the minimization of the occupied area in terms of this evaluation function.

In Fig. 5, occupation ratios (all of the occupation area vs. space physically used) are shown, again for both versions as well as for both algorithms. The theoretical optimum should be a ratio occupation equal to 1. This would mean that all the available area was in fact occupied with components. This is

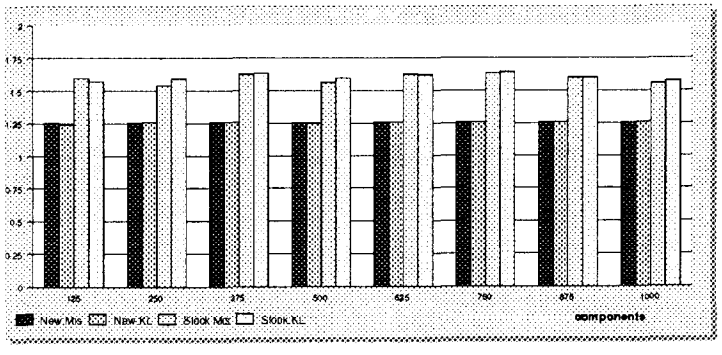

Fig. 5. Area occupation ratios.

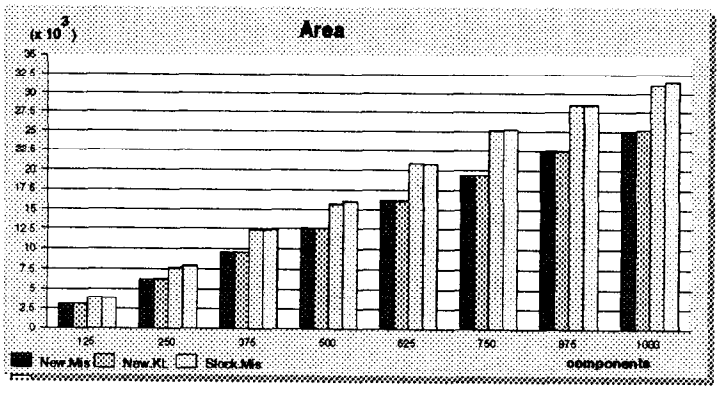

Fig. 6. Area: $\varphi(h,w) = hw$.

however unpractical since in general components are not all of the same size and shape. Again it can be seen that, while with Stockmeyer's algorithm roughly half of the area that can be occupied is wasted, the new algorithm reduces this wastage by more than one half.

Fig. 6 is perfectly analogous to Fig. 4 save that this diagram shows results in terms of the cost measure *Area* and was only presented for the sake of possible readers interested in seeing results using this cost measurement.

In terms of the objective function (8), that is, the evaluation of the final placement, where both the occupied area and the total wire length are expected to be optimized so that a good routing can be performed on the placed circuit, results are presented in the next diagram (Fig. 7).

Again it can be seen that our algorithm performs best, not only because it reduces the occupied area but also because, in doing this, it reduces the value of the function W (since, if the components are closer to each other, then the length of wire used to connect them will also decrease).

Within this context the execution times cannot be fairly compared since the implementation of the new algorithm was considerably altered from the original procedure. In fact, while even for the largest circuits, with 1000 components, Stockmeyer's algorithm did take less than one second on average to compute the minimum (which means that it took 'all' this time to

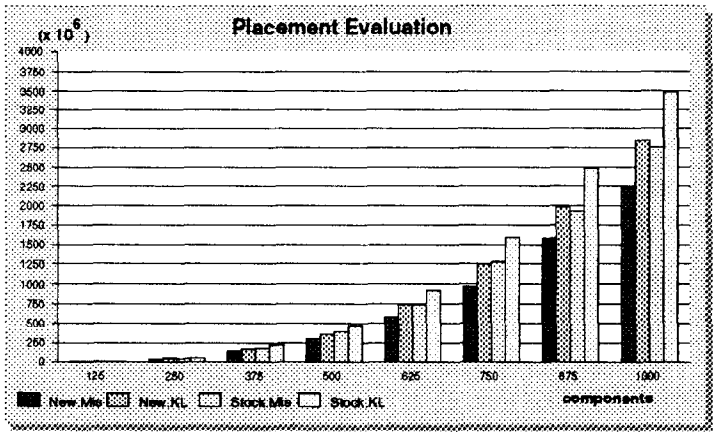

Fig. 7. Total evaluation of placement: $F = A + W$.

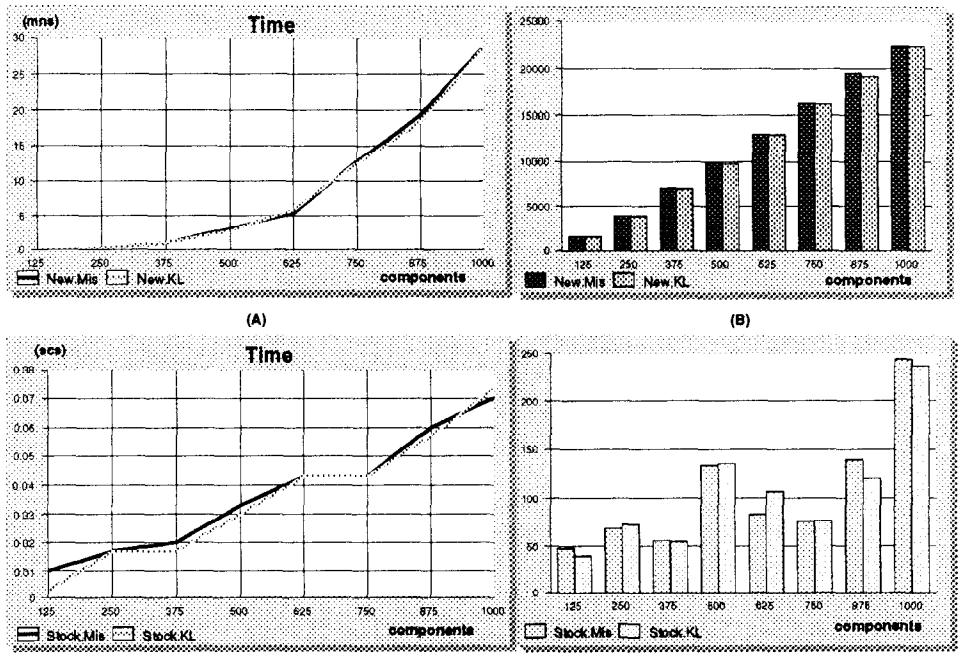

Fig. 8. (A) Execution times; (B) number of solutions present at the root of the trees.

construct the lists associated with the nodes of the tree and then search for the best solution, in terms of the cost measure involved, in the root list), under the restrictions already mentioned, our algorithm took approximately half an hour to do the same. However, when the procedure presented in Section 4 is literally implemented it runs quite fast, presenting average

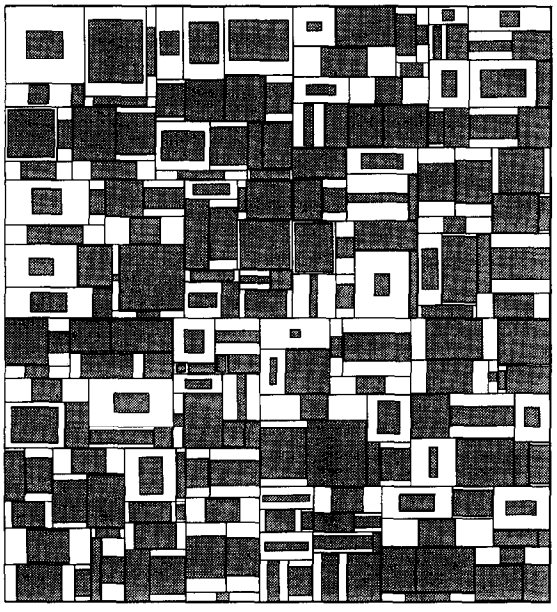

Fig. 9. Best placement obtained with Stockmeyer's heuristic.

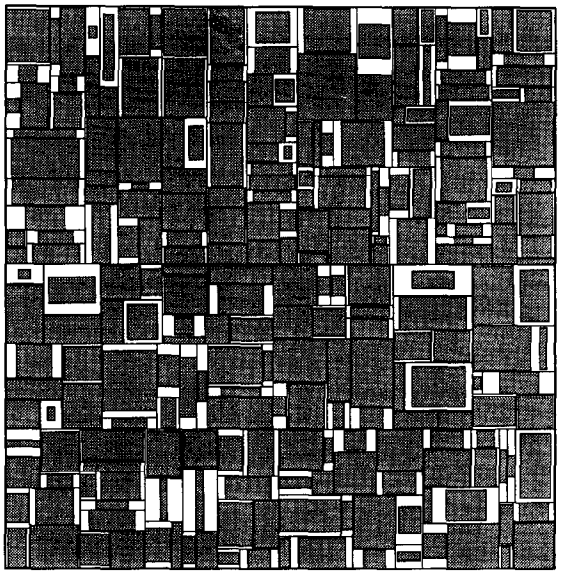

Fig. 10. Placement obtained with our optimal algorithm.

execution times close to linear in the number of leaf nodes.

When we compare the graphics in item (B) of Fig. 8, it is easily seen that Stockmeyer's procedure produces much less feasible solutions. In fact, being no more than an heuristic for the problem we intend to solve, it easily gets stuck in local optima. Depending on the tree at hand and how soon it gets stuck, the lengths of the lists at the roots vary immensely and have no relation with the number of components included in the problem. As for the new algorithm, the number of possible solutions (that certainly include the global optima) are not comparable with the previous ones and are strongly dependent on the components involved in the problem (that is, they depend not only on the number of components but also in their shapes and sizes and how they cluster to produce new solutions). The New algorithm is therefore much slower than Stockmeyer's heuristic. It is however well known that time is not pressing within these areas of research for if we are able to design a good circuit, it does not matter if we take a week or a month to do it as long as it is manufacturable (which will be done hopefully for many a month).

We end this section with an example where we show placements obtained with both Stockmeyer's approach (Fig. 9) and our new algorithm (Fig. 10) for one of the circuits with 250 components used for performance evaluation using the Mis heuristic for construction of the tree. The differences in terms of occupation of the available area are visible and corroborate the conclusions drawn from Fig. 5 (area occupation ratios).

7. Concluding remarks

Although the general compaction problem is classified as NP-complete, the strategy of dividing it into smaller problems has once again brought out a particular subproblem, *optimal orientations and cutting directions for slicing floorplans*, that has here been proved to be solvable in polynomial time.

This work also stresses the need to use binary trees within this framework since the orders of complexity that arise when using other higher order trees tend to be exponential, which is also referred to in [7] amongst others.

The use of this compaction algorithm in bottom-up constructive approaches to obtain the slicing tree seems most appropriate and it constitutes a good field for future research.

Acknowledgements

We do not wish to end this paper without a special acknowledgement to one of the referees for his helpful comments and hints.

References

[1] A.M.C. Almeida, Uma abordagem particular ao estudo de alguns problemas NP-completos, PAPCC Dissertation in Mathematics/Computer Science, Dep. de Matematica FCTUC, Universidade de Coimbra, March 1994.

[2] A.M.C. Almeida, M.R.D. Rodrigues, A new clustering approach to hierarchical layout, presented at EURO XI – 11th European Congress on Operational Research, Aachen, Germany, July 1991.

[3] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Technical Journal 49 (1970) 291–307.

[4] M.R., Garey, D.S. Johnson, L. Stockmeyer, Some simplified NP-complete problems, in: Proceedings of the 6th Annual ACM Symposium of the Theory of Computing, Seattle, April 1974, pp. 47–63.

[5] M.R.D. Rodrigues, A tree-based algorithm for component placement, Ph.D. Thesis, Department of Computer Science, University of Manchester, 1986.

[6] L. Stockmeyer, Optimal orientations of cells in slicing floorplan designs, Information and Control 57 (1983) 91–101.

[7] T. Wang, D.F. Wong, Optimal floorplan area optimization, IEEE Transactions on Computer-Aided Design II (8) (1992) 992–1002.

[8] M. Zeleny, Linear Multiobjective Programming, Lecture Notes in Economics and Mathematical Systems, vol. 95, Springer-Verlag, Berlin, 1974.