Rectangle Packing in Polynomial Time

Gunter Bär - University of Greifswald
Claudia Iturriaga - University of Ottawa

June 17, 1993

Abstract

A variation on the problem of packing rectangles in a rectangle with minimum area is the following formulation: Given a set of rectangles with fixed area with their dimensions varying within a given range, pack them in a minimum area rectangle. This problem has been shown to be NP-complete. We study a special case of this that consists in packing the rectangles in a single column, in which the rectangles are stacked. In this paper we study the following problem: Given a set $\mathcal{F}$ of n elastic rectangles, pack them in a rectangular column R of minimum area. The elements of $\mathcal{F}$ are stacked in R. We present a $O(n \log n)$ algorithm to solve this problem.

1 Introduction

One of the problems that has been studied in VLSI ([GJT80, Sto83, Ott83, WKC88, Hak88, DCL89, WKC89, VTs90]) is that of packing a set of rectangles $\mathcal{F}$ into a rectangle R of minimum area. We shall refer to R as the *packing rectangle*. Usually the rectangles represent circuits, and the packing rectangle represents a chip.

Several variations to the rectangle packing problem have also been studied, i.e. The two-dimensional packing [GJT80] and Floorplan [Sto83]. Both problems have been shown to be NP-complete.

Another variation to this problem is known as " Aspect Ratios" deals with packings of sets of *elastic rectangles*. An elastic rectangle R_i is one in which the length l_i and width w_i of R_i can be stretched or shrunk within prespecified ranges, while keeping the area α_i of R_i unchanged. This problem was studied in [WKC88, DCL89, WKC89, VTs90] and proved to be NP-complete in [Hak88]. Another variation to the rectangle packing problem, known as the column packing problem, consists on packing a set of rectangle $\mathcal{F} = \{R_1, \ldots, R_n\}$ into a single column R of minimal area. In this case, we require the elements of

$\mathcal{F}$ to be stacked in the column. This case was studied in [Hak88]. In the same paper Hakimi obtained an $O(n \log n)$ algorithm to solve this problem.

In this paper we present an $O(n \log n)$ algorithm for the column packing problem allowing the rectangles of $\mathcal{F}$ to be elastic.

2 Formulation of the Problem for a Single Column

We want to develop an algorithm with the following parameters:

Input.- A family of elastic rectangles $\mathcal{F} = \{R_1, R_2, \ldots, R_n\}$. A fixed area α_i for each rectangle R_i. The *aspect ratios* γ_i, β_i which determine the ranges in which the length l_i and the width w_i of the rectangle R_i can vary, while maintaining the area α_i of R_i constant; $i = 1, \ldots, n$. The aspect ratios satisfy $1 \leq \gamma_i \leq l_i/w_i \leq \beta_i$ with $i = 1, \ldots, n$.

Output.- A rectangle R of minimum area such that the rectangles of $\mathcal{F}$ are packed in R stacked on a single column.

Operations.- Two types of operations are allowed:

1) Rotations by 90^0.

2) Stretching and shrinking of the rectangles in $\mathcal{F}$ such that their areas are preserved.

We suppose without loss of generality that $l_i \geq w_i$ for all $i = 1, 2, \ldots, n$. It is easy to see that the following inequalities hold: $\sqrt{\alpha_i/\beta_i} \leq w_i \leq \sqrt{\alpha_i/\gamma_i} \leq \sqrt{\alpha_i\gamma_i} \leq l_i \leq \sqrt{\alpha_i\beta_i}$. Without loss of generality we assume the strict inequalities $\sqrt{\alpha_i/\beta_i} < \sqrt{\alpha_i/\gamma_i} < \sqrt{\alpha_i\gamma_i} < \sqrt{\alpha_i\beta_i}$. Our algorithm can be easily modified to solve the case when equalities are allowed.

We notice that, if for a rectangle R_i we maximized its length l_i ($l_i = \sqrt{\alpha_i\beta_i}$) then its width w_i is minimized ($w_i = \sqrt{\alpha_i/\beta_i}$). Similarly if we minimized its length then its width is maximized.

We can observe that a rectangle R is a solution to our problem if it minimizes the waste area contained in it that is the space within R not covered by any rectangle of $\mathcal{F}$. We use this idea in our algorithm by minimizing the waste area contained in R rather than the actual area of R.

3 ALGORITHM

First, we give an intuitive idea of solve this problem works. Intuitively speaking our algorithm starts by laying down all of our rectangles so that their bases are maximized and parallel to the column base. This way all the rectangles are stretched as much as they are allowed, thus obtaining an initial solution with minimal height. We then proceed to ***shrink*** the base of our enclosing rectangle thus increasing the height of it and during the process we ***shrink*** or rotate one by one the elements of $\mathcal{F}$.

During the shrinking process of the base B of our column, for each elastic rectangle R_i of $\mathcal{F}$ four critical positions have to be analyzed. Since at some point of our algorithm each rectangle R_i is rotated by 90^o, the "base" of R_i, that is the side of it parallel to the base B of our column, could be the side determined by its length or that determined by its width. Thus for our algorithm, we need a parameter b_i which indicates the size of the base of R_i at any given time during the execution of our algorithm.

Moreover, each R_i generates up to four critical *positions* which will be indicated by a variable $p(i)$, i.e.:

a) $p(i) = 1$ if b_i has value $l_i = \sqrt{\alpha_i \beta_i}$.

b) $p(i) = 2$ if the b_i has value $l_i = \sqrt{\alpha_i \gamma_i}$.

c) $p(i) = 3$ if b_i has value $w_i = \sqrt{\alpha_i / \gamma_i}$.

d) $p(i) = 4$ if b_i has value $w_i = \sqrt{\alpha_i / \beta_i}$.

The starting configuration of our algorithm can be describe as follows:

a) Each rectangle R_i of $\mathcal{F}$ is laid down in a column in such a way that its base b_i is maximized. In this case we can see that each rectangle R_i is in critical position $p(i) = 1$, $i = 1, \ldots, n$. We call this state of the column the **initial state.**

b) An initial packing rectangle R is formed and the waste area contained in it is calculated.

c) Get the rectangle R_{i_0} which base $b_{i_0} = \max\{\sqrt{\alpha_i \beta_i}$, for $i = 1, \ldots, n\}$.

d) Let $B = b_{i_0}$.

We now begin to shrink B. Let us focus our attention on the behavior of a single rectangle R_i.

While the size of B is greater than or equal to $\sqrt{\alpha_i \beta_i}$, the shape of R_i remains unchanged. Once the size of B enters the interval $[\sqrt{\alpha_i \gamma_i}, \sqrt{\alpha_i \beta_i}]$ the length of

R_i shrinks with that of B and R_i generates no waste in R. Once the size of B reaches $\sqrt{\alpha_i\gamma_i}$ in order to keep on shrinking B, R_i has to be rotated (for otherwise R_i would prevent us from shrinking B any further). Similarly once R_i has been rotated, we need to deal with it only at the times when the size of B enters and leaves the interval $[\sqrt{\alpha_i/\beta_i}, \sqrt{\alpha_i/\gamma_i}]$.

In view of all of the above, it follows that each rectangle R_i has to be considered at the four critical positions defined before, $i = 1, \ldots, n$; i.e. when the size of R equal: $\sqrt{\alpha_i\beta_i}, \sqrt{\alpha_i\gamma_i}, \sqrt{\alpha_i/\gamma_i}, \sqrt{\alpha_i/\beta_i}$. Therefore, during the shrinking process of the base B of our column, we have to consider, in decreasing order all of the points of

$$S = \cup_{i=1}^{n}\{\sqrt{\alpha_i\beta_i}, \sqrt{\alpha_i\gamma_i}, \sqrt{\alpha_i/\gamma_i}, \sqrt{\alpha_i/\beta_i}\}.$$

If we can update the waste area of R from one of S to the next in constant time, since $|S| = 4n$ we can update the waste area of R in $O(n)$ time.

We now give a pseudo-codification of our algorithm.

Algorithm

- Obtain the the **initial state** of the column R.
- Build a maxheap H which there is a node for each rectangle R_i in $\mathcal{F}$ that contains: i identifier of R_i, $l_{\max} = \sqrt{\alpha_i\beta_i}$, $l_{\min} = \sqrt{\alpha_i\gamma_i}$, $w_{\max} = \sqrt{\alpha_i/\gamma_i}$, $w_{\min} = \sqrt{\alpha_i/\beta_i}$, its current base b_i and height h_i, and its critical position $p(i)$. The sorting key of H is b_i.
- Get the initial pivot rectangle R_{i_0} with maximal key from the heap.
- *stop_rectangle* $\leftarrow R_{i_0}$.
- Calculate the waste area WA in the initial column:

$$\text{WA} = \sum_{i=1}^{n} \sqrt{\alpha_i/\beta_i}\left(stop\_rectangle.b_i - \sqrt{\alpha_i\beta_i}\right).$$

- Initialize the minimum Waste Area $\bar{\text{WA}} \leftarrow$ WA.
- Compute the Height of the initial Waste Area region

$$\text{HWA} = \left(\sum_{i=1}^{n} \sqrt{\alpha_i/\beta_i}\right) - stop\_rectangle.h_i.$$

- **While** $p(i_0) \neq 4$:
  1. *old_stop* $\leftarrow R_{i_0}$.

2. Move the rectangle R_{i_0} to the next critical position, $p(i_0) \leftarrow p(i_0)+1$. Update b_{i_0} and h_{i_0}.
   If $p(i_0) = 3$, the rectangle R_{i_0} has been rotated then update the waste area region and its height:

$$\text{WA} = \text{WA} + R_{i_0}.h_{i_0}\,(stop\_rectangle.b_i - R_{i_0}.b_{i_0})$$

$$\text{HWA} = \text{HWA} + R_{i_0}.h_{i_0}$$

   and update the minimum waste area $\bar{\text{WA}} \leftarrow \text{WA}$.
3. Insert R_{i_0} in the maxheap with the new critical position and its new base b_{i_0}.
4. Get the new pivot rectangle R_{i_0} from the maxheap:
   $stop\_rectangle \leftarrow R_{i_0}$.
5. Update the waste area region WA for the following cases:
   If $stop\_rectangle = old\_stop$ and
   $((stop\_rectangle.p(i) = 2$ and $old\_stop.p(i) = 1)$ or $(stop\_rectangle.p(i) = 4$ and $old\_stop.p(i) = 3))$, then we update only the waste area WA (the height of this region does not change, since the rectangle R_{i_0} is shrunk together with the base of the column).

$$\text{WA} = \text{WA} - \text{HWA}\,(old\_stop.b_i - stop\_rectangle.b_i).$$

   Else, we have to recompute the waste area and the height:

$$\text{WA} = \text{WA} - \text{HWA}\,(old\_stop.b_i - stop\_rectangle.b_i)$$

$$\text{HWA} = \text{HWA} - stop\_rectangle.h_i.$$

6. $\bar{\text{WA}} = \min(\bar{\text{WA}}, \text{WA})$.

end {While }

We now proceed with the complexity analysis of our algorithm. To put the rectangles in initial position and build the heap takes $O(n)$ time. The computation of the waste area and the height of the waste area takes constant time. The loop is executed $3n + 1$ times. Each time the loop is executed we delete and insert on the heap which takes time $O(\log n)$. Therefore the algorithm takes $O(n \log n)$ time.

References

[GJT80] E. G. Coffman, Jr. M.R. Garey, D.S. Johnson and R. E. Tarjan, *Performance Bounds for Level-Oriented two-dimensional Packing Algorithms*, SIAM J. COMPUT. Vol.9 (1980) 808-826.

[Sto83] L. Stockmeyer, *Optimal Orientations of Cells in Slicing Floorplan Designs*, Information and Control 57 (1983), 91-101.

[Ott83] R. H. J. Otten, *Efficient Floorplan Optimization*, IEEE International Conference On Computer Design in VLSI in Computers, (1983), 499-502.

[WKC88] S.Wimer, I. Koren and I. Cederbaum, *Floorplans, Planar Graphs, and Layouts*, IEEE Trans. Circuits and Systems (1988), 267-278.

[Hak88] S. L. Hakimi, *A Problem on Rectangular Floorplans*, Proc. Int. Symp. on Circuits and Systems (1988), 1533-1536.

[DCL89] S. Dong, J. Cong and C.L. Liu, *Constraint Floorplan Design for Flexible Blocks*,IEEE International Conference On Computer-Aided Design (1989), 488-491.

[WKC89] S. Wimer, I. Koren and I. Cederbaum, *Optimal Aspect Ratios of Building Blocks in VLSI*, IEEE Trans. Computer Aided Design of Integrated Circuits and Systems (1989), 139-145.

[VTs90] G. Vijayan, R.S. Tsay, *Floorplan by Topological Constraint Reduction*, IEEE International Conference On Computer-Aided Design (1990), 106-109.

[GJo79] M. Garey and D. Johnson, *Computers and intractability*, Freeman (1979).