Journal of Universal Computer Science, vol. 3, no. 6 (1997), 703-720
submitted: 16/12/96, accepted: 23/5/97, appeared: 28/6/97 © Springer Pub. Co.

Free space modeling for placing rectangles without overlapping

Marc Bernard
(Centre de Recherche en Informatique de Dijon
9, Avenue Alain Savary - B.P. 400
21011 DIJON Cedex - FRANCE
e-mail : *Marc.Bernard@crid.u-bourgogne.fr*)

François Jacquenet
(Centre de Recherche en Informatique de Dijon
9, Avenue Alain Savary - B.P. 400
21011 DIJON Cedex - FRANCE
e-mail : *Francois.Jacquenet@crid.u-bourgogne.fr*)

Abstract: The placement of rectangular objects without overlapping on a bounded surface is a generic problem that may have many applications. Space planning, chipset placement, cutting-stock problems, point-feature label placement, or the placement of articles on a newspaper page, are all instances of this more abstract problem.
All these applications are concerned with the insertion of rectangular objects on a part of a bounded free surface. It is therefore important to be able to efficiently model the free space of the bounded surface.
In this article we present a method to compute free space. The method is based on an iterative insertion process. Our algorithm neither depends on the size of the object to insert, nor on the method of placement. The first feature improves efficiency, while the second allows us to compare different placement methods, and to parameterize the placement system using resolution heuristics.

Key Words: Placement, algorithms, free space

Category: I.3, I.3.5, I.3.6, I.6

1 Introduction

Many problems involve the placement of rectangular objects without overlapping on a bounded rectangular surface. There are also problems that at first sight do not appear to have anything in common with the problem of the placement of rectangles, but nevertheless can be reduced to this form.

We quote some significant examples which have been the subject of research work.

In the field of space planning, 3D objects to be placed in a house or in a room can often be approximated by rectangles if we consider their projection onto the placement plane [Charman 93], [Charman 95], [Mizoguchi, Ohwada 95].

Cutting-stock optimization problems are also linked to the general problem [Dincbas Simonis Van Hentenryck 88]. In such applications we have a set S of predefined rectangular objects and seek to cut an area in order to obtain all the elements of S. Some cuts may be forbidden by sawing constraints, which often leads to a reduction of the complexity of the problem.

The placement of rectangles may also be used in several other fields such as chipset placement on integrated circuits [Du Verdier 90], the composition of pages in computer-aided publication [Bernard, Jacquenet, Nicolini 97], point-feature label placement [Christensen, Marks, Shieber 95] or the design of more intelligent graphical user interfaces [Cruz, Marriott, Van Hentenryck 95].

When we wish to place objects on a surface without overlapping, we need to know, whatever the constraint solving technique we use [Jaffar, Maher 94], [Hower, Graf 95], where new objects can subsequently be inserted. Thus we always need to be able to model the free space left by already-inserted objects in order to allow placement of other objects on the bounded surface.

Let W be a rectangle and S be a set of rectangles.
Later, we will call W the base rectangle.

Problem 1: find a placement of the rectangles of S without overlapping on the rectangle W.

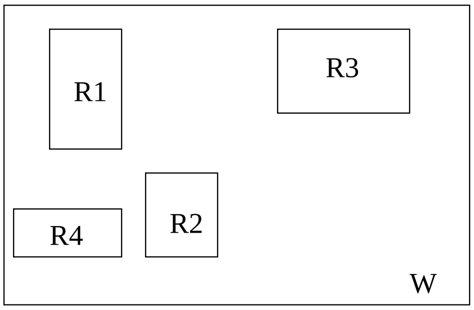

Figure 1: Placement of four rectangles R_1, R_2, R_3, R_4 on a bounded surface W

Problem 2: suppose that n rectangles $R_1, R_2, \ldots, R_n$ have already been placed on the rectangle W. Can a rectangle R_{n+1} be inserted in $W-(R_1 \cup R_2 \cup \cdots \cup R_n)$?

In this article our goal is to model efficiently the free space. We expect a modeling method to be independent of the object to be inserted, and of the placement method.

In the next section we briefly describe three placement methods in current use. For each, we lay emphasis on the free space calculation methods they employ, and discuss their advantages and possible drawbacks. Later, we introduce a new free space calculation method. We present the algorithm and illustrate its behaviour using an example. Finally, we prove the completeness and correctness of the new algorithm.

2 Existing methods

In this section we present several methods for placing rectangles on a rectangular bounded surface. For each method we focus on the free space calculation. First, we consider methods which are integrated with a placement algorithm. We then present an iterative calculation method, and finally a free space calculation algorithm based on the notion of extended rectangles.

2.1 Free space calculation integrated with the placement algorithm

In many placement systems for rectangular objects subject to geometrical constraints, the free space calculation depends on the placement method. This is an efficient way to proceed when we wish to design a complete system, but the approach makes it difficult to compare different resolution algorithms, or different free space calculation algorithms. Moreover, the approach does not allow us to design a generic placement system which may be parameterized by placement heuristics or free space calculation methods.

The main idea in such systems is to draw a parallel between rectangle placement systems and a constraint system over a reference domain, such as integers.

A rectangle is defined by its position and its dimensions. The rectangle domain - also called a rectangle configuration - is the set of positions and dimensions the rectangle can take. The position of a rectangle is represented by the position of a particular point, called a referencing point. Figure 2 draws the parallel between a constraint satisfaction problem and a geometrical constraint satisfaction problem (CSP) over rectangles. Thus, rectangles correspond to CSP variables, and configurations correspond to variable domains.

CSP	Rectangles placement problem
variable	rectangle
domain	configuration
constraint	geometrical constraint

Figure 2: Analogy between the system of placement of rectangles and CSP

When we use CSP systems over integers, each variable has its own domain. When we add a new constraint to the set of constraints of the system, a filtering method may reduce variable domains if necessary.

Choosing such a method involves each rectangle having its own domain, which depends not only on the dimensions and the positions of other rectangles, but also on its own dimensions.

During the resolution of a problem of the placement of rectangles, if the system backtracks on the choice of the position or dimension of a rectangle, all rectangle domains have to be updated. In fact, we compute the free space which corresponds to each rectangle.

Our approach is to design a free space calculation method which is completely independent of subsequent placements, and hence is closer to the methods we consider in the following two subsections.

2.2 Iterative placement method

In this subsection we present the method described in [Du Verdier 90]. Although the method attempts to place interconnected rectangular objects, it also has some interesting principles for the placement of non-connected rectangles.

In this method the placement of rectangles proceeds in two steps. The first step deals essentially with rectangle placement respecting the defined connections. The second step distributes the rectangles without overlapping. Briefly, the two steps consist of:

- perform a factorial analysis based on the distance between the blocks. This step allows us to place the rectangles by considering them as points.
- move and distort the blocks. Following the first step, an iterative process searches a block distribution in the bounded placement surface without overlapping. The distribution has to respect the initial configuration as much as possible. At each iteration, all blocks are moved. The process stops when the displacement of each block is no longer significant.

In the second step, we search the displacement area for each block. This area is computed by determining the blocks whose boundaries are closest to the block being processed. The search is performed in the four directions (top, bottom, right and left) and defines a possible displacement area for this block (cf. figure 3).

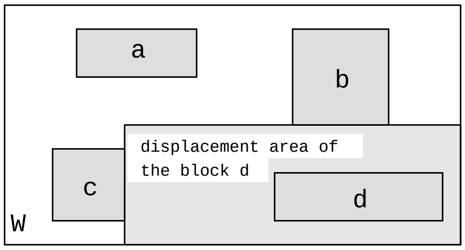

Figure 3: Displacement area of block d

We now consider the iterative placement process. To eliminate the possibility of cycling, a square shape is assigned to each block at the beginning of the process, with the area of each square equal to the area of the corresponding initial block. At each iteration, the block is distorted, preserving a constant area, with a view to restoring its initial shape. The distortion of a block depends on its displacement area, its previous shape, and of course its initial shape (cf. figure 4).

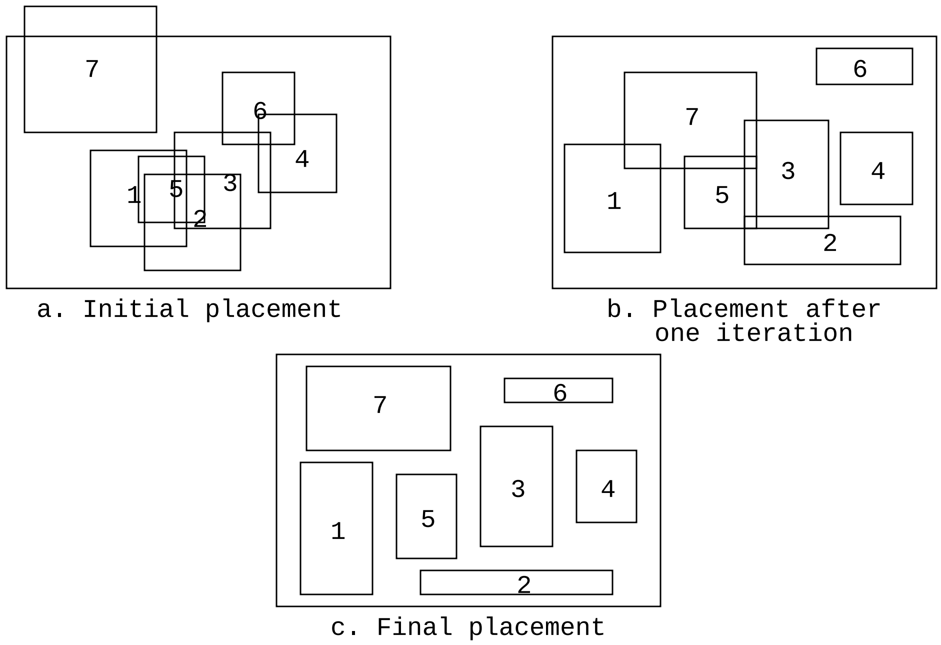

Figure 4: Placement and distortion of the blocks

The main drawback of the method is that the rectangles are initially considered as points without taking their final size into account. This leads to overlapping problems in some configurations, as in figure 5.

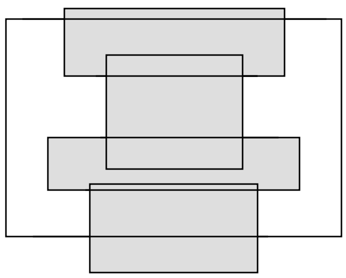

Figure 5: Example of a non-resolved configuration

2.3 Admissible region and extended rectangles

The following method is described in [Tokuyama, Asano, Tsukiyama 91].

Suppose n rectangles $R_i, i \in [1, n]$ have already been placed on the rectangle W without overlapping. To place an $(n+1)^{th}$ rectangle, we construct the region which consists of all possible placements of the right-upper corner of R_{n+1}. This region is called the admissible region.

This notion of the admissible region of a rectangle is separate from the notion of free space. In fact, the admissible region is the geometric area where a reference point of a rectangle can be placed in order to place the rectangle entirely within the free space.

The admissible region is computed considering the notion of extended rectangles using the following notation:

. we denote by $R = [l, r] \times [b, t]$ the rectangle constructed from the points (l, b), (l, t), (r, t), (r, b)
. we denote by $R[\alpha, \beta]$ the extended rectangle defined by $R[\alpha, \beta] = [l, r + \alpha] \times [b, t + \beta]$

We also introduce the notion of a contracted rectangle $R_{\alpha,\beta}$:

. $R_{\alpha,\beta} = [l + \alpha, r] \times [b + \beta, t]$

To insert a rectangle R_{n+1} of width α and height β, we consider all extended rectangles $R_i[\alpha, \beta], i \in [1, n]$, and the contracted rectangle $W_{\alpha,\beta}$.

Figure 6 shows extended rectangles of the rectangles which have already been placed on W, and the contracted rectangle of W.

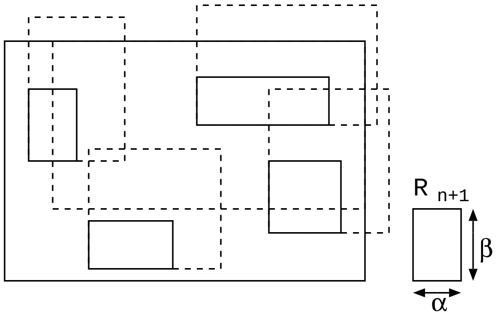

Figure 6: Extended and contracted rectangles (dotted lines)

The admissible region of the right-upper corner of R_{n+1} (cf. figure 7) is given by the following expression:

$$W_{\alpha,\beta} - \bigcup_{i=1}^{n} R_i[\alpha, \beta]$$

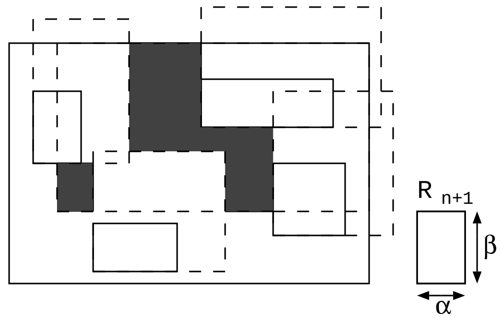

Figure 7: Calculation of the admissible region of the right-upper corner

This method has a significant drawback, since it depends on the rectangle to be inserted. Indeed, we see from the above formula that the free space calculation depends on the extended rectangles, which are directly computed using the size of the rectangle to be inserted.

In the context of solving the problem of the placement of rectangles, if the choice of the rectangle to be inserted is to be taken into account, the free space must be entirely re-computed.

3 A new method for the calculation of free space

In the method we now present, we represent the free space by a set of rectangles, called the set of largest free rectangles.

A rectangle which can be placed on the rectangle W will necessarily be included in one of the largest free rectangles.

We introduce the definitions needed to present the algorithm.

3.1 Definitions

Free rectangle
Assume that n rectangles $R_1, R_2, \ldots, R_n$ have already been placed on a rectangle W. Then R is a free rectangle if and only if:

- $R \subset W$
- $\forall R_i, i \in [1, n], R \cap R_i = \emptyset$

Largest free rectangle (cf. figure 8)
R is a largest free rectangle if and only if:

- R is a free rectangle
- $\nexists R^{'}, R^{'}$ a free rectangle, such that $R^{'} \subset R$

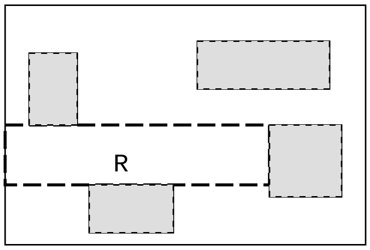

Figure 8: A largest free rectangle R

Rectangle generated by four segments (cf. figure 9)
Let E_1, E_2, E_3 and E_4 be four segments such that:

- E_i is the segment $[(x_{E_i}, y_{E_i}), (x'_{E_i}, y'_{E_i})]$
- $x'_{E_1} = x_{E_1}$ and $x'_{E_2} = x_{E_2}$ (that is, segments E_1 and E_2 are vertical)
- $y'_{E_3} = y_{E_3}$ and $y'_{E_4} = y_{E_4}$ (that is, segments E_3 and E_4 are horizontal)
- $x_{E_1} < x_{E_2}$ (that is, E_1 is to the left of E_2)
- $y_{E_3} < y_{E_4}$ (that is, E_3 is below E_4)
- $\max(y_{E_1}, y'_{E_1}) > y_{E_3}$
- $\min(y_{E_1}, y'_{E_1}) < y_{E_4}$
- $\max(y_{E_2}, y'_{E_2}) > y_{E_3}$
- $\min(y_{E_2}, y'_{E_2}) < y_{E_4}$
- $\max(x_{E_3}, x'_{E_3}) > x_{E_1}$
- $\min(y_{E_3}, y'_{E_3}) < x_{E_2}$
- $\max(x_{E_4}, x'_{E_4}) > x_{E_1}$
- $\min(y_{E_4}, y'_{E_4}) < x_{E_2}$

Then the rectangle generated by the segments E_1, E_2, E_3 and E_4 is:

$$\mathcal{R}(E_1, E_2, E_3, E_4) = [x_{E_1}, x_{E_2}] \times [y_{E_3}, y_{E_4}]$$

Figure 9b shows a situation in which four segments do not generate a rectangle corresponding to our definition; condition $\max(x_{E_4}, x'_{E_4}) > x_{E_1}$ does not hold.

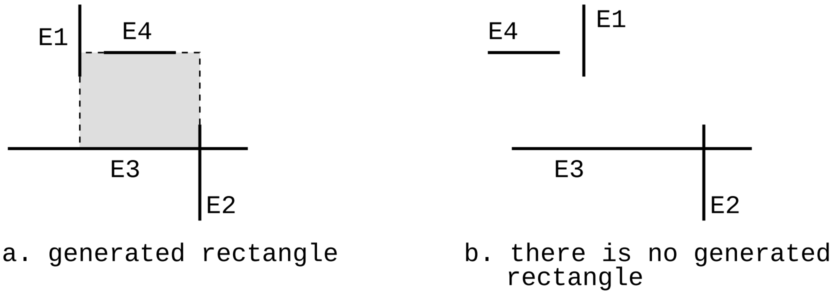

Figure 9: Method for rectangle generation

Rectangle modified by a segment
We say that a rectangle is *modified by a segment* if its intersection with the segment is not empty.

Reduced rectangle (cf. figure 10)
For a rectangle $R = [l, r] \times [b, t]$ and a segment $[(x_1, y_1), (x_2, y_2)]$ we define the *left-reduced rectangle*[1] to be the rectangle

$$\Box^{left}_{[(x_1,y_1),(x_2,y_2)]}(R) = [\max(x_1, x_2), r] \times [b, t]$$

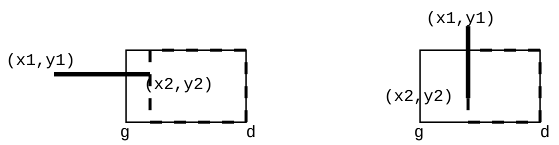

Figure 10: Left-reduced rectangle

In a similar way, we can define right-, top-, and bottom-reduced rectangles.

We denote by $\Box^*(R)$ a reduced rectangle of R without specifying if it is left-, right-, top- or bottom-reduced.

3.2 Algorithm

The free space calculation method we now propose is an incremental method. The set of largest free rectangles is computed by considering the successive insertions of the rectangles placed on the rectangle W.

Let $LFRSet$ be the set of the largest free rectangles.
We also employ $NewSet$ and $RemoveSet$ as two sets of rectangles.
The free space calculation algorithm is:

---
[1] The notion of a left-reduced rectangle is defined if and only if $\max(x_1, x_2) < r$.

$LFRSet \leftarrow \{W\}$
for each rectangle R_i to be inserted
  $NewSet \leftarrow \emptyset$, $RemoveSet \leftarrow \emptyset$
  let E_r be the right edge of R_i
    . let B be the set of rectangles of $LFRSet$ which are modified by E_r
    . add all the left-reduced rectangles of B by E_r to $NewSet$
    . add the rectangles of B to $RemoveSet$
  let E_b be the bottom edge of R_i
    . let B be the set of rectangles of $LFRSet$ which are modified by E_b
    . add all the top-reduced rectangles of B by E_b to $NewSet$
    . add the rectangles of B to $RemoveSet$
  let E_l be the left edge of R_i
    . let B be the set of rectangles of $LFRSet$ which are modified by E_l
    . add all the right-reduced rectangles of B by E_l to $NewSet$
    . add the rectangles of B to $RemoveSet$
  let E_t be the top edge of R_i
    . let B be the set of rectangles of $LFRSet$ which are modified by E_t
    . add all the bottom-reduced rectangles of B by E_t to $NewSet$
    . add the rectangles of B to $RemoveSet$
  $LFRSet \leftarrow LFRSet - RemoveSet$
  $LFRSet \leftarrow LFRSet \cup NewSet$

When the algorithm terminates, $LFRSet$ contains all the largest free rectangles.

Note: when we add rectangles with this method, we need to make an inclusion test to ensure that only the largest free rectangles are added.

3.3 Example

Consider the following example where we wish to model the free space remaining after the insertion of two rectangles R_1 and R_2 (see figure 11).

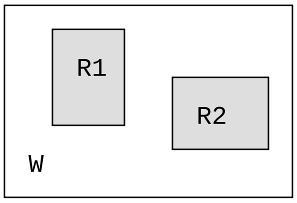

Figure 11: Placement of two rectangles R_1, R_2 on a bounded surface W

After the initialization step we have $LFRSet = \{W\}$, $NewSet = \emptyset$ and $RemoveSet = \emptyset$.

We begin with the insertion of the rectangle R_1. Consider the right edge of this rectangle: it modifies only one element of $LFRSet$, namely the rectangle W.

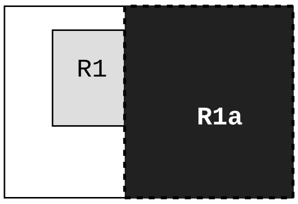

Figure 12: Consideration of the right edge of the rectangle R_1

R_1^a, the left-reduced rectangle of W by the right edge of R_1 is added to $NewSet$, and W is added to $RemoveSet$ We now have: $LFRSet = \{W\}$, $NewSet = \{R_1^a\}$ and $RemoveSet = \{W\}$.

Consideration of the three remaining edges of the rectangle R_1 leads to the identification of the three black rectangles of figure 13. Thus we have $LFRSet = \{W\}$, $NewSet = \{R_1^a, R_1^b, R_1^c, R_1^d\}$, and $RemoveSet = \{W\}$.

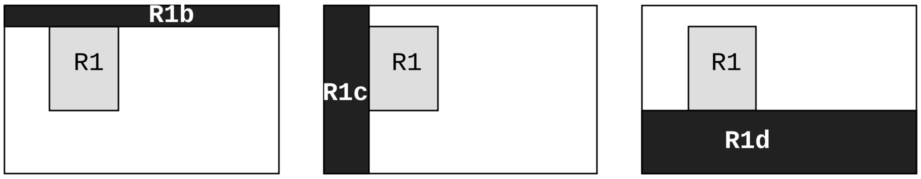

Figure 13: Consideration of the remaining edges of the rectangle R_1

The final two steps of the algorithm applied to R_1 lead to the result $LFRSet = \{R_1^a, R_1^b, R_1^c, R_1^d\}$.

We now insert the second rectangle, R_2.

The right edge of this rectangle *modifies* two rectangles of the set $LFRSet$, namely R_1^a and R_1^d. The two reduced rectangles R_2^a and R_2^b (cf. figure 14) should consequently be added to $NewSet$.

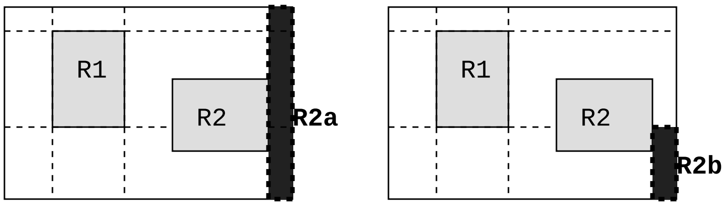

Figure 14: Consideration of the right edge of R_2

When these rectangles are added to $NewSet$, an inclusion test is performed. Here, one of the two generated rectangles is included in the other ($R_2^b \subset R_2^a$) and hence the rectangle R_2^b, which is not a largest free rectangle, is not added to $NewSet$. We now have: $LFRSet = \{R_1^a, R_1^b, R_1^c, R_1^d\}$, $NewSet = \{R_2^a\}$ and $RemoveSet = \{R_1^a, R_1^d\}$.

We do not elaborate on the process associated with the bottom edge of R_2 since the situation is similar to the previous case. We simply observe that during this step the rectangle R_2^c is added to $NewSet$ (cf. figure 15).

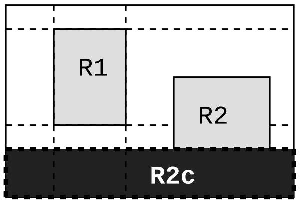

Figure 15: Consideration of the bottom edge of R_2

We now consider the left edge of the rectangle R_2 which *modifies* two rectangles of $LFRSet$, namely R_1^a and R_1^d. The two rectangles R_2^d and R_2^e of figure 16 are added to $NewSet$. We now have: $LFRSet = \{R_1^a, R_1^b, R_1^c, R_1^d\}$, $NewSet = \{R_2^a, R_2^c, R_2^d, R_2^e\}$ and $RemoveSet = \{R_1^a, R_1^d\}$.

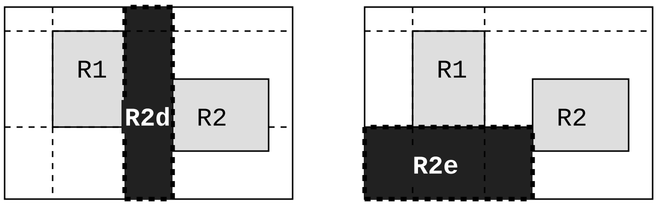

Figure 16: Consideration of the left edge of R_2

Finally, the top edge of R_2 concerns only the rectangle R_1^a and consideration of this edge leads to the addition of the rectangle R_2^f to $NewSet$ (cf. figure 17). Hence we have: $LFRSet = \{R_1^a, R_1^b, R_1^c, R_1^d\}$, $NewSet = \{R_2^a, R_2^c, R_2^d, R_2^e, R_2^f\}$ and $RemoveSet = \{R_1^a, R_1^d\}$.

After processing the second rectangle R_2, $LFRSet$ contains seven rectangles, that is, $LFRSet = \{R_1^b, R_1^c, R_2^a, R_2^c, R_2^d, R_2^e, R_2^f\}$.

3.4 Completeness

Let W_n be the rectangle W on which the rectangles R_i ($i \in [1, n]$) have been placed.

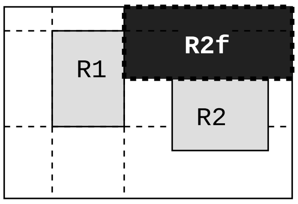

Figure 17: Consideration of the top edge of R_2

Let $LFRSet_n$ be the set of the largest free rectangles of W_n constructed by our algorithm.

We remark that the algorithm constructs $LFRSet_{n+1}$ from $LFRSet_n$ by:

. removing from $LFRSet_n$ the rectangles which are modified by the edges of R_{n+1}

. adding to $LFRSet_{n+1}$ the reduced rectangles of the rectangles of $LFRSet_n$ which are modified by the edges of R_{n+1}

We denote by $\mathcal{E}_n$ the set of edges of the rectangles $R_i, i \in [1, n]$ and of the rectangle W.

We introduce two lemmas which are useful for the proof of completeness.

3.4.1 Lemma 1

The two following assertions are equivalent:

- R is free and $R = \mathcal{R}(E_1, E_2, E_3, E_4)$, where $E_i \in \mathcal{E}_n$
- R is a largest free rectangle

Proof

Let $R = \mathcal{R}(E_1, E_2, E_3, E_4)$ be a free rectangle.
We extend the rectangle R in the direction of E_i.
Let R_j be the rectangle of which E_i is an edge.
The extended rectangle of R has a non empty intersection with R_j (see figure 18).
If the edge E_i is an edge of W, then the extended rectangle does not belong to W. Thus the extended rectangle is not a free rectangle.
Therefore, there does not exist a free rectangle which contains $\mathcal{R}(E_1, E_2, E_3, E_4)$, which proves that $\mathcal{R}(E_1, E_2, E_3, E_4)$ is a largest free rectangle.

Conversely, let R be a largest free rectangle. Then R is a free rectangle.
Suppose that R cannot be written in the form $\mathcal{R}(E_1, E_2, E_3, E_4)$. This means that there exists at least one edge E of R such that $\forall i\ E \notin R_i$.
We can therefore extend the rectangle R to construct R', a free rectangle which contains R.
Hence R is not a largest free rectangle, which contradicts the original assumption.

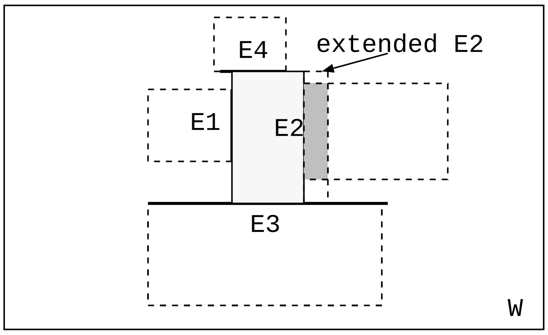

Figure 18: The extended rectangle is not free

3.4.2 Lemma 2

Let three edges E_1, E_2 and E_3 belong to $\mathcal{E}_n$ such that:

. E_i is the segment $[(x_{E_i}, y_{E_i}), (x'_{E_i}, y'_{E_i})]$
. E_1 is the right edge of a rectangle R_i
. E_2 is the left edge of a rectangle R_j
. E_3 is the top edge of a rectangle R_k
. $x_{E_1} < x_{E_2}$ (that is, E_1 is to the left of E_2)
. $\max(y_{E_1}, y'_{E_1}) > y_{E_3}$
. $\max(y_{E_2}, y'_{E_2}) > y_{E_3}$
. $\max(x_{E_3}, x'_{E_3}) > x_{E_1}$
. $\min(y_{E_3}, y'_{E_3}) < x_{E_2}$
. the rectangle $[x_{E_1}, x_{E_2}] \times [y_{E_3}, \max(\min(y_{E_1}, y'_{E_1}), \min(y_{E_2}, y'_{E_2}))]$ is free

Then there exists $E_4 \in \mathcal{E}_n$ parallel to E_3 such that $R = \mathcal{R}(E_1, E_2, E_3, E_4)$ is a largest free rectangle.

We can generalize this Lemma to the three remaining symmetrical cases.

Proof

In Lemma 2 we deal with a configuration which is similar to that of figure 19.
Let Π be that part of the plane defined by the set of points (x, y) such that:

. $x_{E_1} \leq x \leq x_{E_2}$
. $y \geq y_{E_3}$

Let $\mathcal{E}$ be the set of edges E_i ($i \in [1, n]$) such that $E_i \cap \Pi \neq \emptyset$.
Let $E \in \mathcal{E}$ be such that

. $\min(y_E, y'_E) = \min_{E_i \in \mathcal{E}}(y_{E_i}, y'_{E_i})$
. $y_E \geq \max(\min(y_{E_1}, y'_{E_1}), \min(y_{E_2}, y'_{E_2}))$

Then $R = \mathcal{R}(E_1, E_2, E_3, E)$ satisfies the definition of a rectangle generated by edges. Moreover, R is free because the edge E has been chosen to have the

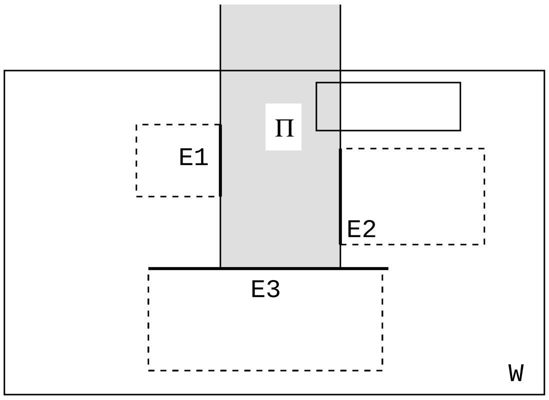

Figure 19: Illustration of the hypothesis of the proof of lemma 2

smallest y-ordinate in Π. Therefore, from Lemma 1 it follows that R is a largest free rectangle.

Hence there exists an edge $E_4 = E$ such that $R = \mathcal{R}(E_1, E_2, E_3, E_4)$ is a largest free rectangle.

3.4.3 Proof of completeness

We wish to prove that $LFRSet_{n+1}$ contains all the largest free rectangles of W_{n+1}.

To do this, we prove by induction that if R is a largest free rectangle for W_{n+1}, then it belongs to the set of the largest free rectangles $LFRSet_{n+1}$ computed by our algorithm.

We first prove that the assumption is true for $n = 0$.

It is trivial to show that the only largest free rectangle of W_0 is W. Our algorithm begins with the initialization of $LFRSet$ to the set $\{W\}$, and terminates immediately because there are no more rectangles R_i to place on W. Thus $LFRSet_0 = \{W\}$.

Induction assumption: Suppose that $LFRSet_n$ contains all the largest free rectangles of W_n.

We now prove that if R is a largest free rectangle of W_{n+1}, then $R \in L_{n+1}$.

Let R be a largest free rectangle of W_{n+1}. Since R is a largest free rectangle, from Lemma 1 it can be written $R = \mathcal{R}(E_1, E_2, E_3, E_4)$, where E_1, E_2, E_3, E_4 are edges of W or $R_i, i \in [1, n+1]$.

case 1: $\{E_1, E_2, E_3, E_4\}$ contains no edge of R_{n+1}.

We consider W_{n+1} without R_{n+1}, that is W_n.

$\mathcal{R}(E_1, E_2, E_3, E_4)$ is a largest free rectangle of W_{n+1}, and therefore $R \cap R_i = \emptyset \; \forall i \in [1, n+1]$.

Since $R = \mathcal{R}(E_1, E_2, E_3, E_4)$ with the edges E_i existing in W_n, and $R \cap R_i = \emptyset \; \forall i \in [1, n]$, Lemma 1 leads us to conclude that R is a largest free rectangle of W_n. Following the induction assumption, we have $R \in L_n$.

Now, the algorithm computes $LFRSet_{n+1}$ from $LFRSet_n$ by removing from $LFRSet_n$ the rectangles which are modified by the edges of R_{n+1}. Since $R \cap R_{n+1} = \emptyset$, no edge of R_{n+1} modifies R. Hence R is not removed from $LFRSet_n$, and thus we have $R \in L_{n+1}$.

case 2: $\{E_1, E_2, E_3, E_4\}$ contains an edge of R_{n+1}. We suppose here that this edge is E_4. If the edge is other than E_4 the proof follows in a similar manner.

We consider W_{n+1} without R_{n+1}, that is W_n.

R is a free rectangle, and so from Lemma 2 we can find an edge E of a rectangle $R_i, i \in [1, n]$ such that $S = \mathcal{R}(E_1, E_2, E_3, E)$ is a largest free rectangle.

We observe that $E_4 \cap S \neq \emptyset$.

According to the induction assumption, $S \in L_n$.

By the construction of S, $R = \square^*_{E_4}(S)$ (see figure 20).

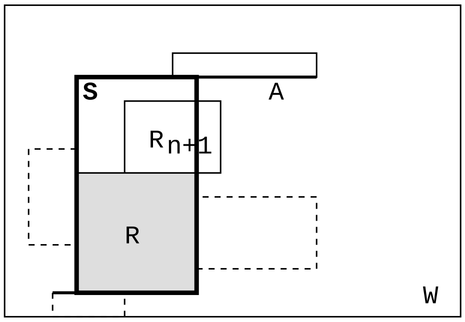

Figure 20: Illustration of case 2 of the proof of completeness

By constructing $LFRSet_{n+1}$ from $LFRSet_n$ using our algorithm, when R_{n+1} is inserted, S is modified by the edge E_4, since $E_4 \cap S = \emptyset$.

Therefore, to construct $LFRSet_{n+1}$ we add the rectangle of S reduced by the edge E_4 (that is $\square^*_{E_4}(S) = R$) to $LFRSet_n$.

This proves that $R \in L_{n+1}$.

We have proved that the assumption is true for $i = 0$, that is W_0 contains all largest free rectangles. Moreover, we have proved that if the assumption is true for $i = n$, then it is true for $i = n + 1$. Hence, by induction, we have that $\forall R$, a largest free rectangle of W_{n+1}, we have $R \in LFRSet_{n+1}$. Our algorithm effectively computes all the largest free rectangles of W_{n+1}.

3.5 Correctness

When we construct $LFRSet_{n+1}$ from $LFRSet_n$ the algorithm removes the set of rectangles affected by edges of R_{n+1} from $LFRSet_n$. Using the argument of case 1 of the completeness proof, all rectangles remaining in $LFRSet_{n+1}$ are largest free rectangles.

To prove correctness, the problem is to determine whether the algorithm adds rectangles which are not largest free rectangles.

First, by construction, the reduced rectangles which are added are free. Moreover, when a rectangle is added, the algorithm tests if the rectangle is included in an existing largest free rectangle; this test ensures correctness.

4 Conclusion

To design a free space calculation algorithm for the placement of rectangles, and considering existing methods [Charman 93] [Charman 95] [Du Verdier 90] [Tokuyama, Asano, Tsukiyama 91], we have introduced, a method which is independent of the size of the objects to be inserted and of any method of placement. This independence allows us to employ efficient placement methods which may be parameterized by some placement heuristics or by a free space calculation method. The algorithm presented here also allows comparisons between several placement heuristics to be made easily, and this is indispensable for our future research work.

Completeness and correctness of the algorithm presented here have been proved. This provides a formal basis for future development. Indeed, we aim to integrate Inductive Logic Programming [Muggleton, De Raedt 94] and Inductive Constraint Logic Programming mechanisms to provide the induction of geometrical constraints [Bernard, Jacquenet, Nicolini 97], [Mizoguchi, Ohwada 95]. In this context, the algorithm presented here will form part of the background knowledge for an inductive learning system.

References

[Bernard, Jacquenet, Nicolini 97] M. Bernard, F. Jacquenet, and C. Nicolini. Induction of Constraint Logic Programs for Computer-Aided Publishing. In *International Conference on Artificial Intelligence and Soft Computing*, Banff, Canada, July 1997.

[Charman 93] P. Charman. Solving Space Planning Problems Using Constraint Technology. Research Report CS-57/93, Institute of Cybernetics - Estonian Academy of Sciences, 1993.

[Charman 95] P. Charman. Gestion des contraintes géometriques pour l'aide à l'aménagement spatial. Thèse - Université de l'Ecole Nationale des Ponts et Chaussées, Novembre 1995.

[Christensen, Marks, Shieber 95] J. Christensen, J. Marks, and S. Shieber. An Empirical Study of Algorithms for Point-Feature Label Placement. *ACM Transaction on Graphics*, 14(3):203–232, July 1995.

[Cruz, Marriott, Van Hentenryck 95] I. Cruz, K. Marriott, and P. Van Hentenryck, editors. *Proceedings of the International Workshop on Constraints for Graphics and Visualization*, Monash University, 1995. Department of Computer Science. in association with CP'95.

[Dincbas Simonis Van Hentenryck 88] M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving a cutting-stock problem in constraint logic programming. Technical Report TR-LP-28, ECRC, January 1988.

[Du Verdier 90] F. Du Verdier. Placement de rectangles par répartition dans une surface bornée. In *MICAD '90 proceedings of the 9th International conference on the CADCAM*, volume 2, pages 634–648. Hermes, February 1990.

[Hower, Graf 95] W. Hower and W.H. Graf. Research in constraint-based layout, visualization, cad, and related topics : A bibliographical survey. In *CGV '95*, Cassis, France, September 1995.

[Jaffar, Maher 94] J. Jaffar and M.J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[Mizoguchi, Ohwada 95] F. Mizoguchi and H. Ohwada. Using inductive logic programming for constraint acquisition in constraint-based problem solving. In Luc De Raedt, editor, *5th International Workshop on Inductive Logic Programming*, pages 297–322, September 1995.

[Muggleton, De Raedt 94] S. Muggleton and L. De Raedt. Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19-20:629–679, 1994.

[Tokuyama, Asano, Tsukiyama 91] T. Tokuyama, T. Asano, and S. Tsukiyama. A Dynamic Algorithm for Placing Rectangles without Overlapping. *Journal of Information Processing*, 14(1):30–35, 1991.

Acknowledgments

The authors would especially like to thank M. Jean-Jacques Chabrier of Centre de Recherche en Informatique de Dijon, Crid (FRANCE), and M. Chris Phillips of the University of Newcastle (UK) for useful help and suggestion during this research.