

# Largest Empty Rectangle — Implementation Issues\*

Eli Packer

School of Computer Science  
Tel Aviv University  
elip@post.tau.ac.il

## Abstract

We describe an implementation issue that came up in the development of a software package for the Largest Empty Rectangle problem. The article on which the algorithm is based assumes that the input points are in general position, namely that no two points have the same  $x$  or  $y$  coordinates. This assumption is restrictive and as we show below is fairly easy to relax. We describe how we modify the algorithm to support arbitrary set of input points.

## 1 Introduction

The Largest-Empty-Rectangle problem is defined as follows: Given a set of input points in  $\mathbb{R}^2$  and a bounding box  $B$  such that all the input points are located inside  $B$ , find the largest axis-parallel rectangle that contains no points in its interior and is contained inside  $B$  (we refer to this object by Largest Empty Rectangle, or LER for short). See Figure 1 for an illustration. We use the following notation:  $P = \{p_1 \dots p_n\}$  are the set of  $n$  input points. Let  $x(p_i)$  be the  $x$  coordinate of  $p_i$ .

We developed a CGAL package that solves this problem using floating-point arithmetic. It is based on the algorithm of M. Orlowski [2]. In his paper Orlowski makes the typical assumption that the input points are in general position, and in particular that no two points have the same  $x$  or  $y$  coordinates (in what follows we refer without loss of generality only to points with the same  $x$  values). Since this assumption is somewhat artificial, and indeed two points may have

---

\*This work has been supported in part by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), by the ESPRIT LTR project No. 26473 (ECG), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

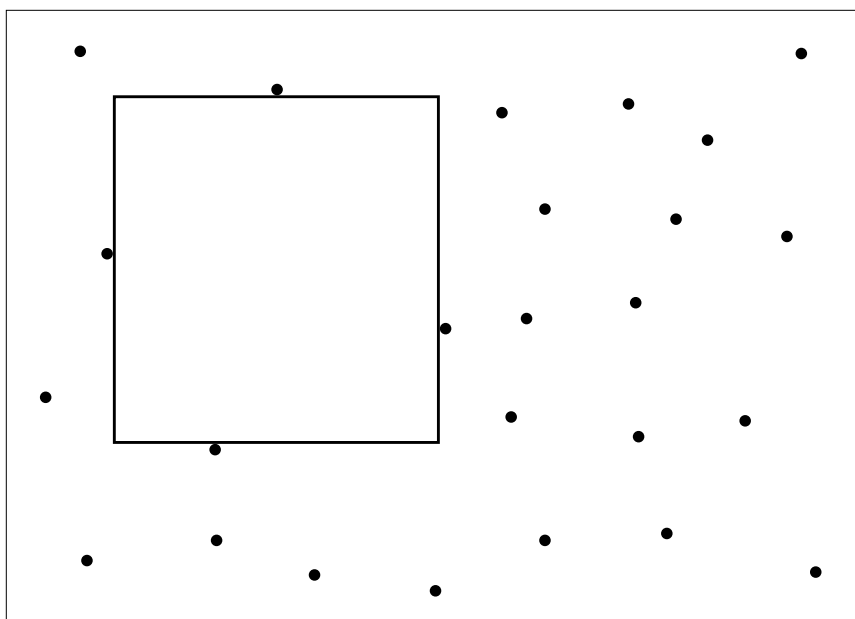


Figure 1: LER example

the same  $x$  coordinates, we have to devise a way to modify the algorithm such that it supports points with the same  $x$  or  $y$  coordinates.

We next mention three possible ways to relax the general position assumption.

**Points Perturbation.** We process each input point in its turn. Let  $x'(p_i)$  be the  $x$  coordinate value of  $p_i$  after the process on  $p_i$  has been completed. We perturb  $x(p_i)$  a little if there exists  $j < i$  such that  $x(p_i) = x'(p_j)$ . If  $x(p_i)$  is perturbed, it is perturbed such that there is no  $j < i$  such that  $x'(p_i) = x'(p_j)$ . The disadvantage of this process is that the  $x$  values of some points might be modified.

**Axis Perturbation.** The idea is to rotate the axis coordinate system such that no two points have the same  $x$  values. The disadvantage of this way is that the rotation is a relatively time-costly operation.

**Algorithm Modification.** The idea is to modify the algorithm such that the assumption of general position is removed. We developed such an modification which is easy to implement, thus we chose this option in our implementation.

In the next section we describe where in LER indeed the general position assumption is crucial, and how to modify the algorithm to remove it.

## 2 Modifying the Algorithm

Orlowski defined a restricted rectangle (RR for short) as an empty rectangle which is bounded in four directions by either points or bounding box edges. Notice that the LER must be an RR. Otherwise there is at least one direction in which the RR may be stretched and remain empty (thus introducing a larger RR). The algorithm traverses all RR's (there are  $O(n^2)$  such rectangles) and finds the largest among them. In order to do that, the algorithm introduces three phases in which three kinds of RR's are traversed. In the first one all RR's whose two opposite edges are bounded by edges of  $B$  (we denote this kind of RR's by  $R_1$ ) are tested. In the second phase, all other RR's that are bounded by at least one edge of  $B$  (we denote this kind of RR's by  $R_2$ ) are tested. In the third phase all other RR's (we denote this kind of RR's by  $R_3$ ) are tested. An example of the three kinds are illustrated in Figure 2. The algorithm uses several sorted lists of points, ordered both in ascending and descending orders. These sorted lists are the case in which the general position assumption matters. Since the algorithm assumes general position, it does not make the order of two points with the same  $x$  value. Our solution is to order two points with the same  $x$  values by their  $y$  values, either in ascending or descending order (notice that if they have the same  $y$  value too, then we can regard them as one point). In what follows if we chose an ascending order, we denote the sorting by  $S_{\uparrow}$ , and if we chose a descending one, we denote the sorting by  $S_{\downarrow}$ . Theorem 1 proves that by using  $S_{\uparrow}$ ,  $S_{\downarrow}$  and the analogous sorts along the  $y$  axis, the algorithm remains accurate and gives correct results even if there are points in general position. We implemented the algorithm accordingly. A similar technique can be found at Chapter 5.5 of [1].

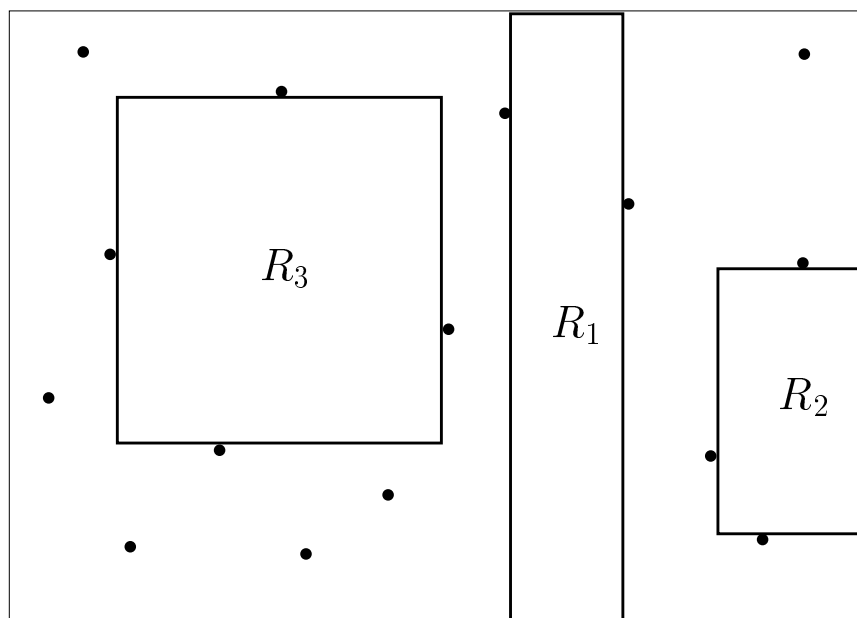


Figure 2: Example of three RR's, one for each kind

The next theorem proves that our modifications gives accurate and robust results. This theorem also summarizes our modification. In order to understand the proofs, the user must be

familiar with the algorithm.

**Theorem 2.1** *Given a set of input points, the Largest Empty Rectangle algorithm with our modifications gives accurate and robust results in the sense that it allows points in general position. Our modification involves the changing of the various sorts as explained above.*

**Proof:** Since the algorithm traverses RR's and find the largest among them, all we need to do is to prove that all RR's are traversed. We show for each phase how each RR is traversed, even if it is restricted by several points in general position.

**First Phase.** In this phase, the algorithm traverses RR's of kind  $R_1$ . We describe the traversal of RR's whose top and bottom edges laying on  $B$ . The traversal of RR's whose left and right edges laying on  $B$  is analogous. We use  $S_{\uparrow}$ . Suppose that an RR,  $a$ , is restricted by several points with the same  $x$  on its left. We get that  $a$  is traversed with the point with the biggest  $y$  value among them. On the other hand, if  $a$  is restricted by several points with the same  $x$  value on its right, it is be traversed with the point with the smallest  $y$  value among them. For example, consider Figure 3. The RR is traversed with the points  $r$  and  $t$ .

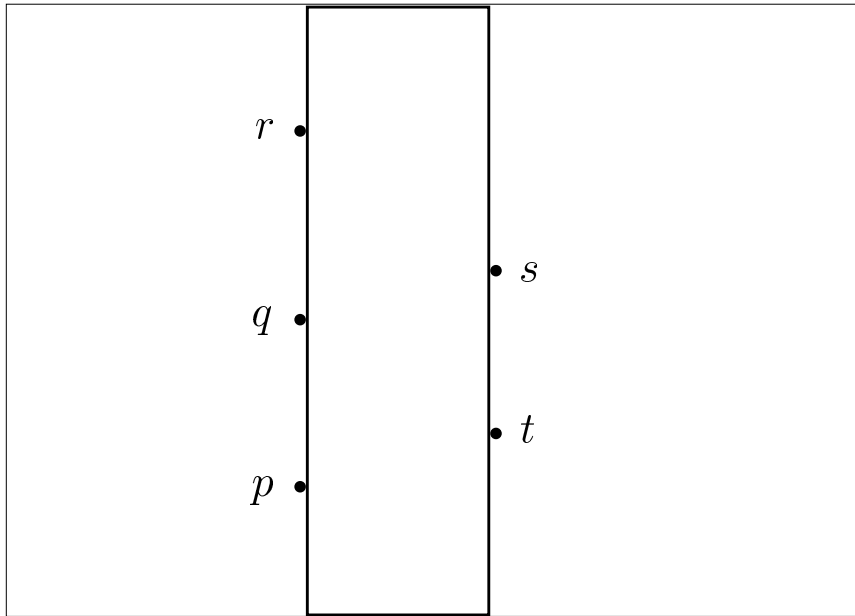


Figure 3: General position points restricting an RR of kind  $R_1$

**Second Phase.** In this phase, the algorithm traverses RR's of kind  $R_2$ . We describe the traversal of RR's whose bottom edges laying on  $B$ . The traversals of other RR's of this kind are analogous. We use  $S_{\uparrow}$ . Suppose an RR,  $a$ , is restricted by several points on its left, top and right edges. We get that  $a$  is traversed with the following three points: the point with the biggest  $y$  value on its left edge, the point with the smallest  $x$  value on its top edge and the point with

the smallest  $y$  value on its right edge. Note that points on its top edge corners are regarded as belonging to its right and left edges. For example, consider Figure 4. The RR is traversed with  $r$ ,  $s$  and  $v$  as the restricting points.

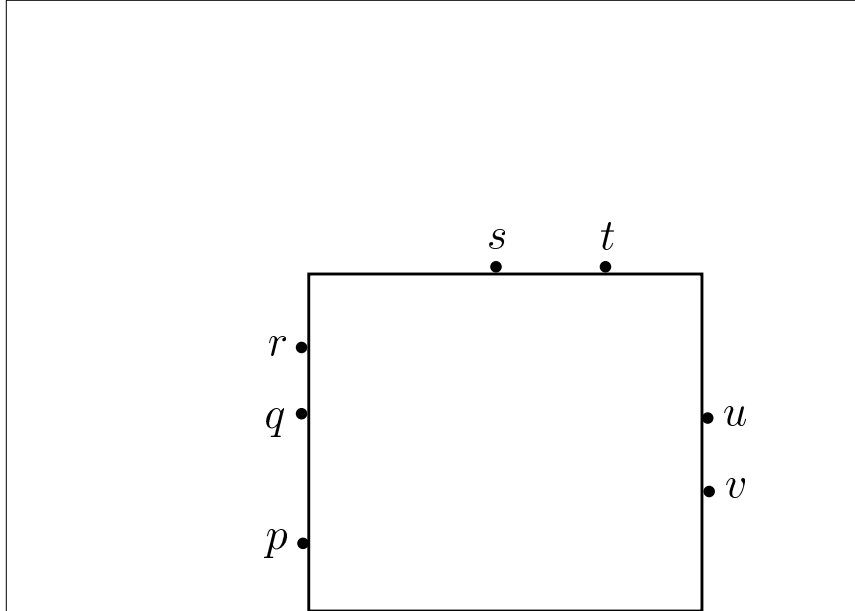


Figure 4: General position points restricting an RR of kind  $R_2$

**Third Phase.** In this phase, the algorithm traverses RR's of kind  $R_3$ . Suppose an RR,  $a$ , is restricted by several points on each of its edges. In the third phase, the points are traversed such that for each one, all the RR's that are restricted by the point from below are traversed. Thus each point on  $a$ 's bottom edge traverses  $a$  once. Let  $b$  be a point on  $a$ 's bottom edge. we order the points on  $b$ 's left tent by ascending  $y$ , and if two points have same  $y$  values, by an ascending  $x$ . On the other hand, we order the points on  $b$ 's right tent by ascending  $y$  and if two points have same  $y$  values, by a descending  $x$ . We get that  $a$  is traversed once for each point on its bottom edge and each traversal is with the following points: the point with the biggest  $y$  value on its left edge, the point with the smallest  $x$  value on its top edge, and the point with the smallest  $y$  value on its right edge. Note that points on corners of  $a$  are regarded as belonging to its right and left edges. For example, consider Figure 5. The RR is traversed once with  $p$ ,  $s$ ,  $t$  and  $w$  and once with  $q$ ,  $s$ ,  $t$  and  $w$  as the restricting points.

□

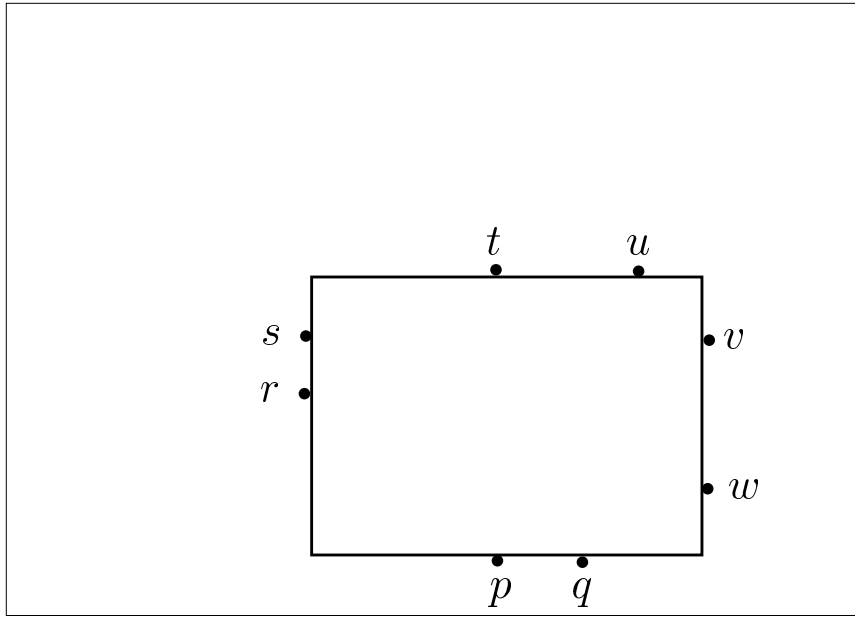


Figure 5: General position points restricting an RR of kind  $R_3$

### 3 Conclusions

We described in general the algorithm behind the Largest Empty Rectangle package. The algorithm, as described in [2], assumes that the input points are in general position. We devised a way to relax this assumption in our implementation. By that we achieved a more general and still robust implementation.

### Acknowledgement

The author thanks Andreas Fabri for both CGAL conformance, writing CGAL documentation, comments on the implementation and many helpful advises. The author also thanks Dan Halperin for helpful advises, and CGAL team people in Tel Aviv University for the help in the implementation.

### References

- [1] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Heidelberg, Germany, 1997.
- [2] M. Orlowski. A new algorithm for the largest empty rectangle problem. *Algorithmica*, 5:65–73, 1990.