# AN EFFICIENT ALGORITHM FOR THE MAXIMUM EMPTY RECTANGLE PROBLEM IN THREE DIMENSIONS

(Extended Abstract)

Amitava Datta and Kamala Krithivasan
Department of Computer Science and Engineering
Indian Institute of Technology, Madras 600 036, INDIA.
email : kamala@shiva.ernet.in

## 1. INTRODUCTION

The Maximum Empty Rectangle (MER) problem is the following. Given a bounding isothetic rectangle BR and a point set P ($|P|=n$) inside it, we have to find the maximum area/perimeter isothetic rectangle R such that R is completely contained in BR and it does not contain any point from the set P. This problem has been studied extensively in two dimensions [1,2,3]. The best algorithm runs in $O(nlog^2n)$ time for the area problem and in $O(nlogn)$ time for the perimeter problem using $O(n)$ space [1]. In this paper, we present an efficient algorithm for the MER problem in three dimensions. It enumerates all three dimensional empty rectangles and finds the maximum one. The time and space complexities of our algorithm are $O(n^4)$ and $O(n^2logn)$ in the worst case. We show that the number of empty rectangles in three dimensions is $O(n^4)$ and any algorithm which enumerates all of them cannot perform better. No previous algorithm was known for this problem. The rest of this abstract is organised as follows. In section 2 we explain some definitions. We present the algorithm in section 3.

## 2. DEFINITIONS

The point set consists of n points $\{p_1,p_2,...,p_n\}$. The three coordinates of point $p_i$ are represented by $p_i.x$, $p_i.y$ and $p_i.z$. The bounding rectangle for the set P is denoted as BR. The two extreme sides for the x direction are written as BR.x.top and BR.x.bottom. Similarly, we define other two pairs of extreme sides in the directions y and z. A rectangle R is called restricted if all the six planes of R pass through either a point from the set P or are aligned with one of the extreme planes of BR. A restricted rectangle (RR) should not contain any point from the set P. We call the maximum

empty rectangle in three dimensions as the Maximum Empty Hyper Rectangle (MEHR). It is easy to see that the MEHR is a member of the set of RRs.

## 3. THE ALGORITHM

We solve this problem by enumerating all possible RRs.

**Lemma 3.1** *There may be $O(n^4)$ RRs in the worst case.*

**Proof.** Fixing two pairs of points on two pairs of opposite planes completely specifies a RR. There may be $O(n^4)$ such points. **Q.E.D.**

We sweep a plane parallel to the x-y plane, i.e., perpendicular to the z axis. The sweep starts at the top z plane of BR,i.e., at BR.z.top and successively other points with decreasing z coordinates are reached. We maintain a data structure such that when a point $p_j$ is reached during the sweep, we can report all RRs with $p_j$ as the bottom support in the z direction. The top support in the z direction for such RRs will be a point $p_i$ such that $p_i.z > p_j.z$. There will be RRs with BR.z.top as the top support and similarly RRs with BR.z.bottom as the bottom support in the z direction. These RRs are also enumerated by our data structures but we will omit the details of enumeration of such RRs in this version. Our basic algorithm is the following :

```
for j:=1 to n do
 begin
    Compute the volumes of all RRs with p_j as bottom support;
    Update the variables containing the MEHR if necessary;
    Update the data structure.
 end.
```

At the end of this computation, we get the MEHR. Our data structure is a combination of segment and interval trees and similar to the one used for direct dominance problem in [4]. For each point $p_i$, we define four quadrants in the following way. We draw lines parallel to the x and y axis. The intervals corresponding to these lines will be [BR.x.top, BR.x.bottom] and [BR.y.top, Br.y.bottom]. We call the four quadrants as the NE,SE,SW and NW quadrants according to the four compass directions. We call a RR which has $p_i$ as the top support as $RR(p_i)$. The points which can be the supports of the side planes (i.e., supports in the x and y direction) for a $RR(p_i)$ form a three dimensional staircase structure in each quadrant. The sweep plane intersects these three dimensional structures at each event point $p_j$ ($p_j.z > p_i.z$) and forms four two dimensional staircases (Figure 1). The points which bound these two dimensional staircases are directly dominated by $p_i$ in three dimensions. For details, see [4]. We associate four staircases with each point $p_i$. These are stored in arrays denoted by

$NW_i$, $SW_i$, $NE_i$ and $SE_i$ respectively. In addition to the four arrays mentioned above, we store all the staircases for all the points in another data structure. This is necessary to find quickly whether a point $p_j$ falls within any of the staircases of a particular point $p_i$. If $p_j$ does not fall within any of the staircases of $p_i$, there cannot be any RR with $p_i$ as top and $p_j$ as bottom supports in the z direction. For this second data structure, we use a combination of segment and interval trees and store the staircases as a collection of rectangles as in [4]. A staircase is represented by as many rectangles as it has treads (Figure 2). Each rectangle has one end point at the point $p_i$ (we call it the distinguished end point) and the opposite corner at one of the treads. So, each rectangle is uniquely identified with a point $p_i$ (the distinguished end point). The y intervals of these rectangles are stored in a segment tree and the corresponding x intervals are stored as interval trees which are associated with each internal node of the segment tree. Again we refer to [4] for details. Now, to find out in which rectangle $p_j$ falls, we have to do a two dimensional range searching which is decomposed into two one dimensional searches in our data structure. We do this search for two purposes. Firstly, to know in which staircases $p_j$ lies and hence can form RRs as the bottom support in the z direction. Secondly, we want to update our two data structures, i.e., the arrays which store the staircases and the segment-interval tree data structure. This is necessary because, after crossing the point $p_j$, the shape of the staircases change. First we describe the reporting of RRs.

### 3.1. Reporting of Restricted Rectangles.

Suppose, during the search we find that $p_j$ lies within a rectangle $R_k$ of which the distinguished corner is $p_i$. We assume without loss of generality that $R_k$ is a part of the SW staircase of $p_i$. We assume that the arrays $SW_i$, $NW_i$, $NE_i$ and $SE_i$ (which store the four staircases of $p_i$) have the highest y coordinate point as the first entry and the y coordinate decreases in the succeeding entries. We will drop the subscript of the arrays when it is understood that they are associated with $p_i$. At the sweep plane, we get a rectangular cross section (RCS) of the RRs of which $p_i$ and $p_j$ form the top and bottom supports respectively. This RCS has four supports from the points of the four staircases. There are sixteen different types of possible RCSs depending on the supports. In this version we consider only one type.

(1) Left and bottom supports from SW, right and top support from NE staircase.

We describe the method for enumerating the type 1 RCSs. The others can be found in using the same method.

#### 3.1.1. Enumeration of type 1 RCS.

It has two supports in the SW and two in the NE staircases. We first note the following facts (Figure 3).

**Observation 1.** *The left and bottom supports should be SW[j] and SW[j+1] i.e., two consecutive elements of the SW array. Similarly, the top and right supports should be NE[k] and NE[k+1].*

**Observation 2.** *$SW[j].x<p_j.x$ and $SW[j+1].y<p_j.y$.*

**Observation 3.** *Suppose, SW[j] and SW[j+1] are the first pair (from the bottom of the SW staircase) which form a RCS with the first pair NE[k] and NE[k+1] (from the top of the NE staircase). The pair SW[j-1] and SW[j] cannot form a RCS with any pair NE[i],NE[i+1] such that $i<k$.*

Observation 3 suggests that we can monotonically go up the SW staircase and come down the NE staircase while reporting the RCSs and hence RRs. The first pair SW[j],SW[j+1] can be found in $O(logn)$ time by a binary search in the SW array. After that no extra time is spent other than enumeration of RRs. So, the enumeration of all RRs with $p_j$ as the bottom support can be done in $O(nlogn+K)$ time, where K is the number of RRs with $p_j$ as the bottom support. Similar monotonicity conditions can be derived for other classes of RCSs. All RRs with points from the set P as top and bottom supports in the z direction can be enumerated in
$O(n^2logn+K)$ time and hence in $O(n^4)$ time in the worst case during the whole plane sweep process.

**3.2. Modification of data structures.** When a point $p_j$ falls within the staircase structure of a point $p_i$ ($p_i.z<p_j.z$), the staircases have to be modified. We again assume for simplicity that $p_j$ falls within the SW staircase of the point $p_i$. We delete from the segment-interval tree structure all the rectangles of the SW staircase which has the point $p_j$ inside them. Two new rectangles are inserted. Suppose, $R_i$ is the rectangle which contains $p_j$ inside it and its left side ($R_i$.left) has the minimum x coordinate among all such rectangles. We insert a rectangle $R_k$ such that $R_k$.left is flush with $R_i$.left and $R_k$.bottom passes through $p_j$. One corner of $R_k$ is the point $p_i$. Similarly, $R_j$ is the rectangle containing $p_j$ and $R_j$.bottom has the minimum y coordinate among all such rectangles. We insert a rectangle $R_m$ such that $R_m$.left passes through $p_j$ and $R_m$.bottom is flush with $R_j$.bottom (Figure 4). Similar modifications in the array SW can be done easily. It is clear that each rectangle is inserted in the segment-interval tree structure at most once and deleted once. Each insertion and deletion takes $O(log^2n)$ time. The number of such rectangles is the number of direct dominance pairs in the set P [4] which is $O(n^2)$. So the overall time requirement for modification of data structures is $O(n^2log^2n)$. Since, at a time at most $O(n^2)$ rectangles exist in the segment-interval tree structure, the worst case space requirement is $O(n^2logn)$.

**Theorem 1.** *The MEHR problem can be solved in $O(n^2 \log^2 n + K)$ time, i.e., $O(n^4)$ time in the worst case using $O(n^2 \log n)$ space.*

## REFERENCES

[1] A.Aggarwal and S.Suri, "Fast Algorithms for Computing the Largest Empty Rectangle", *Proc. of the Third Annual ACM Symposium on Computational Geometry*, (1987), pp.278-290.

[2] B.M.Chazelle, R.L.Drysdale and D.T.Lee, "Computing the Largest Empty Rectangle", *SIAM J. Computing*, **15**, No.1 (1986), pp.300-315.

[3] A.Datta, "Efficient Algorithms for the Largest Rectangle Problem", *Information Sciences*, to appear.

[4] R.H.Guting, O.Nurmi and T.Ottmann, "Fast Algorithms for Direct Enclosures and Direct Dominances", *Journal of Algorithms*, **10**,(1989),pp.170-186.

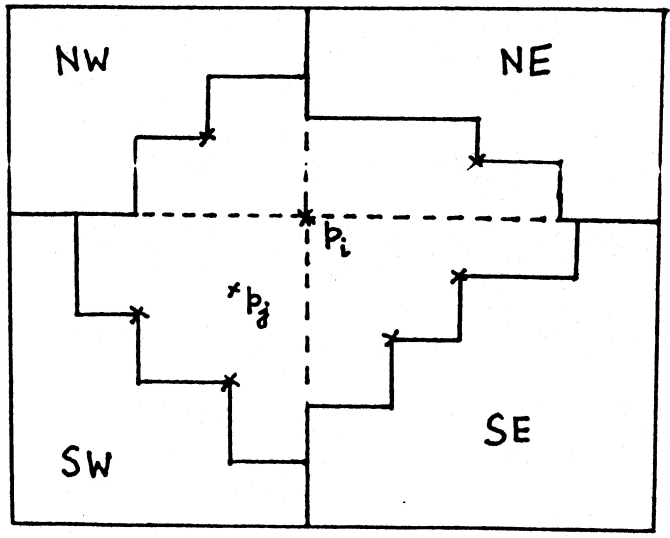


**Figure 1.** The intersection of the sweep plane with the three dimensional staircases. The four staircases for the point $p_i$ are shown here. $p_j.z<p_i.z$.

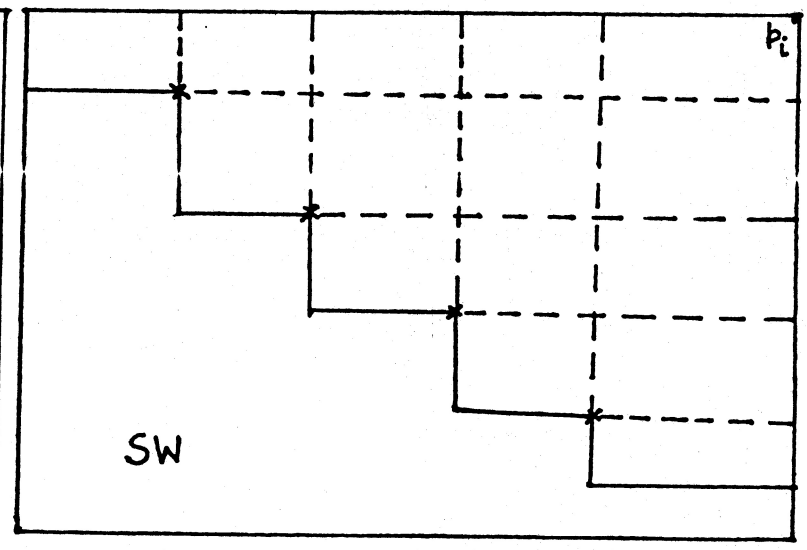


**Figure 2.** The SW staircase is represented as a collection of rectangles. One corner of each rectangle is the point $p_i$.

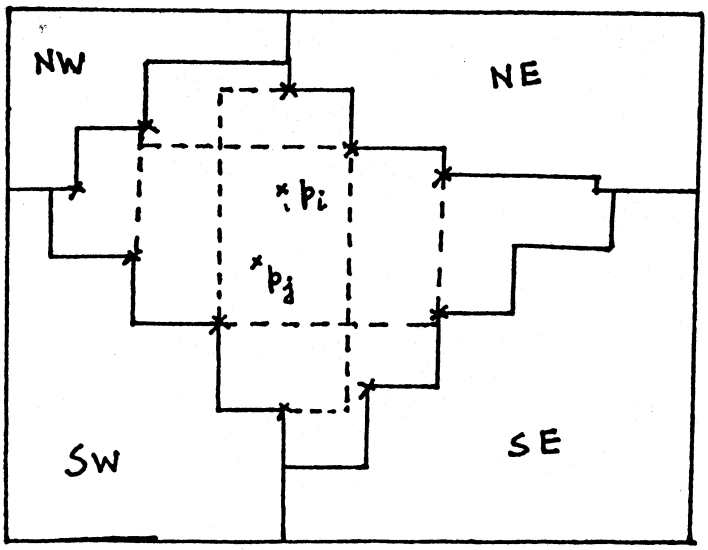


**Figure 3.** Formation of RCSs in the sweep plane. The left and bottom supports are from the SW and the right and top supports are from the NE staircase. All the points are projected on the sweep plane containing $p_j$.

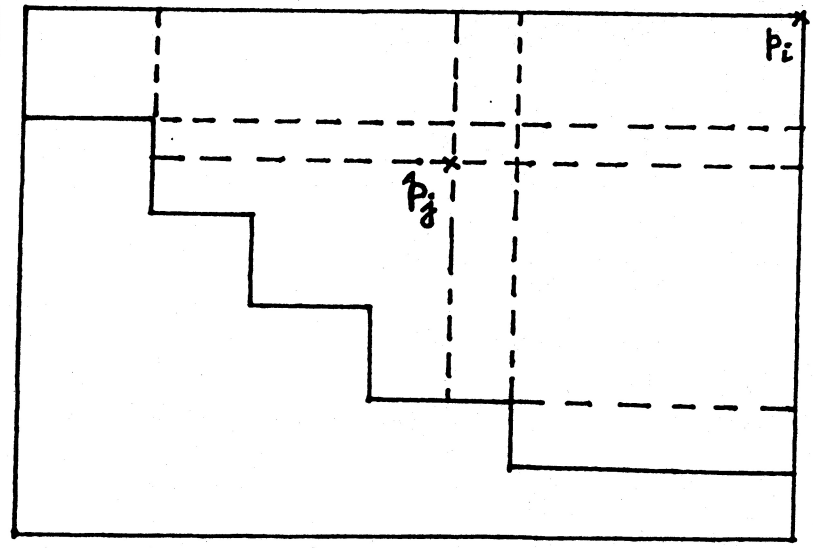


**Figure 4.** The modification of the SW staircase due to the point $p_j$. Two new rectangles are introduced. All rectangles in the SW staircase containing $p_j$ are removed from the segment-interval tree structure.