Efficient Algorithms for Identifying A Pair of Maximal Empty Rectangles of Maximum Total Area Amidst a Point Set

Aniruddha Biswas

Efficient Algorithms for Identifying A Pair of Maximal Empty Rectangles of Maximum Total Area Amidst a Point Set.

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Aniruddha Biswas
[Roll No: CS-1509]

under the guidance of

Dr. Subhas C. Nandy
Professor
Advanced Computing and Microelectronics Unit

Indian Statistical Institute
Kolkata-700108, India

July 2016

To my family and my guide

CERTIFICATE

This is to certify that the dissertation entitled **"Efficient Algorithms for Identifying A Pair of Maximal Empty Rectangles of Maximum Total Area Amidst a Point Set"** submitted by **Aniruddha Biswas** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Subhas C. Nandy
Professor,
Advanced Computing and Microelectronics Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgments

I would like to show my highest gratitude to my advisor, *Prof. Subhas C. Nandy*, Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, for his guidance and continuous support and encouragement. He has literally taught me how to do good research, and motivated me with great insights and innovative ideas.

I would also like to thank *Dr. Sasanka Roy*, Assistant Professor, Indian Statistical Institute, Kolkata, for his valuable suggestions and discussions.

My deepest thanks to all the teachers of Indian Statistical Institute, for their valuable suggestions and discussions which added an important dimension to my research work.

Finally, I am very much thankful to my parents and family for their everlasting supports.

Last but not the least, I would like to thank all of my friends for their help and support. I thank all those, whom I have missed out from the above list.

Aniruddha Biswas
Indian Statistical Institute
Kolkata - 700108 , India.

Abstract

Given an isothetic rectangular floor containing a set of n randomly placed points, an important problem that arises in computational geometry is to identify the largest maximal empty rectangle(MER). The best known algorithm for this problem has complexity of $O(n \log^2 n)$. Here, our motivation is to develop an algorithm for a given set of n points on a rectangular floor, identify a pair of maximal empty rectangles such that the area of their union is maximum among all pairs of maximal empty rectangles that exist on the floor. We have observed that if we try to find two disjoint MER whose union is maximum, the by line sweep technique it can be solved in $O(R + n \log n)$ time, where R is the maximum number of MER's. We have also observed that the unconstrained version of this problem can be solved in less time if it is possible to devise a sub quadratic algorithm for the problem of finding the pair of rectangles whose union of area is maximum among m rectangles. In order to solve this problem we have observed that if we try to find two disjoint rectangles whose union is maximum among m rectangles, it can be solved in $O(m \log m)$ time. We have also observed that if they are axis-parallel squares rather than rectangles, and they are of same area, then even for the non-disjoint case, the problem can be solved in $O(m \log m)$ time.We have approached to solve the same problem for rectangles with same area using matrix space, but have not found any way to approach the general case. This study is still in process.

Keywords: *Computational geometry, geometric algorithms, complexity, balanced binary search tree.*

Contents

Chapter 1

Introduction

1.1 Introduction

Computational geometry involving isothetic polygons, particularly the geometric intersection problem [8] plays a significant role in VLSI, e.g., layout design , design rule checking, circuit extraction , resizing , routing and testing and also in image processing and pattern recognition. An isothetic polygon is a polygon which is formed by isothetic(straight) lines parallel to X or Y axis [21].

Given an isothetic rectangular floor containing a set of isothetic non-overlapping solid rectangles, an important problem that arises in VLSI layout design is to identify a suitable empty rectangular space where a solid rectangle with a given area and aspect ratio is to be placed. This problem arises in VLSI placement [15] where some rectangular components are already placed on a chip, and a new component is to be placed; the objective is to search for a region where the new module can be accommodated without any overlap with the existing solid rectangles. This calls for an algorithm to identify the maximum empty rectangle in the layout area. This is a generalization of the problem discussed by Naamad et.al [14], where the goal was to find the maximum empty rectangle on a Cartesian plane amidst a set of randomly placed points, for which an $O(min(n^2, Rlogn))$ algorithm was described, where n is the number of points on the floor plan and R is the total number of maximal-empty-rectangles(MER) present on the floor. The complexity was later improved to $O(nlog^2n + R)$ in [10] and then to $O(nlogn + R)$ in [[3], [18], [5]]. The algorithms given in [20], [4] locate the maximum-area-rectangle in a point set, without inspecting all MER's in time $O(nlog^3n)$ and $O(nlog^2n)$ respectively. The algorithm developed in [14] was used in [15] to find an efficient strategy of automatic/interactive general block placement. A related problem is to locate a largest rectangle contained in an isothetic polygon [21] for which an $O(n^2)$ time and $O(nlogn)$ space algorithm is known. A similar problem is to find the Largest Axis-Aligned Rectangle in a Polygon, [9] for which an $O(nlogn)$ time algorithm is also known. A unified algorithm has been developed which is applicable to MER for the following three cases : (i) in a collection of solid rectangles, (ii) in an ensemble of points and (iii) within an isothetic polygon. [17] This algorithm was to generate all MER's is suggested which runs in O(nlogn + R) time and consumes O(n) space.Later the problem of identifying all Maximal Empty Isothetic Cuboids (MEC) was introduced In [16]. An output-sensitive algorithm, based on plane-sweep paradigm, is proposed for generating all the MECs present on the floor. The algorithm runs in $O(C + n^2 logn)$ time in the worst case, and requires O(n) space, where C is the number of MECs present inside the box. In this paper it was also shown that the total number of MECs is bounded by $O(n^3)$ in the worst case. It was conjectured by Datta and Soundaralakshmi [17] that the maximum number of maximal empty boxes is $O(n^d)$ for each (fixed) d. The conjecture was confirmed by Kaplan et. al. [11], who obtain the first tight $\theta(n^d)$ bound on this number.Recently it Dumitrescu and Jiang has showed that the expected number of maximal empty axis-parallel boxes amidst n random points in the unit hypercube $[0, 1]^d$ in R^d is $(1 \pm o(1)) \frac{(2d-2)!}{(d-1)!} \, n \, ln^{d-1} n$, if d is fixed.

Here, our motivation is to develop an algorithm for a given set of points on a rectangular floor, identify a pair of maximal empty rectangles such that the area of their union is maximum among all pairs of empty rectangles that exist on the floor.

Chapter 2

Algorithm For The Point Set

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ be n points on a rectangular region $\mathcal{A}$. The upper left corner of $\mathcal{A}$ is assumed to be the origin of our co-ordinate system with x and y. An Axis parallel(isothetic) rectangle R in a given layout that does not contain any point of P is said to be an empty rectangle. An empty rectangle is said to be a maximal empty rectangle(MER) no other empty rectangle contains R.

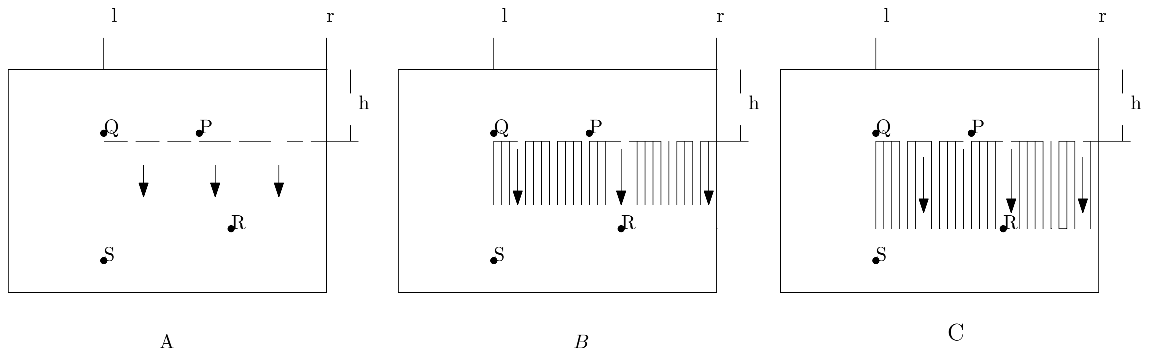
We first consider a simpler variation of the problem, where the objective is to report two disjoint empty rectangles of maximum area. Next, we consider the general problem, where the reported rectangles may be non-disjoint

2.1 Disjoint pair of MER of maximum area.

We use line sweep technique and the algorithm in [17] to report two disjoint MER of maximum area. Before Describing it let summarize the algorithm in [17]. Here they have given an algorithm to identify all possible MER in a given point set. An for that they have introduced the concept of a window. Let us introduce the concept of window as follows:

Consider a point p(a_1,h). Draw a horizontal line through the point p. It hits the left and right boundary at the points $l_1(l, h)$ and $r_1(r, h)$ respectively. A primary window is said to be generated in this event and is represented by an ordered tuple $([l, r], h)$, where $[l, r]$ is a horizontal interval and h denotes the height where it was originated (see Figure 2.1).

Now a window $S([l, r], h)$ which remains active till it strikes a point $R(a_1, b_1)$. Before it dies down, it returns an MER$[(l, h), (r, b_1)]$ and splits itself to give birth to two new windows whose intervals are subsumed by the interval $[l, r]$ of the parent window S and whose heights are inherited. These windows are called secondary windows, which in turn, may give birth to other secondary windows(see Figure 2.2).



(A): Generation of a new window $([l, r], h)$ while processing the bottom of a point P at height h.

(B) A curtain coinciding with the interval $[l, r]$ continues to fall from height h.

(C): The window remains alive till the falling curtain first hits a point $R(a1, b1)$. The corresponding MER $[(l, h), (r, b1)]$ is identified.

Figure 2.1: Processing the bottom of a point P

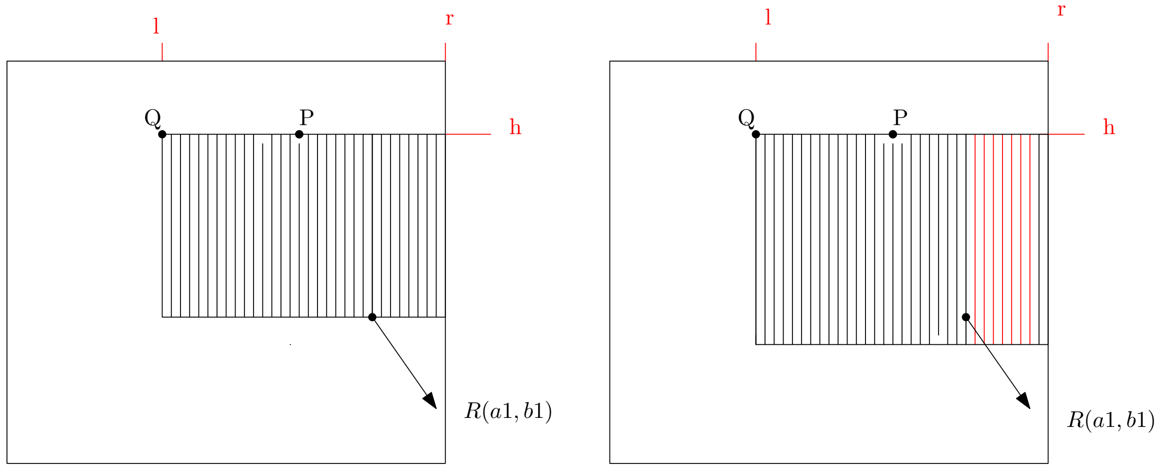


Figure 2.2: Two windows $([l, a_1], h)$ and $([a_1, r], h)$ are generated while S disappears.

We apply a line sweep method to compute the largest MER among the points above each horizontal line passing through the points in P. While sweeping line L from the top boundary of $\mathcal{A}$, whenever a point(p) is reached, some MER has been generated by the above algorithm. We then compute the maximum one among them and store it in variable T(p) attached to the point p. The same method is followed to compute the largest MER among points below each horizontal line passing through the points in P.Here also we attach a variable B(p) with the point p, to store the largest MER below the point p. Next , a linear scan is performed; at each point we add T(p) and B(p) to identify the point p for which this sum is maximum.

2.1.1 Data Structure

Our algorithm needs the following data structures:
(A) A list Y, with three fields Y.x, Y.y, Y.T, Y.B;

1. Y.x contains the x co-ordinate of each point $p \in P$.
2. Y.y contains the y co-ordinate of each point $p \in P$.
3. Y.T contains the largest MER among the points above each horizontal line passing through the points for each point $p \in P$.
4. Y.B contains the largest MER among the points below each horizontal line passing through the points for each point $p \in P$.

Initially Y.T an Y.B this two list are assigned to zero.

(B) A balanced binary search tree where each of its leaf node from left to right contains the x co-ordinate of the points in P sorted in ascending order. Each of the internal node contains the following information:

(1) A discriminant value $d(w) = (d(w_1) + d(w_2))/2$, where w_1 and w_2 are the left and the right child of w respectively. The discriminant value of a leaf node is the x co-ordinate attached to it.

(2) A secondary list w.L . It's each node contains windows containing p, and it consists of three fields L.l, L.r and L.h, sorted in increasing order with respect to L.h, is attached to each node w of T in the form of a doubly linked list with an additional direct forward link from w to the last node in the list.All the windows[say A([l,r],h)] which contains d(w) i.e. $l \leq d(w) \leq r$ are in this secondary list.

We also maintain a global variable M. It keeps the area of the largest MER above each point considered so far.

We also maintain two pointers $POINTER\_L$ and $POINTER\_R$ while processing a point p(a_1,b_1). Initially, these pointers point to the root of T. Later, $POINTER\_L$

points the current node of T which contains the last inserted secondary window with right extremity at a_1 and $POINTER\_R$ points the current node containing the last inserted secondary window with left extremity at a_1.

List Y is sorted in non-decreasing order on the basis of the value of Y.y . If points have same y co-ordinate then they are kept in Y in increasing order with respect to their x co-ordinates.

2.1.2 Algorithm

Input: A rectangular floor [(0,0),(a,b)] with n points.

Output: Return the list P.T

Step1 : Initially an empty balanced binary search tree T is prepared.

Step2 : A window ([0,a],0) corresponding to the roof of the floor plan[(0,0),(a,b)], is inserted in T.

Step3 : Create the sorted list Y,/*This gives the sequence of processing*/

Step4 : Let p(a_1,b_1) be the current point which is to be processed.Search the balanced binary search tree T to identify the node w where d(w) = a_1[x co-ordinate of the point p].Let P_{IN} be the set of nodes traversed from the root to w.

Step5 : Find all windows in w.L and in P_{IN} whose intervals contains p. Let this set of windows be A. This set of windows are processed from the end of the secondary list of their respective node(i.e., the window with the largest interval first and then in decreasing order). Each of the window will generate an MER. We will store the area of generated MER in p.T. Every time we will update the value of p.T, if the newly generated MER has larger area than the previous one. When the processing of all the windows in w.L is over; we will store $MAX(p.T, M)$ in P.T .

Step6 : After reporting each MER, the corresponding window will be split. One of the splitting part will still remain a valid window(Assume no two points have same x or y co-ordinate). The surviving window say A[(l_1,r_1),h_1] needs to be inserted in the secondary list of the proper node; only if there is no window say B[(l_2,r_2),h_2] is present where ($l_1 = l_2$) and ($r_1 = r_2$). This whole process can be efficiently done by the use of $POINTER\_L$ and $POINTER\_R$.

Step8 : Insert the window C[(0,a),b_1] at the proper node in T.

Step7 : Delete all the processed windows from T.

Step8 : Repeat from Step4 until all points are processed.

Now run this algorithm in the opposite direction i.e. Bottom to Top direction, to find out the list P.B . Now, for each point $p \in P$ we have these two value p.T (represents the largest MER among all MER whose bottom end is on or above p) and p.B (represents the the largest MER among all MER whose upper end is on or below p). So, now return the maximum value of the summation of p.T and p.B, among all the points in P.

Now we will run this algorithm in X direction to find out the best possible disjoint pairs which are separated only by vertical line. Now compare and return the best disjoint pair.

2.1.3 Complexity Analysis

1. The construction of the balanced binary search tree including the sorting of Y can be done $O(n \log n)$ time.
2. Step4 can be done in $O(\log n)$ time. Step4 will run for n times as we have total n number of points. So, overall time taken by Step4 is $O(n \log n)$.
3. Step5 will take $O(|A|)$ time to complete. $|A|$ is the number of MER generated during the processing of one point. Now Step5 will run for each of the point. So, overall time taken by Step5 is: $\sum\limits_{p \in P} |A| = R$. R is the total number of MER.
4. In step6 all secondary windows generated from set A can be inserted in T in $O(\log n) + |A|)$ time by properly using $POINTER\_L$ and $POINTER\_R$. So, overall time taken by step6 is $O(n \log n + R)$.

So, overall time complexity to find the list P.T is $O(n \log n + R)$. An to find the best disjoint pair we are doing this similar method for 4 times and doing some linear scan which will not affect the total complexity. Naamad et. al. [14] shows that the total number of MER(i.e. R) can be of $O(n^2)$. So, the worst case complexity of our algorithm is also $O(n^2)$.

2.2 Non disjoint pair of MER of maximum area

Now to find the pair of Non-Disjoint MER whose union of area is maximum we can't use the sweep line technique. So, we have attempted to solve it in some different way. Naamad et. al. [14] shows that the total number of MER is of $O(n^2)$, so if we try to find the pair among all $O(n^2)$ MER, then our ob is done. So, the most trivial way is to do the Brute Force search. So, we will check all possible pair and will return the pair whose area of union is maximum. But for that we need to check $(n^2(n^2 + 1)/2)$ number of pairs; which is of $O(n^4)$. So, now if it is possible to find some algorithm where it is possible to find out such pairs among all $O(n^2)$ MER in some efficient technique then our problem is solved.

So, now our main problem boils down into a different problem. We have a set of n **axis parallel** rectangles. We need to identify a pair of rectangles whose union is of maximum area. The trivial algorithm of considering all possible pairs needs $O(n^2)$ time. The objective is to have a sub quadratic algorithm. We have discussed it in the next chapter.

Chapter 3

Best possible pair among n given axis parallel rectangles

3.1 Best disjoint pair of rectangles

We again use the sweep line algorithm to solve this problem in $O(n \log n)$ time.(see Figure 3.1). We execute a left to right sweep to compute the largest rectangle among all rectangle that lie to the left side of the right boundary of each rectangle. A similar right to left sweep is performed to compute the largest rectangle among all rectangle that lie to the right side of the left boundary of each rectangle.We now explain the left to right sweep.

Here we maintain a list P with two fields P.m and P.x; where $P_r.m$ contains the largest rectangle among all rectangle that lie to the left side of the right boundary of r and $P_r.x$ contains the x co-ordinate of the right boundary of r for each $r \in R$.

Here we maintain a global variable MAX. It keeps the area of the largest rectangle on the left of the right side of each rectangle considered so far.

Input: A set $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ of n rectangles.

Output: Return the list P.

Step1: Sort the rectangles on the basis of their right boundary.

Step2: Move a sweep line from left to right direction where the event of the sweep line is the right side of the rectangle.

Step3: Let at an instant we are processing the right boundary of rectangle r. compare its area with MAX; and store the maximum one in $P_r.m$ and x co-ordinate of right boundary of r in $P_r.x$ for each $r \in R$.

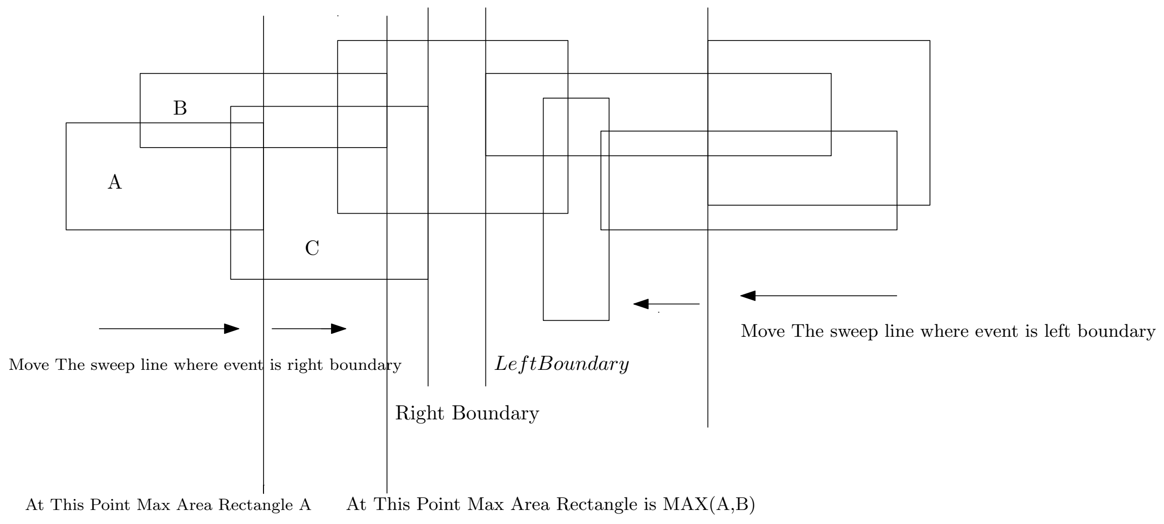

B
A
C
Move The sweep line where event is right boundary
LeftBoundary
Right Boundary
Move The sweep line where event is left boundary
At This Point Max Area Rectangle A
At This Point Max Area Rectangle is MAX(A,B)

Figure 3.1: Demonstration of the algorithm for finding disjoint pair of rectangles of maximum area

Step4: Repeat from the step2 until the right sides of all rectangles are processed.

Now we need to do the right to left sweep in similar manner. And we store the result in a list Q with two fields Q.m and Q.x; where $Q_r.m$ contains the largest rectangle among all rectangle that lie to the right side of the left boundary of r and $Q_r.x$ contains the x co-ordinate of the left boundary of r for each $r \in R$.

Now to find the disjoint pair of rectangles of maximum area we will do as follows:

1. Initialize M(a scalar variable) to zero.

2. Keep two pointers at the beginning of both the list P and Q.

3. For an index i in P find the first index j from Q, where $P_i.x \leq Q_j.x$; then add $P_i.m$ and $P_j.m$ and store it in M if it is larger than the earlier value of M.

3. Repeat 2 until all elements of list P are processed.

Now as both the list P and Q are sorted we can do the above process linearly. So, in overall the most costly step is the sorting of the rectangles, which will take $O(n \log n)$ time.

Now repeat this algorithm in y direction also to find out best the disjoint pair of rectangles which are only separated by a horizontal line.

3.2 Unconstrained best pair of rectangles

In this unconstrained version, we first execute the algorithm of 3.1 too identify a disjoint pair of rectangles of maximum size. Then we need to device an algorithm to identify a pair of possibly intersecting rectangles of maximum size. This study is still in process. We considered a simplified variation of this problem where all the objects are axis parallel squares of same size in the next chapter.

Chapter 4

Best possible pair of squares among n given axis parallel squares of unit size

It is clear that in this problem if there exists at least one disjoint pair of square then that should be our answer. So, first we will run our algorithm in 3.1 for this set of squares. If the solution pair is of size 2 unit in total, then our job is done. Otherwise, if all of the n squares are are intersecting with each other then algorithm in 3.1 will return only one square in the answer.

Lemma 1 *Let $S_1, \ldots S_n$ be a collection of axis parallel squares. If the intersection of every pair of these squares is nonempty, then the whole collection has a nonempty intersection; that is,*
$\bigcap_{j=1}^{n} S_j \neq \varnothing$.

Proof:
As every pair of these given squares are non empty and all of the squares are axis parallel squares, from geometry it is clear that collection of any three squares also have a non empty intersection. Then rest of the lemma follows from Helly's Theorem. [1]
$\square$

So, we consider an environment where all the squares have a common intersection region.(see Figure 4.1).

4.1 Notation

Let us denote $X\_INT(A, B)$ as the length of intersection area between square A and B in the X-direction, $Y\_INT(A, B)$ as the length of intersection area between square

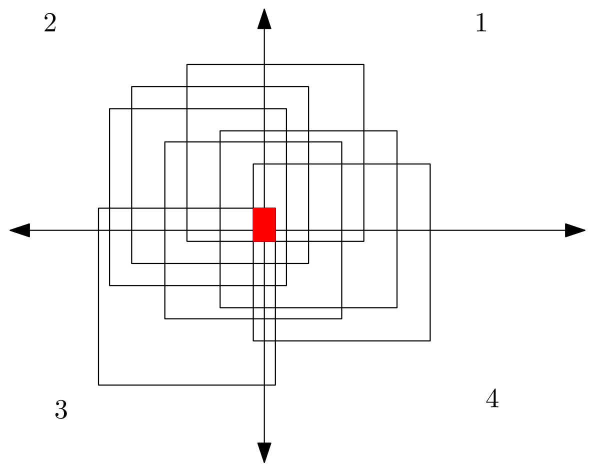


Figure 4.1: Shaded region is the intersection area of all the squares

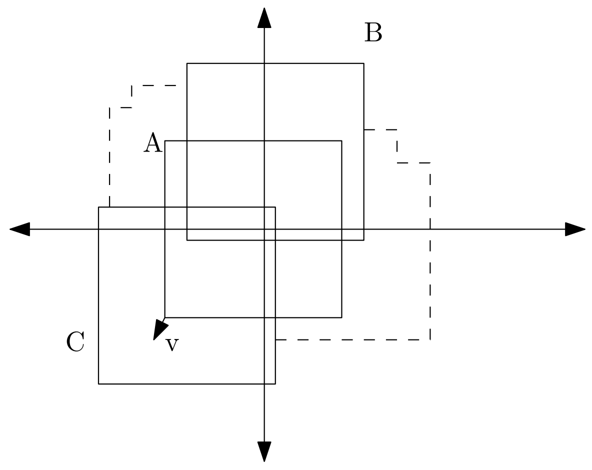


Figure 4.2: No vertices of Square A are in the boundary

A and B in the Y-direction, INT(A,B) is the intersection area between square A and B, and U(A,B) is the union of area of square A and B.

4.2 Observations and their proofs

Observation 1 *Let $S_1, \dots S_n$ be a collection of axis parallel unit squares and $\bigcup_{j=1}^{n} S_j = U$. All those squares which has at least one vertex in the boundary of the region U only they can be present in the solution of our problem.*

Proof:
For contradiction, let it is a possible to get the best possible pair of square where at least one of the square does not have any vertex at the boundary. Let A be such a

square[see Figure 4.2] and suppose union(A,B) is maximum. Now, as no vertex of A(say v) is not in the boundary so, there must be some square (Suppose C) whose left boundary is on the left of A and bottom boundary is on the bottom of A to cover the vertex v. So, $(X\_INT(A,B) > X\_INT(C,B))$ and $(Y\_INT(A,B) > Y\_INT(C,B))$ also. So, $(INT(A,B) > INT(C,B))$ which implies that $(U(C,B) > U(A,B))$ which is a contradiction. □

As of now we will only consider all those squares which has at least one vertex at the boundary. As all the squares are sharing a common area(shaded in Figure 4.1), we can choose our axes passing through this shaded area. This splits $\mathbb{R}^2$ into four quadrants. All the border vertices of the squares will fall into any one of these four quadrant.

Now our first task is to to find out all the vertices of the squares which are at the boundary of region U. It is possible to detect in $O(n \log n)$ time.[see page 340 in [19]]. Now to detect the quadrant of each boundary vertex we need to find the intersection region of all squares. It is possible to to detect by sorting of squares in both x and y direction in $O(n \log n)$ time.

Note That algorithm in [19] will return only the endpoints of each line segment of the hull U. Let this set of point is R, and the set of vertices of all squares is V. We will give number to only those points which are in the intersection of these two sets R and V. Numbering of the vertices will be needed later.

A square having its one corner in any quadrant i is said to lie in quadrant i. Observe that a square may lie in more than one quadrants.

Observation 2 *If in the optimal solution both the squares lie in the same quadrant then their vertices appear respectively at the bottom end and top most points of the stair in that quadrant.*

Proof:
Obvious.See Figure 4.3, here in the 3^{rd} quadrant the square corresponding to the vertex 'i' and the square corresponding to the vertex 'l' will be the best pair. □

So, in order to identify such a solution, if exists, in O(1) time by computing the area of the union of these two squares and storing them as a candidate for the best pair.

•We denote extreme endpoint for a horizontally adjacent quadrant are left most stair vertex in the left quadrant and right most stair vertex in the right quadrant and for a vertically adjacent quadrant are top most stair vertex in the upper quadrant and bottom most stair vertex in the lower quadrant.

Observation 3 *If in the optimal solution both the squares lie in adjacent quadrant then at least one of them will be the square corresponding to the extreme endpoint as defined above.*

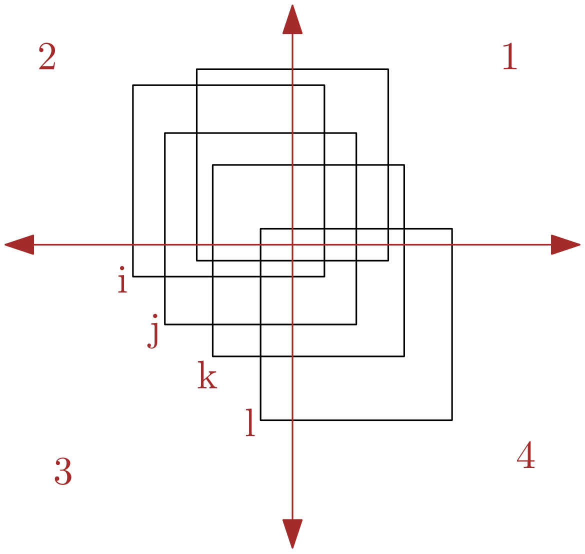


Figure 4.3: In the 3^{rd} quadrant square corresponding to the vertex 'i' and the square corresponding to the vertex 'l' will be the best pair

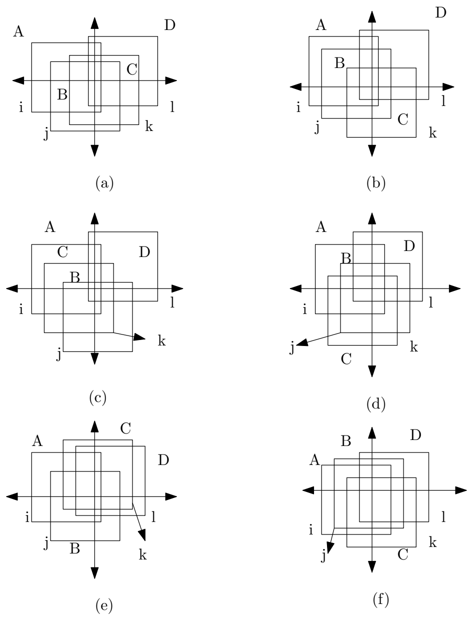


Figure 4.4: Among all the squares which has a stair vertex in the 3^{rd} or in the 4^{th} quadrant, either A or D will be in the optimal solution pair.

Proof:

Let the optimal solution lie in 3^{rd} and in 4^{th} quadrant. For contradiction, let the best pair (B,C) in Figure 4.4.a or in Figure 4.4.b corresponds to two squares whose corresponding stair vertices i.e. j and k are in 3^{rd} and 4^{th} quadrant respectively such that no one corresponds to an extreme endpoint of its respective stair. Let i and l are two extreme endpoints and corresponding squares are A and D respectively. WLOG let us assume that D is on top of A.

Now it is clear that the square B will always at the left of the square C. If not suppose C is at the left of B. Then we have two possibility either C is at the top of B or C is at the bottom of B. But both these cases are not possible; because in the first case k will no more a stair vertex[see Figure 4.4.c], and in the second case j will no more a stair vertex[see Figure 4.4.d].

Now as we have already assumed that i and l are two extreme border vertex in 3^{rd} and 4^{th} quadrant, so B can't be at the left of A and C can't be at the right of D and from Figure 4.4.e and 4.4.f it is clear that C will always on the bottom of D, o.w. k will no more a stair vertex and B will always on the bottom of A, o.w. j will no more a stair vertex.

Hence, C will be at the right of B and on the bottom and left of D along with that B is at the right and bottom of A.[see Figure 4.4.a or Figure 4.4.b]

Now we have two cases:

Case1: C is at the top B. In this case $(X\_INT(B,C) > X\_INT(B,D))$ (as C is at right of B and D is right of C) and also $(Y\_INT(B,C) > Y\_INT(B,D))$ (as C is at top of B and D is top of C). So, $(INT(B,C) > INT(B,D))$. So, $U(B,C) < U(B,D)$; which is a contradiction.[see Figure 4.4.a]

case2: C is at the bottom of B. In this case $(X\_INT(C,A) < X\_INT(C,B))$ (as B is at left of C and A is left of B) and also $(Y\_INT(C,A) < Y\_INT(C,B))$ (as B is at top of C and A is top of B). So, $(INT(B,C) > INT(A,C))$. So, $U(B,C) < U(A,C)$; which is a contradiction.[see Figure 4.4.b]

□

Now we deal with the problem of identifying two vertices v_1 and v_2 of region U such that they lie in diagonally opposite quadrant, and their corresponding squares S_1 and S_2 produces maximum area of $S_1 \cup S_2$ among all possible pair of vertices in diagonally opposite quadrants.

We number the vertices of region U with the natural numbers from 1 to n in clockwise order. We use Q_i to denote the vertices of the region U in i^{th} quadrant, for i = 1,2,3,4. Consider the vertices in bottom-left and top-right quadrants. Let A_{ij} denote the area of $S_1 \cap S_2$ where $v_i \in$Q$_3$ and $v_j \in$Q$_1$. Let $|Q_3| = m$ and $|Q_1| = n$.

Observation 4 *Let $M_{m\times n}$ be a matrix whose ij^{th} entry M_{ij} corresponds to $S_1 \cap S_2$ where $v_i \in Q_3$ and $v_j \in Q_1$. The matrix $\mathcal{M}$ is a totally monotone matrix as defined*

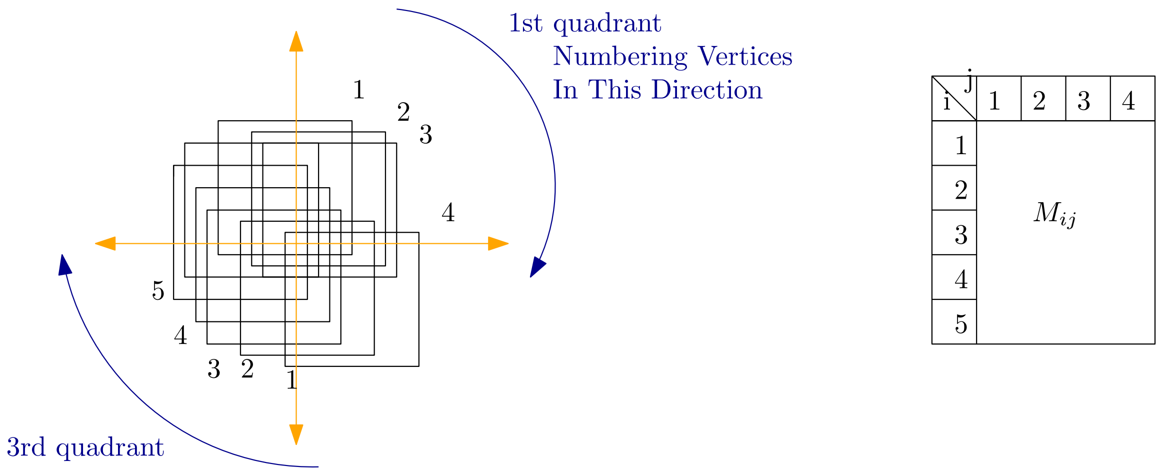


Figure 4.5: Demonstration of observation 4

below.

Definition 1 *A matrix is said to be totally monotone if for every* 2×2 *sub matrix* $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$*, it is not simultaneously possible that* $a > b$ *and* $c < d$*.*

Proof:
[Observation 3]: To prove this matrix M totally monotone we will take any 2×2 sub matrix, $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, and we will show that it is not simultaneously possible that $a > b$ and $c < d$.

[See Figure 4.6.a]Let i,j are any two row index of matrix M i.e. the stair vertex in Q_3 and k,l are any two column index of matrix M i.e. the stair vertex in Q_1. Now, 'a' is the intersection area between the squares corresponding to the stair vertex i and k; similarly b,c,d is defined.

Let S_i, S_j, S_k, S_l are the squares corresponding to the stair vertex i(x_i,y_i),j(x_j,y_j), k(x_k,y_k),l(x_l,y_l) respectively.

Now from the way of the numbering of the stair vertices we can conclude two conditions:

1. $(x_i > x_j) and (y_i < y_j)$
2. $(x_l > x_k) and (y_l < y_k)$

Case1: i and l are the vertices of the same square and j and k are the vertices of the same square. Here $a = M[i,k] < b = M[i,l]$, so monotonicity holds.[see Figure 4.6.b]

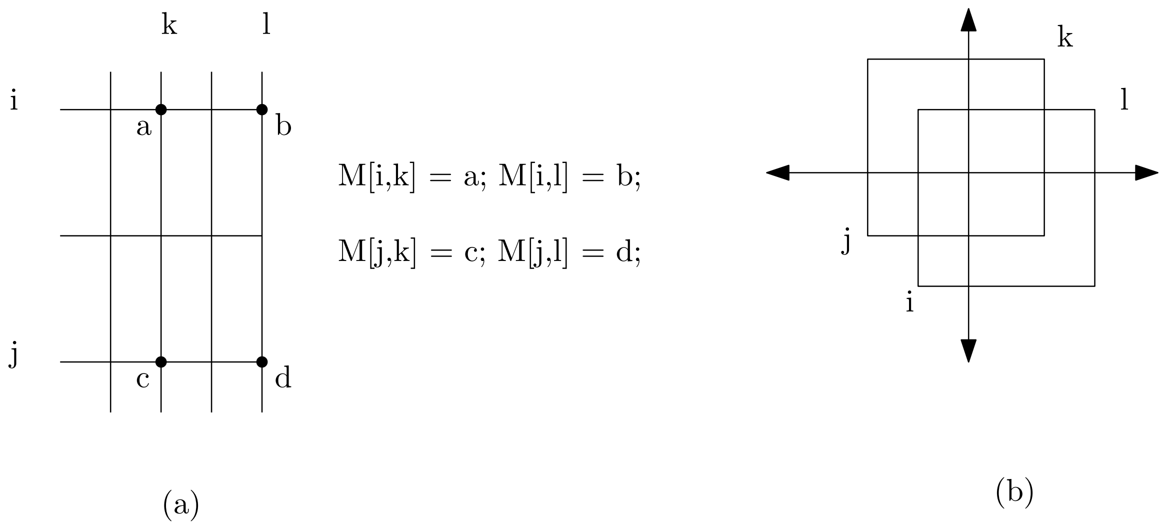


Figure 4.6: (a): An instance of a matrix M , (b): i and l are the vertices of the same square and j and k are the vertices of the same square.

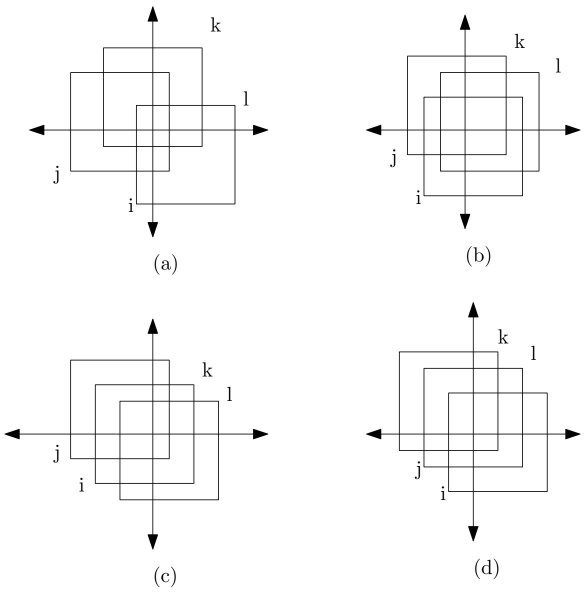


Figure 4.7: Two of the stair vertices are of same square

Case2: i and k are the vertices of the same square and j and l are the vertices of the same square. By construction it is not possible.

Case3: i and l are the vertices of the same square and j and k are the vertices of the different square. Here $a = M[i,k] < b = M[i,l]$,so monotonicity holds.[see Figure 4.7.a]

Case4: i and l are the vertices of the different square and j and k are the vertices of the same square. Here $(c = M[j,k] > d = M[j,l])$,so monotonicity holds.[see Figure 4.7.b]

Case5: i and k are the vertices of the same square and j and l are the vertices of different square.[see Figure 4.7.c]. Here S_j is to the left and above of S_k (as S_i and S_k are same square),so $(X\_INT(S_j, S_k) > X\_INT(S_j, S_l))$ and $(Y\_INT(S_j, S_k) > Y\_INT(S_j, S_l))$. So, $(INT(S_j, S_k) > INT(S_j, S_l))$ i.e. (c > d) always.

Case6: j and l are the vertices of the same square and i and k are the vertices of different square. [see Figure 4.7.d]. Here S_i is at the right and bottom of S_l (as S_j and S_l are same square),so $(X\_INT(S_i, S_k) < X\_INT(S_i, S_l))$ and $(Y\_INT(S_i, S_k) < Y\_INT(S_i, S_l))$. So, $(INT(S_i, S_k) < INT(S_i, S_l))$ i.e. (a < b.) always.

Case7: All S_i, S_j, S_k, S_l are different squares. Here we prove the totally monotone matrix property exhaustively enumerating all possible cases.Since $(x_i > x_j) and (x_l > x_k)$ we sort the squares S_i, S_j, S_k, S_l in increasing order on the basis of their left boundary. They may appear in six ways as demonstrated in Figure 4.1. Similarly we sort the squares in increasing order on the basis of their bottom boundary. They also may appear in six ways.Considering the pair of ordering of four participating squares S_i, S_j, S_k, S_l namely(Ai : Bj),i=1, . . . , 6,j=1, . . . , 6 we have 36 configuration. We will consider each configuration and will either justify that the said configuration is not possible, or if such a configuration is possible then show that the monotonicity holds.

- Among these 36 configurations there are 16 which are invalid.[see Figure 4.8]

(A4:B4): Here S_l is at left and bottom of S_i; So border vertex i will not exist.

(A4:B5): Same reasoning as before.

(A4:B6): Same reasoning as before.

(A5:B6): Here S_k is at left and bottom of S_i; So border vertex i will not exist.

(A1:B1): Here S_k is at left and bottom of S_j; So border vertex j will not exist.

(A1:B5): Same reasoning as before.

(A1:B6): Same reasoning as before.

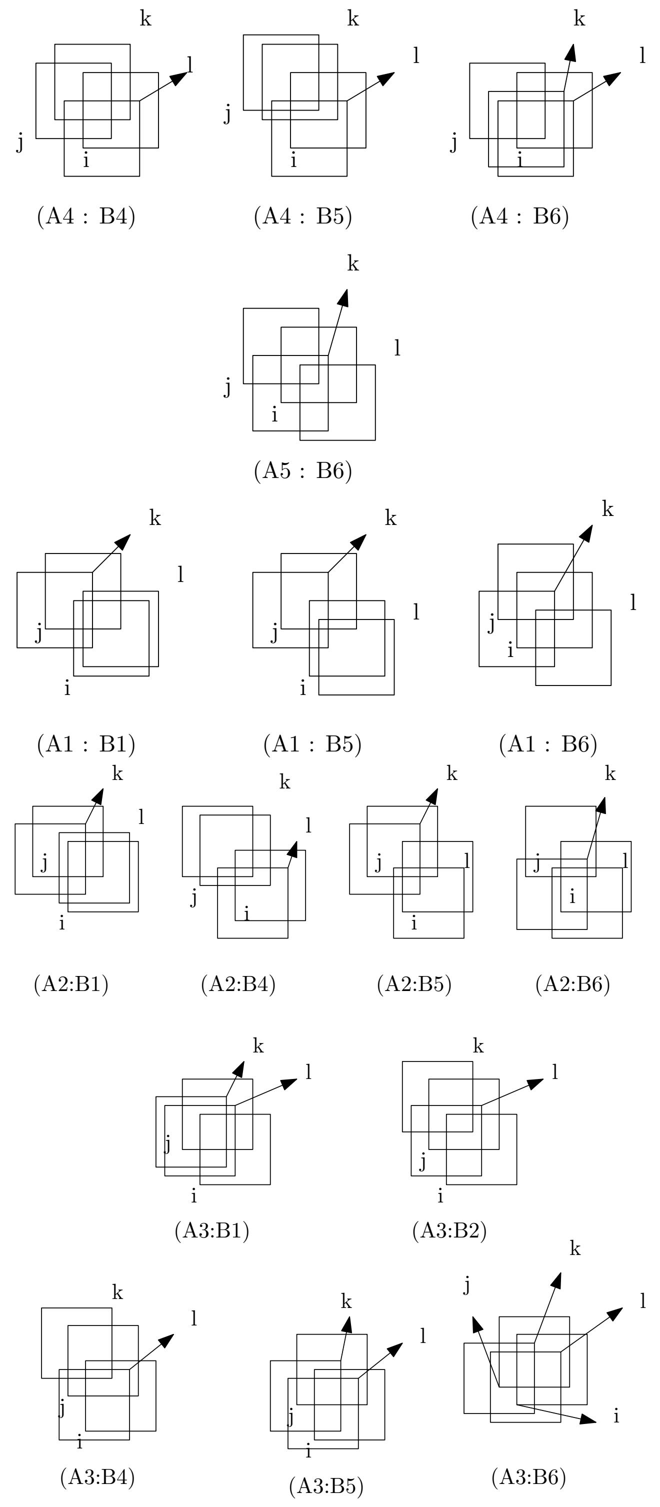


Figure 4.8: Invalid configurations

Table 4.1

All possible combinations if we sort S_i, S_j, S_k, S_l in increasing order on the basis of its left boundary:	All possible combinations if we sort S_i, S_j, S_k, S_l in increasing order on the basis of its bottom boundary:
A1: S_k, S_j, S_i, S_l	B1: S_i, S_l, S_k, S_j
A2: S_k, S_j, S_l, S_i	B2: S_i, S_l, S_j, S_k
A3: S_k, S_l, S_j, S_i	B3: S_i, S_j, S_l, S_k
A4: S_j, S_k, S_l, S_i	B4: S_l, S_i, S_j, S_k
A5: S_j, S_k, S_i, S_l	B5: S_l, S_i, S_k, S_j
A6: S_j, S_i, S_k, S_l	B6: S_l, S_k, S_i, S_j

(A2:B1): Same reasoning as before.

(A2:B5): Same reasoning as before.

(A2:B6): Same reasoning as before.

(A2:B4): Here S_l is at left and bottom of S_i; So border vertex i will not exist.

(A3:B1): Here S_k and S_l both are at left and bottom of S_j; So border vertex j will not exist.

(A3:B2): Here S_l is at left and bottom of S_j; So border vertex j will not exist.

(A3:B4): Here S_l is at the left and bottom of both S_i and S_j; So both i and j will not exist.

(A3:B5): Same reasoning as before.

(A3:B6): Same reasoning as before.

• Now among these remaining 20 structure there are 6 structures where $INT(S_i, S_k) < INT(S_i, S_l)$ i.e. $a < b$.[see Figure 4.9]

Here, S_l is at the left and above of S_i, and S_k is at the left and above of S_l. So, $X\_INT(S_i, S_k) < X\_INT(S_i, S_l)$ and similarly $Y\_INT(S_i, S_k) < Y\_INT(S_i, S_l)$;Hence $INT(S_i, S_k) < INT(S_i, S_l)$ i.e. $a < b$.

• Next we show that there are 5 structures where $INT(S_j, S_k) > INT(S_j, S_l)$ i.e. $c > d$.[see Figure 4.10]

Here, S_j is at the left and above of S_k, and S_k is at the left and above of S_l. So, $X\_INT(S_j, S_k) > X\_INT(S_j, S_l)$ and similarly $Y\_INT(S_j, S_k) > Y\_INT(S_j, S_l)$;Hence $INT(S_j, S_k) > INT(S_j, S_l)$ i.e. $c > d$.

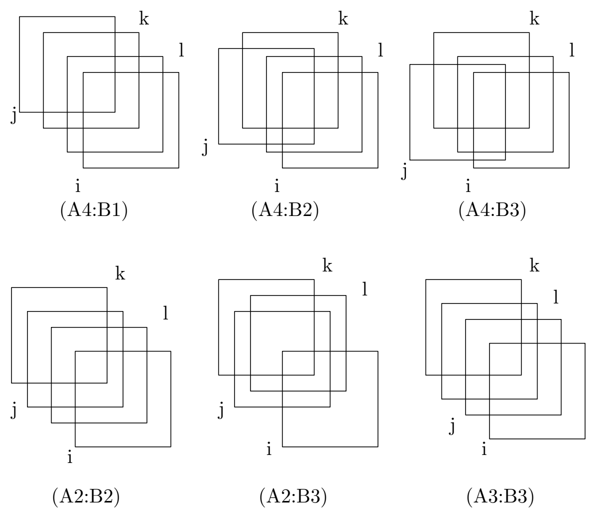


Figure 4.9: Here $INT(S_i, S_k) < INT(S_i, S_l)$ is always true

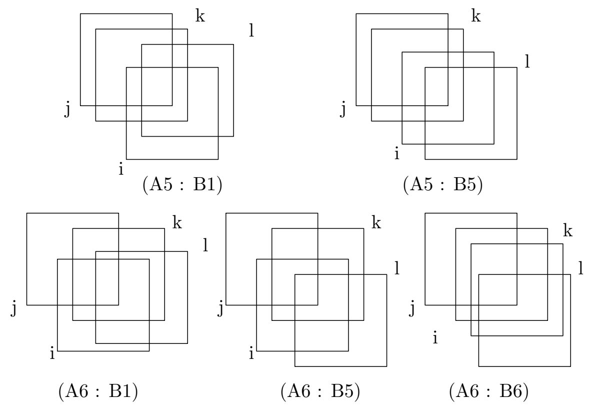


Figure 4.10: Here $INT(S_j, S_k) > INT(S_j, S_l)$ is always true

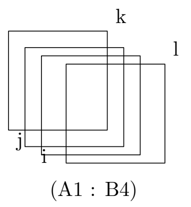


Figure 4.11

• Now consider the case (A1:B4) as shown in Figure 4.11. Here, we can't ensure that $c > d$ or in other words $INT(S_j, S_k) > INT(S_j, S_l)$. But if $a > b$ i.e. $(INT(S_i, S_k) > INT(S_i, S_l))$ then $c > d$ can be ensured as follows:

Let, $INT(S_i, S_k) > INT(S_i, S_l) \ldots (A)$

S_k is to the left and above of S_j, and S_j is to the left and above of S_i :

$\implies INT(S_j, S_k) > INT(S_i, S_k) \ldots (B)$

Together (A) and (B)

$\implies INT(S_j, S_k) > INT(S_i, S_l) \ldots (C)$

Also S_l is to the right and bottom of S_i, and S_i is to the right and bottom of S_j:

$\implies INT(S_i, S_l) > INT(S_j, S_l)$ as $\ldots (D)$

So, from (C) and (D) we can say that,

$INT(S_j, S_k) > INT(S_j, S_l)$ i.e. $c > d$.

•Now we will consider the four cases shown in Figure 4.12. Here if the total monotonicity is not followed, then (i) $a > b$ and (ii)$c < d$ where, a = INT(S_i , S_k), b = INT(S_i , S_l), c = INT(S_j , S_k) and d = INT(S_j , S_l) have to occur simultaneously. From (i) and (ii):

$INT(S_i, S_k) + INT(S_j, S_l) > INT(S_i, S_l) + INT(S_j, S_k)$.

But this is not possible since,

INT(S_i , S_k) $\cup$ INT(S_j ,S_l) $\subseteq$ INT(S_i , S_l) $\cup$ INT(S_j , S_k).

Each of the rest four cases, we will consider individually and show that for all these cases totally monotonicity holds.[see Figure 4.13]

•[Case(A5 : B4)]: [see Figure 4.13.a]

Suppose in this case $INT(S_j, S_k) < INT(S_j, S_l)$ is true.

$\implies (d + b + c) < (c + a)$

$\implies (b + c) < (a + c) \ldots (1)$

Now as S_j is at the left of S_k and S_k is at the left of S_l.

$\implies X\_INT(S_j, S_k) > X\_INT(S_j, S_l)$

So, to satisfy $INT(S_j, S_k) < INT(S_j, S_l)$ it must be $Y\_INT(S_j, S_k) < Y\_INT(S_j, S_l)$

$\implies$ [$Y\_INT(S_j, S_k)\times$ base] < [Y_INT(S$_j$, S_l)$\times$ base]

$\implies (o + p + q) < (q + s)$

$\implies (p + q) < (q + s) \ldots (2)$

Both (1) and (2)

$\implies (b + c + p + q) < (a + c + q + s) \ldots (3)$

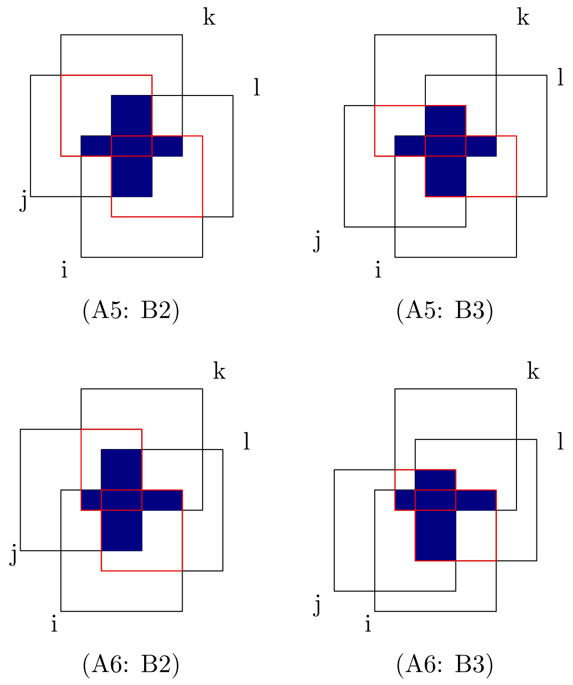


Figure 4.12: Total area of the blue region is completely contained into the total area of 2 rectangles with red boundary

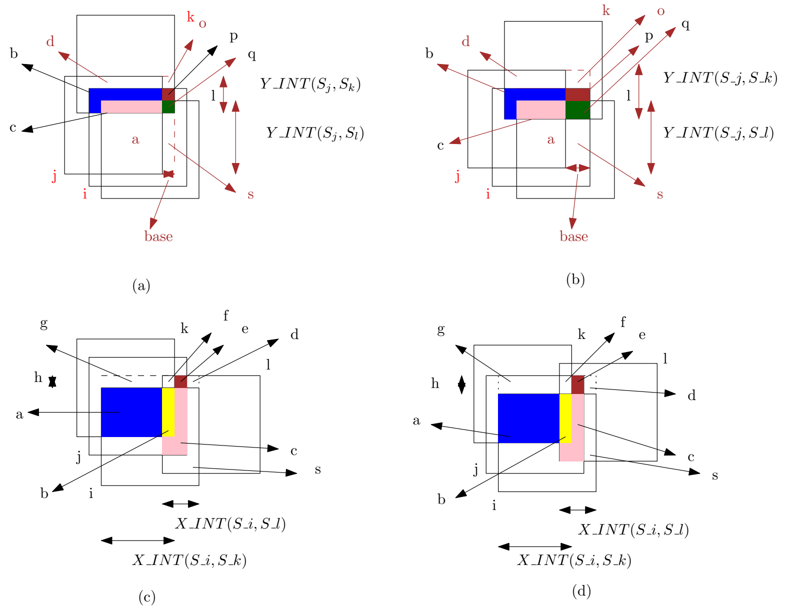


Figure 4.13: In each image one directional arrow is representing the area of the region from where it has been generated, two directional arrow is representing the length.

And from Figure 4.13.a it is clear that:
$INT(S_i, S_l) > (a + c + q + s) \ldots (4)$
And, $INT(S_i, S_k) = (b + c + p + q) \ldots (5)$
Both (4) and (5)
$\implies INT(S_i, S_k) < INT(S_i, S_l)$.
So, here monotonicity holds.

•[Case(A6 : B4)]: Proof technique is same as before[see Figure 4.13.b]

•[Case(A1 : B2)]: [see Figure 4.13.c]
Suppose in this case $INT(S_i, S_k) > INT(S_i, S_l)$ is true.
$\implies (a + b) > (b + c + s)$
$\implies (a + b) > (b + c) \ldots (1)$
Now as S_i is at the bottom of S_l and S_l is at the bottom of S_k.
$\implies Y\_INT(S_i, S_k) < Y\_INT(S_i, S_l)$
So, to satisfy $INT(S_i, S_k) > INT(S_i, S_l)$ it must be $X\_INT(S_i, S_k) > X\_INT(S_i, S_l)$
$\implies$ [$X\_INT(S_i, S_k)\times$ h] > [X_INT(S$_i$, S_l)× h]
$\implies (g + f) > (f + e + d)$
$\implies (g + f) > (f + e) \ldots (2)$
Both (1) and (2)
$\implies (a + b + g + f) > (b + c + f + e) \ldots (3)$
And from Figure 4.13.c it is clear that:
$INT(S_j, S_k) > (a + b + g + f) \ldots (4)$
And, $INT(S_j, S_l) = (b + c + f + e) \ldots (5)$
Both (4) and (5)
$INT(S_j, S_k) > INT(S_j, S_l)$.
So, here monotonicity holds.

•[Case(A1 : B3)]: Proof technique is same as before[see Figure 4.13.d]

□

4.3 Complexity Analysis:

If in the optimal solution both the squares lie in the same quadrant then it is possible to identify such a solution in O(1) time.[From observation 1]. If in the optimal solution both the squares lie in the adjacent quadrant then it is possible to identify such a solution in O(n) time.[From observation 2].If in the optimal solution both the squares lie in the diagonal quadrant then also it is possible to identify such a solution in O(n) time because in O(n) it is possible to identify the minimum value of every row of a totally monotone matrix. [2]. So, the most costly operation in this algorithm is to find the stair vertices and their respective quadrants which is of $O(n \log n)$. So, overall complexity of this algorithm is $O(n \log n)$.

Chapter 5

Future Work and Conclusion

We have developed a sub quadratic algorithm to find the best disjoint pair of axis parallel rectangles. But in the unconstrained version of finding the best pair of rectangles we have not been able to find any sub quadratic algorithm. This study is still in process. We have developed a $O(n \log n)$ algorithm simplified variation of this problem also, where each of the rectangles are squares of same area.

Now we are trying to solve a more difficult variation of this problem where each of the polygons area rectangles but of same area. Here first we have defined one distance function as follows:
$d(R_1, R_2) = A(\cup(R_1, R_2)) - A(R_1)$; where

1. $d(R_1, R_2)$: Distance between two rectangles R_1 and R_2.
2. A($\cup(R_1, R_2)$): Area of the union of rectangles R_1 and R_2.
3. '-' : Subtraction operation between natural numbers.
4. $A(R_1)$: Area of rectangle R_1 which is a constant number 'C'.

Lemma 2 *$d(R_1, R_2)$ is a metric.*

Proof:
To prove this lemma we nee to show that $d(R_1, R_2)$ will follow all three properties of a metric.

1. $d(R_1, R_1) = A(\cup(R_1, R_1)) - A(R_1) = 0$.
2. $d(R_1, R_2) =$ A($\cup(R_1, R_2)$) $- A(R_1) = \cup(R_1, R_2) - A(R_2) = d(R_2, R_1)$, since $A(R_1) = A(R_2) = C$.

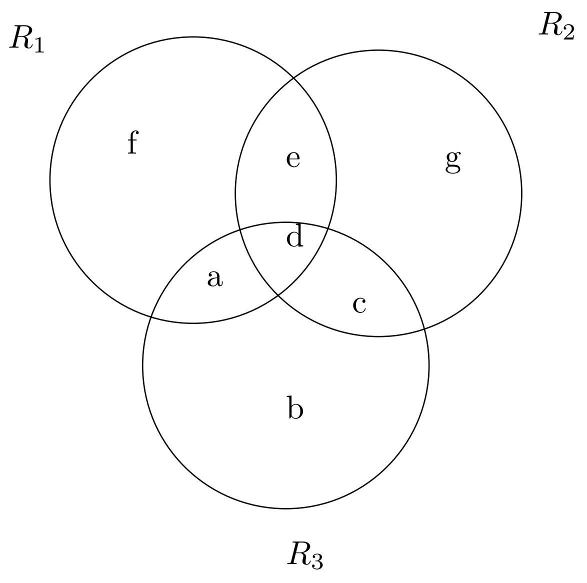


Figure 5.1

3. Triangle inequality:[see Figure 5.1]

   $C = A(R_1) = (a+d+e+f) = A(R_2) = (c+d+e+g) = A(R_3) = (a+d+c+b).$

   $d(R_1, R_2) = (f + e + g + a + d + c) - C \dots (i)$

   $d(R_2, R_3) = (a + d + c + b + e + g) - C \dots (ii)$

   $d(R_1, R_3) = (a + d + e + b + c + f) - C \dots (ii)$

   (i)+(ii)-(iii) $\implies (g + a) \geq 0$

   (i)+(iii)-(ii) $\implies (f + c) \geq 0$

   (ii)+(iii)-(i) $\implies (b + e) \geq 0$

Hence, $d(R_1, R_2)$ is a metric. □

It is clear that optimal solution pair of rectangles will be the farthest pair of points in this new metric space. Our motivation of defining this distance function has from [6] and [7]. Here they have deifned Convex Polygon-Offset Distance Function and with this new distance function they have generated furthest site voronoi diagram for convex polygon. Hence, in our problem if it is possible to represent each rectangle uniquely by some disjoint point or tuples, and as the distance function is also metric then we can generate the furthest site voronoi diagram for the set of rectangles with our distance function [[13], [12]] and we will get our optimal solution pair. We are currently working in this problem and later we will approach for the general case.

Bibliography

[1] https://en.wikipedia.org/wiki/Helly%27s_theorem#cite_note-1

[2] http://asishm.myweb.cs.uwindsor.ca/cs557/F10/handouts/matrixSearching.pdf

[3] Aggarwal, A., Suri, S.: Fast algorithms for computing the largest empty rectangle. In: Proceedings of the third annual symposium on Computational geometry. pp. 278–290. ACM (1987)

[4] Atallah, M.J., Frederickson, G.N.: A note on finding a maximum empty rectangle. Discrete Applied Mathematics 13(1), 87–91 (1986)

[5] Atallah, M.J., Kosaraju, S.R.: An efficient algorithm for max dominance, with applications. Algorithmica 4(1-4), 221–236 (1989)

[6] Barequet, G., De, M.: Voronoi diagram for convex polygonal sites with convex polygon-offset distance function. In: CALDAM. pp. 24–36 (2017)

[7] Barequet, G., Dickerson, M.T., Goodrich, M.T.: Voronoi diagrams for convex polygon-offset distance functions. Discrete & Computational Geometry 25(2), 271–291 (2001)

[8] Bentley, J.L., Ottmann, T.A.: Algorithms for reporting and counting geometric intersections. IEEE Transactions on computers (9), 643–647 (1979)

[9] Boland, R.P., Urrutia, J.: Finding the largest axis-aligned rectangle in a polygon in o(n log n) time. In: In Proc. 13th Canad. Conf. Comput. Geom. pp. 41–44 (2001)

[10] Chazelle, B., Drysdale, R., Lee, D.: Computing the largest empty rectangle. SIAM Journal on Computing 15(1), 300–315 (1986)

[11] Datta, A., Soundaralakshmi, S.: An efficient algorithm for computing the maximum empty rectangle in three dimensions. Information Sciences 128(1), 43–65 (2000)

[12] Klein, R., Mehlhorn, K., Meiser, S.: Randomized incremental construction of abstract voronoi diagrams. Computational Geometry 3(3), 157–184 (1993)

[13] Klein, R., Wood, D.: Voronoi diagrams based on general metrics in the plane. STACS 88 pp. 281–291 (1988)

[14] Naamad, A., Lee, D., Hsu, W.L.: On the maximum empty rectangle problem. Discrete Applied Mathematics 8(3), 267 – 277 (1984), `http://www.sciencedirect.com/science/article/pii/0166218X84901240`

[15] Nandy, S., Bhattacharya, B.: Maximal empty cuboids among points and blocks. Computers Mathematics with Applications 36(3), 11 – 20 (1998), `http://www.sciencedirect.com/science/article/pii/S0898122198001254`

[16] Nandy, S., Bhattacharya, B.: Maximal empty cuboids among points and blocks. Computers Mathematics with Applications 36(3), 11 – 20 (1998), `http://www.sciencedirect.com/science/article/pii/S0898122198001254`

[17] Nandy, S.C., Bhattacharya, B.B., Ray, S.: Efficient algorithms for identifying all maximal isothetic empty rectangles in vlsi layout design. In: Proceedings of the Tenth Conference on Foundations of Software Technology and Theoretical Computer Science. pp. 255–269. FST and TC 10, Springer-Verlag New York, Inc., New York, NY, USA (1990), `http://dl.acm.org/citation.cfm?id=111662.111682`

[18] Orlowski, M.: A new algorithm for the largest empty rectangle problem. Algorithmica 5(1-4), 65–73 (1990)

[19] Preparata, F.P., Shamos, M.: Computational geometry: an introduction. Springer Science & Business Media (2012)

[20] Wimer, S., Koren, I.: Analysis of strategies for constructive general block placement. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 7, 371–377 (Mar 1988)

[21] Wood, D.: "an isothetie view of computational geometry" (1985)