F. Dehne, "A one dimensional systolic array for the largest empty rectangle problem," in Proc. Allerton Conference on Communication, Control and Computing, Monticello, Ill., 1985, pp. 518-524.

A ONE-DIMENSIONAL SYSTOLIC ARRAY FOR THE LARGEST EMPTY RECTANGLE PROBLEM

FRANK DEHNE
Lehrstuhl fuer Informatik I, Univ. of Wuerzburg
Am Hubland, D-8700 Wuerzburg, W.-Germany

ABSTRACT

Given a rectangle with its edges parallel to the coordinate axes containing a set S of n points in 2-dimensional Euclidean space we consider the problem of finding the largest area subrectangle with sides parallel to those of the original rectangle which contains no point of S and describe a one-dimensional systolic array which solves this problem in linear time.

1. INTRODUCTION

Given a rectangle with its edges parallel to the coordinate axes containing a set S of n points in 2-dimensional Euclidean space we consider the problem of finding the largest area subrectangle with sides parallel to those of the original rectangle which contains no point of S. [8] and [3] gave $O(n^2)$ time, linear space, and $O(n \log^3 n)$ time, $O(n \log n)$ space, respectively, algorithms to solve the problem on a sequential computer.
Motivated by [7], [11], and [2] who studied geometric problems from a similar point of view we describe a one-dimensional systolic array (see fig.1) ,called LER, solving the problem in linear time which is asymptotically optimal since any nontrivial computation requires time $\Omega(n)$ on a linear array.

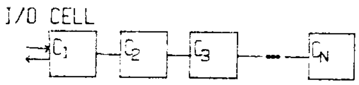


fig.1

LER will support the following operations :
- Insert/delete a point
- Report the largest empty rectangle

Since it consists of N cells, it can handle up to N points at a given time. For more details about one-dimensional systolic arrays consult [2], see also [5]. Note, that for LER all I/O-operations are performed by the leftmost cell C_1, since otherwise our algorithm would run in time linear with respect to N.

2. INSERTION, DELETION

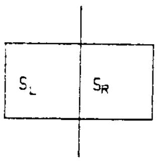
To insert a new point, just put it into LER at its leftmost

I/O-cell and let it move to the right, until it finds an empty cell. To delete a point, send an identifier to LERs I/O-cell C_1 and let it move to the right, until it finds the specified point. Delete this point and send a signal (special record) to its right neighbor , to let the following points shift to the left and close the gap.

3. REPORTING THE LARGEST EMPTY RECTANGLE IN LINEAR TIME

3.1. Basic Structure of Algorithm

Let $S=\{s_1,\ldots,s_n\}$ be the current set of $n\leq N$ points sorted by their x-coordinates (sorting can be done in linear time applying the methods of [10] to a one-dimensional array). Let x_{min}, x_{max}, y_{min}, y_{max} be the boundaries of the bounding rectangle.
Note that each edge of the largest empty rectangle is supported by either an edge of the bounding rectangle or at least one point of S (as described in [3]). We shall call these supporting edges or points "supporting elements with resp. to S".
To simplify exposition, we shall assume, that all points of S have distinct x-coordinates and distinct y-coordinates and do not lie on the boundary. Thus, the largest empty rectangle has exactly four supporting elements with resp. to S. As we shall see at the end of this paper, the existence of some more supporting elements will not change our algorithm significantly.
In order to compute the largest empty rectangle, we split S into two halves $S_L=\{s_1,\ldots,s_{\lfloor n/2\rfloor}\}$ and $S_R=\{s_{\lfloor n/2\rfloor+1},\ldots,s_n\}$ (with their bounding rectangles adjusted) and recursively solve the problem for S_L and S_R using the systolic cells $C_1,\ldots,C_{\lfloor n/2\rfloor}$ and $C_{\lfloor n/2\rfloor+1},\ldots,C_n$, respectively (see fig.2).



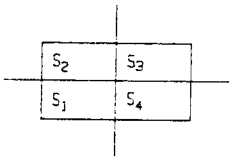


fig.2

fig.3

Given the largest empty rectangles with resp. to S_L and S_R we have to compare the maximum area rectangle of these with the largest empty rectangle having at least one supporting element with resp. to S_L and S_R, respectively.
This "merging step" will be done by a second divide and conquer procedure.
After sorting S by y-coordinates, we split it into four subsets as described by fig.3 with $S_1\cup S_2=S_L$, $S_3\cup S_4=S_R$ and $|\ |S_2\cup S_3| - |S_1\cup S_4|\ | \leq 1$.

With this we recursively compute the largest empty rectangle having at least one supporting element with resp. to S_2 (S_1) and S_3 (S_4) and none with resp. to S_1 (S_2) and S_4 (S_3), repectively.

3.2. Computing the Final Result

In order to compute the final result we have to find the largest empty rectangle r, having the following property (I):

Let B_1 $(B_2,\ldots,B_4)$ be the set of supporting elements of r with resp. to S_1 $(S_2,\ldots,S_4)$, then

$|B_1|+|B_2|+|B_3|+|B_4|=4$
$|B_1|+|B_2|>0$
$|B_3|+|B_4|>0$
$|B_2|+|B_4|>0$
$|B_1|+|B_4|>0$.

Let $S_i' := S_i \cup \{p_i, q_i\}$ $(i=1,\ldots,4)$ as sketched by fig.4 .

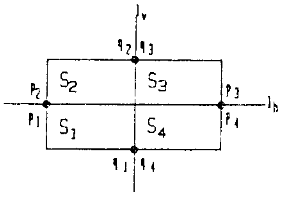


fig.4

Now, we can prove

Lemma 1:
If r is an empty rectangle with property (I) and e is an edge of r supported by a vertical (horizontal) boundary edge of S_i, then e is supported by p_i (q_i) , i=1...4 .

Proof:
From property (I) it is easy to see, that both vertical (horizontal) edges of r have to cross l_h (l_v), thus lemma 1 follows immediatly. ∎

With this we can forget the bounding rectangles simply by adding the points p_i, q_i $(i=1,\ldots,4)$ during the final merging step and considering all empty rectangles with exactly four supporting points and property (I) with resp. to $S_1',\ldots,S_4'$. Note, that we add at most 4n points simultaneously, thus every cell of LER has to store at most 5 points.

Definition 1:

Let (x_1,y_1), (x_2,y_2) be two points, then

$(x_1,y_1) <_{ur} (x_2,y_2)$:<=> $x_1<x_2$ and $y_1<y_2$
$(x_1,y_1) <_{ul} (x_2,y_2)$:<=> $x_1>x_2$ and $y_1<y_2$
$(x_1,y_1) <_{ll} (x_2,y_2)$:<=> $x_1>x_2$ and $y_1>y_2$
$(x_1,y_1) <_{lr} (x_2,y_2)$:<=> $x_1<x_2$ and $y_1>y_2$

Let M be a set of points and $x \in M$, then x is called a ur-maximal [ul-maximal, ll-maximal, lr-maximal] element of M
:<=> x is a maximal element of M with resp. to $<_{ur}$ [$<_{ul}$, $<_{ll}$, $<_{lr}$].

Let M_1 [M_2, M_3, M_4] be the ur [lr,ll,ul] -maximal elements of S_1' [S_2', S_3', S_4'], then we have

<u>Lemma 2:</u>
Let r be an empty rectangle supported by four points $\{t_1,\dots,t_4\} \subseteq S_1' \cup S_2' \cup S_3' \cup S_4'$ with property (I), then $\{t_1,\dots,t_4\} \subseteq M_1 \cup M_2 \cup M_3 \cup M_4$.

Proof:
From property (I) it easy to see, that both vertical (horizontal) edges of r cross l_h (l_v). Since r has to be empty lemma 2 follows immediatly. ∎

Summarizing this, we have

<u>Theorem 1:</u>
In order to compute the final result it is sufficient to find the maximum area rectangle of all empty rectangles supported by four points $\{t_1,\dots,t_4\} \subseteq B_1 \cup B_2 \cup B_3 \cup B_4$ with $B_i \subseteq M_i$ $(i=1,\dots,4)$, $|B_1|+|B_2|+|B_3|+|B_4|=4$, $|B_1|+|B_2|>0$, $|B_3|+|B_4|>0$, $|B_2|+|B_3|>0$ and $|B_1|+|B_4|>0$.

CASE	|B1|		|B2|		|B3|		|B4|		TYPE
1)	0		1	$<$	1	$\wedge$	2	$\vee >$	B
2)	0		1	$<$	2	$\wedge >$	1	$\vee$	B
3)	0		2	$< \wedge$	0		2	$\vee >$	A
4)	0		2	$< \wedge$	1	$>$	1	$\vee$	B
5)	1	$<$	0		1	$\wedge$	2	$\vee >$	B
6)	1	$<$	0		2	$\wedge >$	1	$\vee$	B
7)	1	$<$	1	$\wedge$	0		2	$\vee >$	B
8)	1	$<$	1	$\wedge$	1	$>$	1	$\vee$	C
9)	1	$\vee$	1	$<$	1	$\wedge$	1	$>$	C
10)	1	$\vee$	1	$<$	2	$\wedge >$	0		B
11)	1	$\vee$	2	$< \wedge$	0		1	$>$	B
12)	1	$\vee$	2	$< \wedge$	1	$>$	0		B
13)	2	$< \wedge$	0		1	$>$	1	$\vee$	B
14)	2	$< \vee$	0		2	$\wedge >$	0		A
15)	2	$< \vee$	1	$\wedge$	0		1	$>$	B
16)	2	$< \vee$	1	$\wedge$	1	$>$	0		B

table 1

In table 1 all possible (16) cases are listed with $<$ [$>$, $\wedge$, $\vee$] denoting that a point supports a left [right, upper, lower] edge of an empty rectangle.
There are essentially three types of empty rectangles which we have to consider.
A type A rectangle is supported by two points of M_1 [M_2] and M_3 [M_4], respectively.
A type B rectangle is supported by two points of one quadrant and one point each of two other quadrants, while a type C rectangle is supported by one point of each quadrant.
It is easy to see that the directions of support as given in table 1 are the only possible ones:
For both cases of type A rectangles this is trivial. Concerning type B rectangles let's for exemple look at case

5. There is only one supporting point in the left half, which must be a left support, since otherwise the rules of theorem 1 would be violated. There is only one supporting point in the upper quadrant of the right half, which must be an upper support for the same reason.
Having exactly one supporting point in each quadrant (type C), there are two possible cases. The supporting point sp_1 in the lower left quadrant M_1 must either be a left or lower support, since otherwise there would be no supporting point in M_2 or M_4, respectively. Assuming sp_1 to be a left [lower] support, it is easy to see that the other three directions of support are determined by this choice. With this we get exactly two cases of type C.

In order to compute the final result, **LER** will do 16 global shifts (called type A [B,C] shifts for cases of type A [B,C]) determining the largest empty rectangle for each case (if it exists) in linear time, respectively. The maximum area empty rectangle as described by theorem 1 is the largest of these 16 (or less) rectangles.
Before we can give the details of type A [B,C] shifts, we need the following definition and lemma.

<u>Definition 2:</u>
Let (x_1,y_1), (x_2,y_2) be two elements of M_i $(i=1,\dots,4)$ with $x_1<x_2$.
(x_1,y_1) and (x_2,y_2) are called "close neighbors of M_i" $:\Leftrightarrow$ there is no other point (x_3,y_3) in M_i with $x_1<x_3<x_2$.

<u>Lemma 3:</u>
Let r be an empty rectangle as described by theorem 1 and $\{t_1,t_2\}=B_i\subseteq M_i$ two supporting points in the same quadrant M_i $(i=1,\dots,4)$ $\Rightarrow$
t_1 and t_2 are close neighbors of M_i.

Proof:
Let w.l.o.g. i=1 and r be an empty rectangle as described by theorem 1 supported by $\{t_1,t_2\}=B_1$ (see fig.5). Assuming there is a point $t'\in M_1$ with x-coordinate between t_1 and t_2, t' will lie inside r, since it is ur-maximal with resp. to S_1' and has distinct y-coordinate - a contradiction. ∎

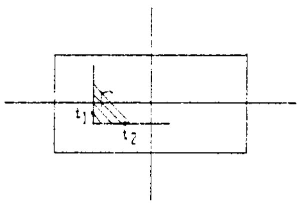


fig.5

With this we can describe the shifts as follows:

<u>Type A shifts</u>
Let's w.l.o.g. take case 3. From lemma 3 we know that all pairs of supporting points of M_3 $[M_1]$ are close neighbors.

Thus, we sort $M_1,\dots,M_4$ by x-coordinate and let each pair of close neighbors of M_3 (represented by the maximum x-coordinate and maximum y-coordinate of both points) shift through M_2 and M_4 to find the rightmost point with smaller y-coordinate and uppermost point with smaller x-coordinate, respectively. With these two points "in mind", we shift each pair of close neighbors of M_3 through M_1 and determine the largest rectangle not containing any point of M_2 or M_4. Pipelining these processes yields linear running time.

Type B shifts

Let's w.l.o.g. take case 2. With the same arguments as given above, we sort $M_1,\dots,M_4$ by x-coordinate. Taking each pair of close neighbors of M_3 as described above, it is easy to see, that the supporting lower point $l_4 \in M_4$ and supporting left point $l_2 \in M_2$ are determined. l_4 is the uppermost point of M_4 with smaller x-coordinate and l_2 is the rightmost point of M_2 with smaller y-coordinate. Thus we let each pair of close neighbors of M_3 shift through M_4 and M_2 and find these both points, respectyvely. With this, a shift through M_1 shows whether this rectangle is empty. Pipelining these processes yields linear running time, too.

Type C shifts

A type C shift is essentially the same, since given the left [lower] supporting point in M_1 the three other supporting points are determined.

3.3. Accumulated Running Time and Space Requirement

Since the the computation of the final result can be done in linear time, the accumulated running time of all divide and merging steps is linear with resp. to n. Each cell has to store a constant amount of information yielding a linear space requirement, too.

REFERENCES

[1] M.J.Atallah, S.E.Hambrusch, SOLVING TREE PROBLEMS ON A MESH-CONNECTED PROCESSOR ARRAY, Report CSD-TR-518, Purdue Univ., West Lafayette, April 1985

[2] B.M.Chazelle, COMPUTATIONAL GEOMETRY ON A SYSTOLIC CHIP, IEEE Trans. on Computers, Vol. C-33, No.9, Sept. 1984

[3] B.Chazelle, R.L.Drysdale, D.T.Lee, COMPUTING THE LARGEST EMPTY RECTANGLE, Proc. Symp. on Theoetical Aspects of Computer Science, 1984, pp 43-54

[4] F.Dehne, $O(n^{1/2})$ ALGORITHMS FOR THE MAXIMAL ELEMENTS AND ECDF SEARCHING PROBLEM ON A MESH-CONNECTED PARALLEL COMPUTER, Techn. Report, Univ. of Wuerzburg, W.-Germany, 1985

[5] A.L.Fisher, H.T.Kung, L.M.Monier, Y.Dohi, A PROGRAMMABLE SYSTOLIC CHIP, Journal of VLSI and Computer Systems, Vol.I, No.2, 1984

[6] H.T.Kung, F.Luccio and F.P.Prparata, ON FINDING THE MAXIMA OF A SET OF VECTORS, J. of the ACM, Vol.22, No.4, Oct. 1975

[7] R.Miller and Q.F.Stout, COMPUTATIONAL GEOMETRY ON A MESH- CONNECTED COMPUTER,Proc.1984 Int. Conf. on Parallel Proc.

[8] A.W.Naamad, W.L.HSU, D.T.Lee, ON THE MAXIMUM EMPTY RECTANGLE PROBLEM, Disc. Applied Math.8, 1984

[9] D.Nassami and S.Sahni, FINDING CONNECTED COMPONENTS AND CONNECTED ONES ON A MESH-CONNECTED PARALLEL COMPUTER, SIAM J. COMPUT., Vol.9, No.4, Nov. 1980

[10] C.D.Thompson and H.T.Kung, SORTING ON A MESH-CONNECTED PARALLEL COMPUTER, Comm. of the ACM, Vol.20,No.4, April 1977

[11] J.D.Ullman, COMPUTATIONAL ASPECTS OF VLSI, Principles of Computer Science Series, Computer Science Press, 1984