UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*

FINDING THE CONTOUR OF A UNION OF ISO-ORIENTED RECTANGLES

5. TYPE OF REPORT & PERIOD COVERED

Technical Report

6. PERFORMING ORG. REPORT NUMBER

R-853 (ACT-16); UILU-ENG 78-2246

7. AUTHOR(s)

Witold Lipski, Jr. and Franco P. Preparata

8. CONTRACT OR GRANT NUMBER(s)

NSF MCS 78-13642
N00014-79-C-0424

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Joint Services Electronics Program

12. REPORT DATE

August 1979

13. NUMBER OF PAGES

18

14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*

15. SECURITY CLASS. *(of this report)*

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Computational geometry, segment tree, contour, rectangles.

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Let $R_1,\ldots,R_m$ be rectangles on the plane with sides parallel to the coordinate axes. An algorithm is described to find the contour of $F = R_1 \cup \ldots \cup R_m$ in $O(m\log m + p\log(2m^2/p))$ time, where p is the number of edges in the contour. This is $O(m^2)$ (optimal) in the general case, and $O(m\log m)$ (optimal) when F is without holes (then $p \leq 8m-4$).

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

UILU-ENG 78-2246

FINDING THE CONTOUR OF A UNION OF ISO-ORIENTED RECTANGLES

by

Witold Lipski, Jr.
Franco P. Preparata

This work was supported in part by the National Science Foundation under Grant NSF MCS 78-13642 and in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under Contract N00014-79-C-0424.

Revised

FINDING THE CONTOUR OF A UNION OF ISO-ORIENTED RECTANGLES

Witold Lipski, Jr.* and Franco P. Preparata

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA

Abstract

Let $R_1,\ldots,R_m$ be rectangles on the plane with sides parallel to the coordinate axes. An algorithm is described to find the contour of $F = R_1 \cup \ldots \cup R_m$ in $O(m\log m + p\log(2m^2/p))$ time, where p is the number of edges in the contour. This is $O(m^2)$ in the general case, and $O(m\log m)$ when F is without holes (then $p \leq 8m-4$); both of these performances are optimal.

Keywords and phrases: Computational geometry, segment tree, contour, rectangles.

---

*On leave from Institute of Computer Science, Polish Academy of Sciences, P. O. Box 22, 00-901 Warsaw PKiN, Poland.

This work was supported in part by the National Science Foundation under Grant MCS 78-13642 and in part by the Joint Services Electronics Program under Contract N00014-79-C-0424.

1. Introduction

In this paper we consider the following geometric problem. Let $R_1,\ldots,R_m$ be rectangles in the plane with sides parallel to the coordinate axes. The union $F = R_1 \cup \ldots \cup R_m$ may consist of one or more disjoint connected components, and each component may or may not have internal holes (note that a hole may contain in its interior some connected components of F). The contour (boundary) of F consists of a collection of disjoint cycles composed of (alternating) vertical and horizontal edges. By convention, any edge is directed in such a way that we have the figure on the left while traversing the edge; this is equivalent to saying that a cycle is oriented clockwise if it is the boundary of a hole, and counterclockwise if it is an external boundary of a connected component. Given $R_1,\ldots,R_m$, our task is to find the contour of $F = R_1 \cup \ldots \cup R_m$.

There are several applications which involve iso-oriented rectangles, or, without loss of generality, rectangles whose sides are parallel to the coordinate axes. Suffice it to mention its relevance to Very Large Scale Integration [5], to geography and related areas [3], to computer graphics (hidden-line problems, shadows, etc.), and to the organization of data in two-dimensional magnetic bubble memories [6]. There is also considerable theoretical interest in problems related to the one discussed in this paper, such as finding the measure of the union of rectangles [2], testing a set of rectangles for disjointness and reporting all the intersections [4], etc.

The paper is organized as follows. In Section 2, we informally describe the algorithm, with the aid of an example. Then, in Section 3, we give the details of the main data structure which is crucial in an efficient implementation of our algorithm, and we describe the implementation of the basic operations performed on this data structure. Section 4 contains

a final description of the whole algorithm, and the analysis of its performance, including a lower bound which proves the optimality of the algorithm when the figure F has no holes.

2. Informal Description of the Algorithm

We informally illustrate the method with the aid of an example. The algorithm consists of two phases. In the first phase we find the set V of vertical edges of the contour (edges 1 through 10 in Fig. 1); in the second

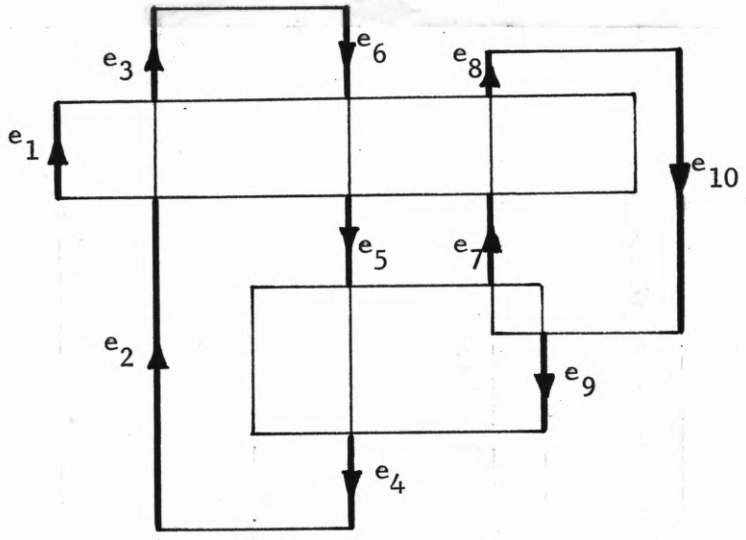

Fig. 1. An instance of the problem.

phase we link these vertical edges by means of horizontal edges to form the oriented cycles of the contour.

In order to obtain the set V we scan the abscissae corresponding to vertical sides of rectangles from left to right. At a generic abscissa c, the section $\mathcal{I}$ of the vertical line x = c with F is a disjoint union of intervals. This section remains constant between two consecutive vertical sides of the rectangles, and is updated in the scan each time one such side is reached. If s is a vertical side of some rectangle R at abscissa c, the portion of the contour of F contributed by s is given by $s \cap \bar{\mathcal{I}}$ where $\bar{\mathcal{I}}$ is, as usual, the union of intervals representing the set theoretical complement of $\mathcal{I}$ on the line x = c (strictly speaking, $\mathcal{I}$ should be

understood as the section to the left or right of $x = c$, depending on whether s is a left or right side of a rectangle). Storing and updating $\mathcal{L}$ and an efficient determination of $s \cap \overline{\mathcal{L}}$ both require the use of some nontrivial techniques and data structures; a discussion of these is deferred to the next section. We assume that each of the edges in V is directed upwards or downwards depending on whether it originates from a left or right side of a rectangle.

We denote by $(x;b,t)$ a vertical edge of abscissa x having b and t $(b < t)$ as its bottom and top ordinates, respectively; similarly, $(y;\ell,r)$ represents a horizontal edge of ordinate y having ℓ and r $(\ell < r)$ as its left and right abscissae, respectively.

We shall now describe the second phase of the algorithm. We first augment the set V by adding triples $(x;t,b)$ for all $(x;b,t)$ in V. Notice that every triple $(x;y_1,y_2)$ in the resulting set V^* is in a one-to-one correspondence with a vertex (x,y_1) of the contour, and that every vertical edge $(x;b,t)$ of the contour is represented in V^* exactly twice by triples corresponding to vertices (x,b) and (x,t) of the contour. We now sort the triples $(x;y_1,y_2)$ in V^* in lexicographic ascending order by (y_1,x); letting $e_j = (x_j;b_j,t_j)$, in our example we obtain the sequence:

$(x_2;b_2,t_2)$ $(x_4;b_4,t_4)$ $(x_4;t_4,b_4)$ $(x_9;b_9,t_9)$ $(x_9;t_9,b_9)$ $(x_{10};t_{10},b_{10})$
$(x_5;b_5,t_5)$ $(x_7;b_7,t_7)$ $(x_1;b_1,t_1)$ $(x_2;t_2,b_2)$ $(x_5;t_5,b_5)$ $(x_7;t_7,b_7)$
$(x_1;t_1,b_1)$ $(x_3;b_3,t_3)$ $(x_6;b_6,t_6)$ $(x_8;b_8,t_8)$ $(x_8;t_8,b_8)$ $(x_{10},t_{10}b_{10})$
$(x_3;t_3,b_3)$ (x_6,t_6,b_6).

In general, let $a_1,a_2,\ldots,a_p$ (p even) be the resulting sequence. It is easy to see that the horizontal edges of the contour can now be obtained as the segments joining the vertices represented by a_{2k-1} and a_{2k}, for $k = 1,2,\ldots,p/2$. More exactly, let $a_{2k-1} = (\ell;y,y_1)$, $a_{2k} = (r;y,y_2)$.

The horizontal edge $(y;\ell,r)$ is assigned the direction from left to right if either the edge corresponding to the triple $(\ell;y,y_1)$ is directed downwards and $y < y_1$, or if $(\ell;y,y_1)$ is directed upwards and $y_1 < y$; otherwise we direct $(y;\ell,r)$ from right to left. In our example, the pair $(a_1,a_2) = ((x_2;b_2,t_2),(x_4;b_4,t_4))$, with $b_2 = b_4$, gives rise to the horizontal edge $(b_2;x_2,x_4)$ which is oriented from left to right because the edge e_2 - corresponding to $(x_2;b_2,t_2)$ - is directed downwards and $b_2 < t_2$; by contrast the pair $(a_{17},a_{18}) =((x_8;t_8,b_8), (x_{10};t_{10},b_{10}))$, with $t_8 = t_{10}$ gives rise to the edge $(t_8;x_8,x_{10})$, which is directed from right to left because e_8 is directed downwards but $t_8 \not< b_8$. It is clear that by a single scan of the sequence $a_1,\ldots,a_p$ we may produce all the horizontal edges and doubly link them into the cycles of the contour, in such a way that the "forward" links determining the orientation of a cycle are consistent with the direction of edges. Then we may find explicitly the cycles of the contour and identify each of them by a vertical edge with the minimal value of abscissa (this can be done in $O(p)$ time). Notice that a cycle corresponds to the boundary of a hole if this edge is directed upwards, and to an external portion of boundary otherwise.

3. Efficient Determination of $s \cap \mathcal{J}$

In order to efficiently implement the first phase of the algorithm we propose to normalize the ordinates of horizontal edges of rectangles (i.e., to replace each ordinate by its rank in the set of ordinates) and to make use of a segment tree, a data structure introduced by Bentley [2], which we now recall.

Let a,b be integers with $a < b$. The segment tree T(a,b) is built in the following recursive way. It consists of the root v with $B[v] = a$, $E[v] = b$[(1)], and, if $b-a > 1$, of a left subtree $T(a,\lfloor(a+b)/2\rfloor)$ and right subtree $T(\lfloor(a+b)/2\rfloor,b)$. The roots of these subtrees are pointed to by LSON[v] and RSON[v], respectively. If $b-a = 1$ then $LSON[v] = RSON[v] = \Lambda$. In our application we shall use the segment tree $T(1,h) \triangleq T$, where $h \leq 2m$ is the number of distinct ordinates.

As is well-known (see also the procedure INSERT below), an interval $s = [b,e]$ (with $1 \leq b < e \leq h$), can be subdivided into O(logm) segments, each of which equals [B[v],E[v]], for some node v of T. Thus, for each v we shall have an integer parameter C[v], which denotes the current multiplicity of insertions of segment [B[v],E[v]] at node v. We shall also use the parameter STATUS[v], which may assume one of three values: full, partial, empty. Intuitively, <u>full</u> denotes that $C[v] > 0$, <u>partial</u> denotes that $C[v] = 0$ but $C[u] > 0$ for some $u \neq v$ in the subtree rooted at v, and <u>empty</u> that $C[u] = 0$ for every u in this subtree. In our algorithm we scan the vertical sides of the rectangles from left to right. Each left side is inserted into T and each right side is deleted from T, so that the current section $\mathcal{J}$ is always given by the union of segments [B[v],E[v]] over all full nodes.

Suppose now that, for some node v in T, we have $s = [b,e] \supseteq [B[v],E[v]]$. The contribution of v to $s \cap \overline{\mathcal{J}}$, denoted as compl(v) is given by

$$\text{compl}(v) = [B[v],E[v]] \cap \overline{\mathcal{J}} = \begin{cases} \phi & \text{if STATUS[v]=full} \\ \text{compl}(LSON[v]) \cup \text{compl}(RSON[v]) & \text{if STATUS[v]=partial} \\ [B[v],E[v]] & \text{if STATUS[v]=empty.} \end{cases}$$

(1) Here B and E are mnemonic abbreviations for "beginning" and "end," respectively.

It appears therefore that $s \cap \bar{\mathcal{J}}$ can be obtained as $\bigcup_{v \in U} compl(v)$, where U is the set of O(logm) nodes of T corresponding to the segmentation of s. Notice that $s \cap \bar{\mathcal{J}}$ may consist of several disjoint segments, and that any such segment is obtained as a collection of contiguous intervals (corresponding to a collection of O(logm) empty nodes of T). To insure that each contour edge be produced as a single item, contiguous intervals have to be merged. To implement this, the edges of $s \cap \bar{\mathcal{J}}$ are assembled in a STACK corresponding to a bottom-to-top scan. At the top of STACK there is always the upper extreme of the last inserted edge (or initial portion of this edge); if the lower extreme of the next segment to be added to STACK coincides with the top of STACK, then the latter is deleted from STACK prior to adding to it the upper extreme of the next segment, thereby realizing the desired merges.

In summary, an interval s = [b,e] is inserted into T by the operation INSERT(b,e,root(T)), where INSERT(b,e,v) is the following recursive procedure:

```
 1  procedure INSERT(b,e,v)
    (*insert segment [b,e] at node v of segment tree*)
 2  begin
 3    if (b ≤ B[v]) and (E[v] ≤ e) then
 4      begin COMPL(v) (*see below*)
 5            C[v] := C[v] + 1
 6      end
 7    else begin if b < ⌊(B[v] + E[v])/2⌋ then INSERT(b,e,LSON[v])
 8               if ⌊(B[v] + E[v])/2⌋ < e then INSERT(b,e,RSON[v])
 9         end
      (*STATUS[v] is to be updated*)
10    if C[v] > 0 then STATUS[v] := full
11    else if LSON[v] = Λ then STATUS[v] := empty (*v is a leaf*)
12         else if STATUS[LSON[v]] = STATUS[RSON[v]] = empty then STATUS[v] = empty
13              else STATUS[v] := partial
14  end
```

We finally give the procedure COMPL(v):

```
 1  procedure COMPL(v)
    (*this procedure makes use of a STACK, which is external to it;
     it pushes on STACK a sequence of segments representing the set
     compl(v)*)
 2  begin
 3     if STATUS[v] = empty then (*compl(v) = [B[v],E[v]]*)
 4       begin if B[v] = top(STACK) then delete top(STACK) (*contiguous
                                                  segments are merged*)
 5             else STACK ⇐ B[v] (*beginning of edge*)
 6             STACK ⇐ E[v] (*current termination of edge*)
 7       end
 8     else if STATUS[v] = partial then
 9             begin  COMPL(LSON[v])
10                    COMPL(RSON[v])
11             end
12  end
```

As mentioned before, a collection of vertical contour edges is generated either in correspondence to a rectangle left side (to be inserted into T) or to a right side (to be deleted from T). The procedure DELETE(e,b,v) is analogous to INSERT(e,b,v). The only difference is that in lines 4 and 5 of DELETE we have C[v] := C[v]-1 and COMPL(v), respectively (reversal of the order of updating C[v] and calling COMPL(v) reflects the fact that for a right side the $\mathcal{L}$ in $s \cap \overline{\mathcal{L}}$ is understood as the section to the right of the current abscissa).

Notice that in order to guarantee the correctness of our algorithm we should process, for any value of x, first all left sides (x;b,t) in increasing order of b, and then all right sides in increasing order of b.

4. Performance Analysis

We shall now give a concise description of the algorithm. Each R_i is represented by a quadruple (ℓ_i, r_i, b_i, t_i), where $\ell_i < r_i$, $b_i < t_i$ and $R_i = [\ell_i, r_i] \times [b_i, t_i]$.

Step 1: Form the sets (without repetitions)

$A = \{\ell_i : 1 \le i \le m\} \cup \{r_i : 1 \le i \le m\}$, $B = \{b_i : 1 \le i \le m\} \cup \{t_i : 1 \le i \le m\}$, and sort each of them. From now on we shall identify any ℓ_i or r_i with

its rank in the first ordering, and any b_i or t_i with its rank in the second ordering. (Comment: In this way we transformed our problem into an equivalent one with ℓ_i, $r_i \in \{1,\ldots,|A|\}$, $b_i, t_i \in \{1,\ldots,|B|\}$ for $1 \leq i \leq m$. Notice that $|A|, |B| \leq 2m$, and that all sorting operations can now be performed in $O(m)$ time by standard bucket sorting (see e.g. [1]). The complexity of this step is clearly $O(m \log m)$).

Step 2: Construct the segment tree T. (Comment: This requires $O(m)$ steps.)

Step 3: Sort the set (representing all vertical sides of rectangles)

$$S = \{(\ell_i, \text{left}, b_i, t_i): 1 \leq i \leq m\} \cup \{(r_i, \text{right}, b_i t_i): 1 \leq i \leq m\}$$

lexicographically by all four coordinates (assuming left < right) and put the sorted sequence in QUEUE. (Comment: This can be done in $O(m)$ time.)

Step 4: Scan the vertical sides of rectangles listed in QUEUE. Any such side is inserted (by calling INSERT) into the segment tree T if it is a left side, or deleted (by calling DELETE) from T if it is a right side. Each of these operations contributes a sequence of edges, placed in STACK, which has to be emptied (outputted) each time we pass either to the next value of abscissa, or from left to right sides with the same abscissa (this is necessary in order to avoid incorrect merging of edges). (Comment: The performance analysis of this step will be done below.)

Step 5: Find the contour cycles. (Comment: This part of the algorithm has been described in detail in Section 2 and shown to run in time $O(p)$, where p is the number of contour edges.)

To analyze Step 4 we shall restrict ourselves to calls of INSERT; calls of DELETE can be handled in a similar way. The work done by INSERT

is best analyzed by recalling an interesting property of segment trees. For any segment [b,e], let $\ell(b,e)$ be the node u of T with B[u] = b and with the maximum possible E[u] $\leq$ e (i.e., [B[u],E[u]] is the left-most piece in the segmentation of [b,e]); similarly, let r(b,e) be the node u with E[u] = e and the minimum possible B[u] $\geq$ b. Denote by P_ℓ and P_r the (sets of nodes of the) paths from the root to $\ell(b,e)$ and r(b,e), respectively. It is easily seen that the segmentation of [b,e] is given by $\ell(b,e)$, r(b,e), and the right sons of $P_\ell \backslash P_r$ not in P_ℓ as well as the left sons of $P_r \backslash P_\ell$ not in P_r. Since T is a nearly-balanced tree, the length of each of its paths is bounded by $\lceil \log m \rceil$; thus the work done by a single call of INSERT(b,e,root(T)) - disregarding the calls of COMPL - is O(logm), since it corresponds to traversing $P_\ell \cup P_r$ and spending a fixed amount of work at each node and possibly at its sibling.

Now we shall extend the analysis to account for the work done by COMPL. Let (x;b,t) be the rectangle side being inserted into the segment tree. Let $[b,t] \cap \mathcal{J} = [c_1,c_2] \cup \ldots \cup [c_{2k-1},c_{2k}]$, $c_1 < c_2 < \ldots < c_{2k}$, let $v_{2j-1} = \ell(c_{2j-1},c_{2j})$, $v_{2j} = r(c_{2j-1},c_{2j})$, $1 \leq j \leq k$, and let $P(v_i)$ be the path from v_i to the root of T. It is easy to see that what the recursive procedure INSERT does, together with COMPL, is exactly the preorder traversal of the subtree with node set $\bigcup_{i=1}^{2k} P(v_i)$. A fixed amount of work is done at each node visited, and possibly at its sibling, so that the total work is proportional to $\left| \bigcup_{i=1}^{2k} P(v_i) \right|$. In order to estimate the latter quantity we shall need the following lemma, proved in Appendix A:

Lemma 1. Let be a balanced binary tree with N leaves (so that any path from the root to a leaf is of length $\lfloor \log N \rfloor$ or $\lceil \log N \rceil$). Let L be a subset of nodes of T, and for every $u \in L$ denote by P(u) the set of nodes on the path from u to the root. Then

$$\left| \bigcup_{u \in L} P(u) \right| < |L| \log \frac{16N}{|L|} \quad .$$

Let us denote by n_i the number of disjoint pieces in $s_i \cap \bar{\mathcal{I}}$, where s_i is the i-th rectangle side processed in the segment tree ($1 \leq i \leq 2m$). By noting that each of the disjoint segments [b,e] in $s_i \cup \bar{\mathcal{I}}$ contributes at most two members of set L (namely $\ell(b,e)$ and r(b,e)) and applying Lemma 1 to the segment tree, the work involved in processing s_i can be bounded above by $Cn_i \log \frac{16m}{n_i}$ for some constant C (this expression is interpreted as 0 for $n_i = 0$). Summing it over all i, the global work in Step 4 is bounded above by

$$C \sum_{i=1}^{2m} n_i \log \frac{16m}{n_i} \leq C\ p \log \frac{64m^2}{p} \quad ,$$

where use has been made of the fact that Σn_i does not exceed p/2, the number of vertical edges of the contour. (Strictly speaking, in the presence of overlapping vertical rectangle sides we should write $\Sigma n_i \leq p/2 + 2m$; however, this does not affect the asymptotic complexity of the algorithm.) It is easy to see that $p \leq m^2 + 4m$, so that $C\ p \log(64m^2/p) = O(p \log(2m^2/p))$. Putting this together with the previous comments on Steps 1, 2, 3 and 5 of the algorithm, we obtain:

Theorem 2. The algorithm correctly finds the p-edge contour of a union of m rectangles in time $O(m \log m + p \log(2m^2/p))$.

Notice that $p\log(2m^2/p)$ is increasing in p in the range $[4, m^2 + 4m]$, for $m > 4$. This means that if m is the only measure of the size of the input,

then the complexity of our algorithm can be written as $O(m\log m + (m^2 + 4m)\log(2m^2/(m^2 + 4m))) = O(m^2)$, which is clearly optimal (the algorithm for the same problem in [6] is of complexity $O(m^2 \log m)$).

Suppose now that F has no holes. For this case we prove in Appendix B the following lemma:

Lemma 3. Let $F = R_1 \cup \ldots \cup R_m$ be without holes, and suppose it is composed of k disjoint connected components. Denote by p the number of edges in the contour of F. Then $p \leq 8m-4k$.

Therefore, in the absence of holes, the algorithm runs in time $O(m\log m + 8m\log(2m^2/p)) = O(m\log m)$. We shall now prove that this is optimal under the usual decision tree model, using the fact that in this model sorting of m numbers requires time $\Omega(m\log m)$ (see [1]).

Theorem 4. The complexity of finding a contour of $F = R_1 \cup \ldots \cup R_m$, where F is without holes, is $\Omega(m\log m)$ under the decision tree model.

Proof. We shall show that sorting of m numbers $x_1,\ldots,x_m$ reduces in $O(m)$ time to our problem. Indeed, given x_i, let R_i be the cartesian product of the x-interval $[0,x_i]$ and of the y-interval $[0,M-x_i]$, where $M = \max_{1 \leq i \leq m} x_i + 1$ (see Fig. 2; without loss of generality we assume $x_1,\ldots,x_n > 0$). It is clear that $F = R_1 \cup \ldots \cup R_m$ is without holes and that from the contour of F we can obtain the sorted sequence of the x_i in $O(m)$ time. □

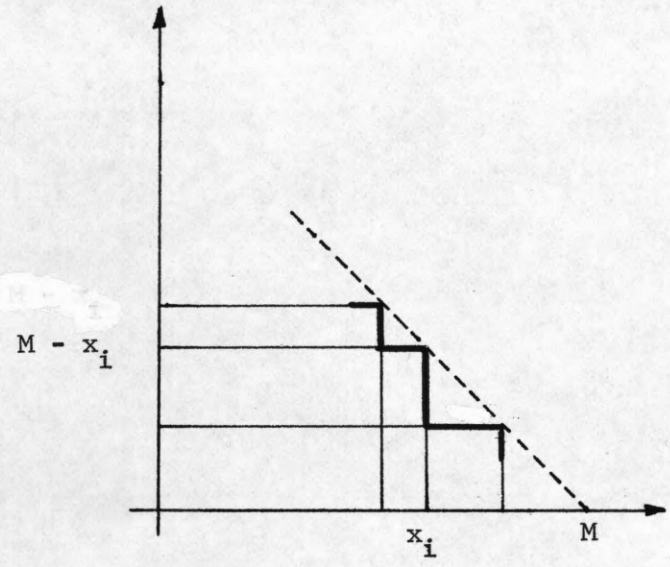

Fig. 2. Reducing sorting to finding the contour of hole-free union of rectangles.

Two comments are in order. The first is that our algorithm requires $O(m+p)$ storage. The second is that for intermediate values of p, say, $p = O(m^{\alpha})$, for $1 \leq \alpha < 2$, our algorithm is not optimal: indeed its running time is $O((m+p)\log m)$. It is an interesting open question whether the problem can in general be solved in time $O(m\log m+p)$.

5. Conclusions

We have described an efficient algorithm for determining the contour of a union of rectangles. The main tool used was the segment tree, a data structure introduced by Bentley, which has already proved useful in several algorithms related to intervals on the real line, and rectangles on the plane. A nice property of the segment tree, which was essential in our algorithm, is that is provides a convenient way of representing the union of a dynamically changing collection of (not necessarily disjoint) intervals.

A related problem is that of finding the external contour of a union of iso-oriented rectangles. It may be proved that it always consists of $O(m)$ edges, and it may be constructed in $O(m \log m)$ time by another algorithm which uses a data structure different from - but related to - the segment tree. This however will be treated in a separate paper.

Appendix A.

Lemma 1. Let T be a balanced binary tree with N leaves (so that any path from the root to a leaf is of length $\lfloor \log N \rfloor$ or $\lceil \log N \rceil$). Let L be a subset of nodes of T, and for every $u \in L$ denote by P(u) the set of nodes on the path from u to the root. Then

$$\left| \bigcup_{u \in L} P(u) \right| < |L| \log \frac{16N}{|L|} .$$

Proof. It is clear that we may restrict ourselves to the case where L is a set of leaves of the tree. Let T(L) be the tree consisting of the nodes in $\bigcup_{u \in L} P(u)$, and let $T_0(L)$ be its subtree consisting of the root of T and those nodes v for which the father of v (or v itself) has at least two descendants in L (see Fig. 3).

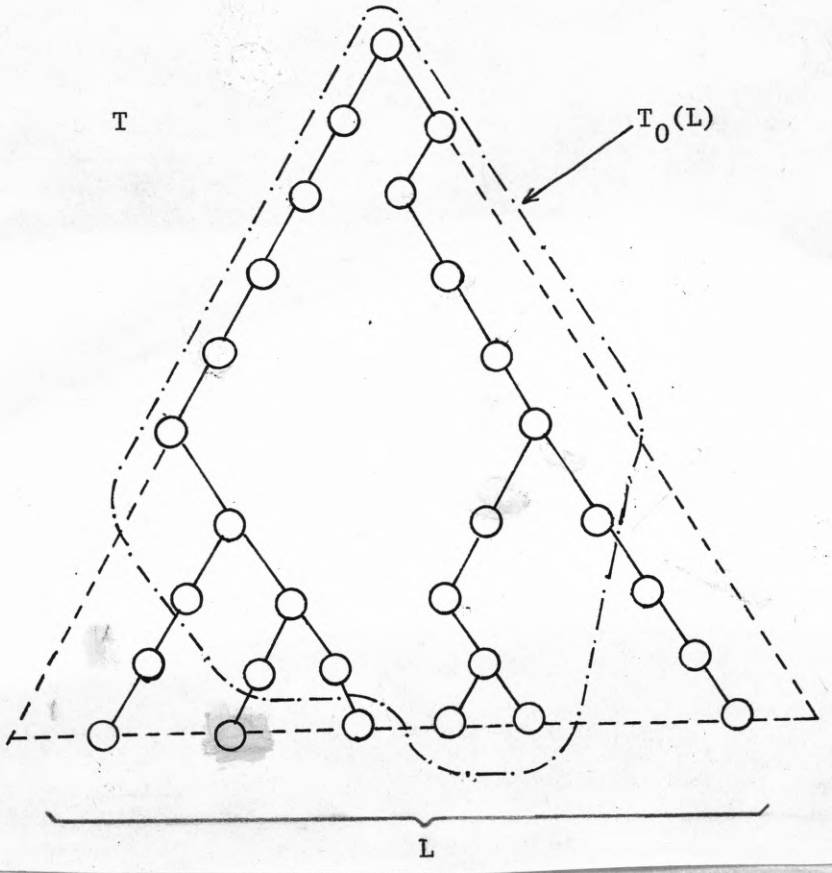

Fig. 3. To the proof of Lemma 1.

It is easy to see that $T_0(L)$ is a tree with $|L|$ leaves. It is also clear that $|T(L)|$ is maximum (for a fixed $|L|$) if $T_0(L)$ is a balanced top-most part of T, with $|L|$ leaves and $2|L|-1$ nodes. In such an extremal case

$$|T(L)| \leq 2|L|-1 + |L|(\lceil \log N \rceil - \lfloor \log |L| \rfloor)$$

$$< |L|(2 + \log N + 1 - \log|L| + 1) = |L| \log \frac{16N}{|L|} \quad . \qquad \square$$

Appendix B

Lemma 3. Let $F = R_1 \cup \ldots \cup R_m$ be without holes, and suppose it is composed of k disjoint connected components. Then $p \leq 8m-4k$.

Proof. We use induction on m. Our lemma is clearly true for $m = 1$. Now suppose it is true for some $m \geq 1$, and consider rectangles $R_i = (\ell_i, r_i, b_i, t_i)$, $1 \leq i \leq m+1$, where $F^* = R_1 \cup \ldots \cup R_m \cup R_{m+1}$ is without holes. Without loss of generality we may assume that the rectangles are labelled so that $\ell_i \leq \ell_{m+1}$ for $1 \leq i \leq m$. It is easy to see that $F = R_1 \cup \ldots \cup R_m$ is without holes. Indeed R_{m+1} cannot fill a hole in F, because R_{m+1} lies entirely to the right of any hole in F. Suppose F consists of k connected parts, and R_{m+1} "glues together" q parts of F. Notice that it necessarily means that the intersection of the left side of R_{m+1} with F consists of q disjoint segments. It is easy to see that the increase of the number of edges in the contour caused by adding R_{m+1} may come from the following sources (see Fig. 4).

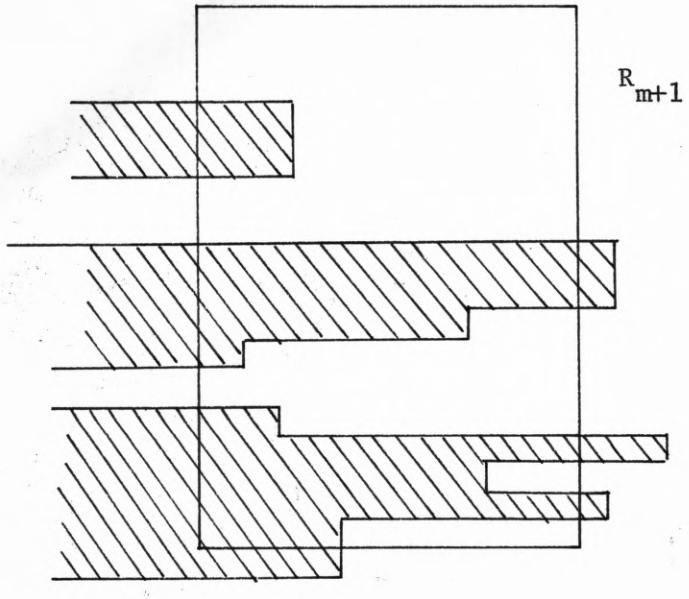

Fig. 4. Adding rectangle R_{m+1} to $F = R_1 \cup \ldots \cup R_m$.

(a) segments between the q disjoint segments on the left side of R_{m+1}:	at most q+1
(b) similar segments on the right side of R_{m+1}:	at most q+1
(c) bottom and top side of R_{m+1}:	at most 2
(d) doubling the edges of the contour of F which intersect both the left and right side of R_{m+1}:	at most 2q
TOTAL	at most 4q+4

Taking into account that the number of connected components of F* is equal to $k^* = k-q+1$, and using the inductive assumption, we obtain the following estimate on the number of edges in the contour of F:

$$p^* \leq (8m - 4k) + (4q + 4) = 8(m + 1) - 4(k - q + 1) = 8(m + 1) - 4k^*.$$

□

Notice that there exist hole-free unions of m rectangles with $p = 8m - 4k$, for any m and any $1 \leq k \leq m$. Every connected part of such a union is of the form shown in Fig. 5.

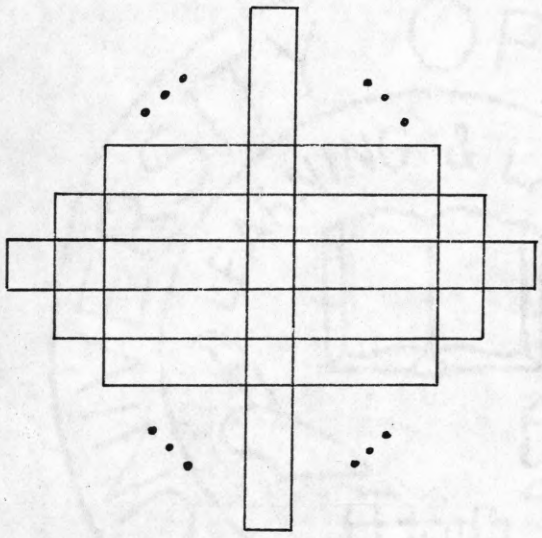

Fig. 5. A hole-free connected union of rectangles with $p = 8m - 4$.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Mass. 1974.

2. J. L. Bentley, Solutions to Klee's rectangle problems. Unpublished notes, Carnegie-Mellon University, 1977.

3. J. L. Bentley and M. I. Shamos, Optimal algorithms for structuring geographic data. Proc. Symp. on Topological Data Structures for Geographic Information Systems, Harvard, Mass. 1977, pp. 43-51.

4. J. L. Bentley and D. Wood, An optimal worst-case algorithm for reporting intersections of rectangles. Carnegie-Mellon University, 1979.

5. U. Lauther, 4-Dimensional binary search trees as a means to speed up associative searches in design rule verification of integrated circuits. Journal of Design Automation and Fault-Tolerant Computing 2, 1978, pp. 241-247.

6. E. Lodi, F. Luccio, C. Mugnai, and L. Pagli, On two dimensional data organization I. Fundamenta Informaticae, to appear.