Intern. J. Computer Math., 1983, Vol. 13, pp. 209–219
0020-7160/83/1304-0209 $18.50/0
© Gordon and Breach Science Publishers Inc., 1983
Printed in Great Britain

A New Approach to Rectangle Intersections

Part I

HERBERT EDELSBRUNNER

Institute for Information Processing, Technical University of Graz, Schiesstattgasse 4a, A-8010 Graz, Austria

(*Received September 1982*)

Rectangle intersections involving rectilinearly-oriented (hyper-) rectangles in d-dimensional real space are examined from two points of view. First, a data structure is developed which is efficient in time and space and allows us to report all d-dimensional rectangles stored which intersect a d-dimensional query rectangle. Second, in Part II, a slightly modified version of this new data structure is applied to report all intersecting pairs of rectangles of a given set. This approach yields a solution which is optimal in time and space for planar rectangles and reasonable in higher dimensions.

KEY WORDS: Computational geometry, rectilinearly-oriented rectangle, intersection, searching problem, data structure, concrete complexity.

CATEGORIES AND SUBJECT DESCRIPTORS: E.1 [Data]: Data structures—*trees*; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical a'gorithms and problems—*geometrical problems and computation, sorting and searching*; H.3.3 [Information Storage and Retrieval]: Information search and retrieval—*search process.*

1. INTRODUCTION

The young branch of computer science called computational geometry deals with the computational complexity of geometric problems. This paper investigates such a problem which has applications in areas like VLSI design and computer graphics. We refer to Bentley and Wood [2] for details concerned with these applications.

We first introduce a few definitions which help us to classify the area of computational geometry. An important class of problems, the so-called searching problems, receive their motivation from the theory of databases: A *searching problem* P involves a set S of *objects* and answers *queries* which depend on S and on a *query object* q. Thus, P can be viewed as a function which maps S and q into some *answer* $P(S, q)$. Note that three types of data, namely objects, query objects, and answers are involved.

A typical example is the so-called *range searching problem* which involves a set S of points in d dimensions. It maps S together with a query *range* q which is the Cartesian product of d intervals one on each coordinate-axis, into the set of points in S which are contained in q. The range searching problem is important to our discussion in Section 2.

The particular searching problem which is investigated in this paper is the so-called rectangle intersection searching problem. It is studied in the d-dimensional real space, for $d \geq 1$. A *d-dimensional rectilinearly-oriented rectangle* (for short *d-rectangle*) is the Cartesian product of d closed intervals one on each coordinate-axis. Two d-rectangles are said to *intersect* if they have at least one point in common. The (*d-dimensional*) *rectangle intersection searching problem* involves a set S of d-rectangles as objects and maps S together with a query d-rectangle q into the set of d-rectangles in S which intersect q.

The usual way to treat a searching problem on a computer is to store the set S of objects in some data structure which allows us to answer queries efficiently. An everyday example is to sort a set of n numbers which allows us to decide in $0(\log n)$ time by binary search whether or not a later specified query number is in the set. To answer a query of the range searching problem or the rectangle intersection searching problem means to report the points or rectangles which are in the desired set.

In certain applications, however, one has to solve so-called *single-shot problems*. That is, a solution of an instance of the problem is computed once and nothing has to be saved in the computer in order to answer later specified queries. Some single-shot problems are directly related to searching problems. We identify two classes of such problems which relate to searching problems whose objects are of the same type as their query objects.

The *all-elements problem* of a searching problem P maps a set S of objects into the set of answers $P(S-\{q\}, q)$, q in S. Intuitively, an all-elements problem answers a query for each object in the set. The all-elements problem of the rectangle intersection searching problem involving a set S of rectangles is solved by reporting for each rectangle q in S those rectangles in S which intersect it.

Since the intersection relation is symmetric (i.e. a rectangle r intersects another rectangle s if and only if s intersects r) we might not be interested in the intersecting rectangles for each rectangle in S but rather in all pairs of intersecting rectangles. More general, we define the *all-pairs problem* of a searching problem P which maps a set S of objects into the set of pairs (p, q), p and q in S, such that p is in $P(S-\{q\}, q)$ and q is in $P(S-\{p\}, p)$. In particular, Part II of this paper examines the all-pairs problem of the rectangle intersection searching problem which we call the *all intersecting rectangles problem.*

The current paper consists of two parts. Section 2 of Part I examines the possibility to solve the rectangle intersection searching problem by means of solutions for the well-known range searching problem. Then Section 3 of Part I improves the thus obtained results by the design of a new data structure for rectangles. This solution is optimal for the one-dimensional case and reasonable in higher dimensions. Part II applies the data structure developed in Section 3 to the all intersecting rectangles problem. The solution obtained is optimal in the two-dimensional case where it improves previous results, and it is reasonable in three and higher dimensions.

The development described in this paper took place in 1980 and was previously reported in Edelsbrunner [3, 4]. Since then several rediscoveries and improvements occurred. McCreight [8] independently obtained a data structure for intervals which is very similar to the 1-fold rectangle tree presented in Section 3.1. His so-called tile tree has the disadvantage that the worst-case bounds of our tree are not guaranteed. Lee and Wong [7] independently discovered the correspondence between rectangle intersection searching and range searching which is presented in Section 2. Finally, Six and Wood [10] as well as Edelsbrunner [5] improved our results for $d \geq 2$ dimensions.

2. RANGE SEARCHING FOR FINDING RECTANGLE INTERSECTIONS

The rectangle intersection searching problem is shown to be solvable by data structures originally designed for the range searching problem. This result follows from a transformation of d-rectangles into $2d$-dimensional points.

LEMMA 2.1. *A rectangle intersection query involving a set S of d-rectangles and a query d-rectangle q can be answered by* (1) *mapping S into an equal-sized set S' of points in $2d$ dimensions,* (2) *mapping q into a range q' in $2d$ dimensions,* (3) *solving the range query for S' and q' and* (4) *interpreting the answer for the range query as the answer for the rectangle intersection query for S and q.*

Proof Let us first examine the rectangle intersection problem in one dimension. Each 1-rectangle is a closed interval. An interval $i=[i_1, i_2]$ is fully determined by two values, namely the left endpoint i_1 and right endpoint i_2. Thus, we may interpret this pair of values (i_1, i_2) as a point in two dimensions. Let S' be the set of two-dimensional points such that $i'=(i_1, i_2)$ is in S' if and only if $i=[i_1, i_2]$ is in S. We proceed by mapping the query interval $q=[q_1, q_2]$ into the range q' which is the Cartesian product of the intervals $(-\mathrm{inf}, q_2]$ on the first and $[q_1, \mathrm{inf})$ on the second coordinate-axis, where inf stands for the infinite value. Now the range query for S' and the query range q' is answered. As can be readily seen, each point of S' in q' is the image of an interval in S which intersects q which completes the argument for the one-dimensional case.

Note that two d-rectangles intersect if and only if their intervals on the jth coordinate-axis intersect, for $1 \leqq j \leqq d$. This fact allows the generalization to $d \geqq 2$ dimensions to be accomplished. Let S be a set of d-rectangles and let S' denote the set of points in $2d$ dimensions such that the point $r'=(r_1, r_2, \ldots, r_{2d})$ is in S' if and only if the d-rectangle r which is determined by the intervals $[r_1, r_2], \ldots, [r_{2d-1}, r_{2d}]$ is in S. The query d-rectangle q determined by the intervals $[q_1, q_2], \ldots, [q_{2d-1}, q_{2d}]$ is mapped into the $2d$-dimensional range q' determined by the unbounded intervals $(-\mathrm{inf}, q_2]$, $[q_1, \mathrm{inf})$, $(-\mathrm{inf}, q_4], \ldots,$ $[q_{2d-1}, \mathrm{inf})$. We find all rectangles which intersect q by solving the range query for S' and the query range q'. This completes the argument.

These considerations give us a first glimpse of the complexity of the rectangle intersection searching problem, since the range searching problem is well studied and efficient solutions are known.

COROLLARY 2.2. *For the rectangle intersection searching problem involving a set S of n d-rectangles there exists a data structure which requires* $0(n\log^{2d-1} n)$ *space and* $0(n\log^{2d-1} n)$ *time for construction such that* $0(\log^{2d-1} n+t)$ *time suffices to report the t d-rectangles in S which intersect a query d-rectangle.*

The bounds are achieved by the range tree which is described in Bentley [1] and improved in Willard [11].

As the range searching problem is already well studied we do not attempt to improve the complexity of range searching in order to improve the bounds for the rectangle intersection searching problem. But there are two facts which lead us to hope that a specially designed data structure for the latter problem will be better. These facts are:

i) The d-rectangles are mapped only into a subset of the $2d$-dimensional space, that is to $\{(x_1, x_2, \dots, x_{2d}) \mid x_{2j-1} \leqq x_{2j}, 1 \leqq j \leqq d\}$.

ii) The range in $2d$ dimensions obtained by mapping the query d-rectangle is not an arbitrary range. Each one of its intervals is unbounded in one direction and the finite vertex of the range is a point in $\{(x_1, x_2, \dots, x_{2d}) \mid x_{2j-1} \geqq x_{2j}, 1 \leqq j \leqq d\}$.

In Section 3 we exploit these facts in the design of a new data structure for d-rectangle, the so-called d-fold rectangle tree.

3. A NEW DATA STRUCTURE FOR *d*-RECTANGLES

We first consider the 1-fold rectangle tree which stores 1-rectangles, that is, intervals on a line. Then Section 3.2 generalizes the results to two and higher dimensions.

3.1. The 1-fold rectangle tree

Let S denote a set of n intervals and let $e_1, \ldots, e_{2n}$ be the sorted list of left and right endpoints of these intervals. For convenience, we assume that no two endpoints are the same. Sets of intervals with potentially coinciding endpoints can be treated by the same method trivially modified. The root r of the *1-fold rectangle tree* for S contains

i) a value, $v(r) = (e_n + e_{n+1})/2$, that partitions the list into two parts each comprising exactly n endpoints, and
ii) three pointers to its sons left(r), middle(r), and right(r).

Let S_1 and S_2 denote the set of intervals in S which lie completely to the left and right of $v(r)$, respectively. S_m denotes the set of remaining intervals, that is, those intervals which contain $v(r)$. Left (r) is the root of the 1-fold rectangle tree for the intervals in S_1 and right(r) is the root of the 1-fold rectangle tree for the intervals in S_2. middle(r) is the root of a minimal height binary tree which stores the endpoints of the intervals in S_m in its leaves. We call this tree the *middle subtree* of r. The endpoints in this tree are stored in increasing order and the leaves of the middle subtree are organized as a doubly-chained list which reflects the same ordering. Additionally, the root of the middle subtree contains pointers to the leftmost and rightmost leaves in the middle subtree, respectively, see Figure 3.1.

In a more informal manner we say that the 1-fold rectangle tree is binary tree (termed the *primary structure*) each node of which is associated with a third tree (the middle subtree) which stores a subset of the given intervals. The totality of middle subtrees is termed the *secondary structure*. We call a node belonging to the primary structure a *primary node*, and a node belonging to the secondary structure a *secondary node*.

We leave to the interested reader the task to derive a detailed algorithm which constructs a 1-fold rectangle tree. Such an algorithm follows almost immediately from the definition of the tree. In the sequel, the search algorithm is described which detects for a given 1-fold rectangle tree T and a query interval q all intervals in T which intersect q.

The intersecting intervals are determined by descending the primary structure of T and performing certain actions at each

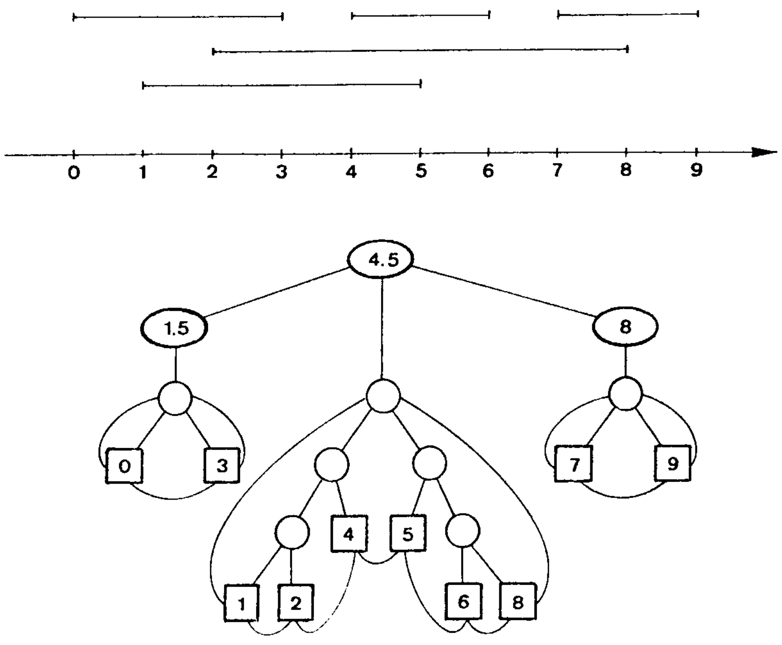

FIGURE 3.1. The 1-fold rectangle tree for five intervals.

primary node visited. Let p denote the current primary node. Two substantially different cases are distinguished:

Case 1. There is no ancestor of p which has its value inside of q.

Case 1.1. The value of p is outside of q. W.l.o.g. we assume that q is to the left of $v(p)$. Then the middle subtree of p is examined, that is, the leaves are traversed from left to right and each interval whose left endpoint is encountered is reported until the first endpoint to the right of q is reached. In addition, the left son of p is visited recursively.

Case 1.2. The value of p is contained in q. Note that q is the first node visited with this property. Then all intervals stored in the middle subtree of p are reported and both sons of p are visited recursively.

Case 2. There exists an ancestor of p which has its value in q. Let $a(p)$ denote the first ancestor on the way from p to the root with this property. W.l.o.g. we assume that p is in the right subtree of $a(p)$.

Case 2.1. The value of p lies to the right of q. Then the middle subtree of p is examined as in Case 1.1 and the left son of p is visited recursively.

Case 2.2. The value of p lies in q. Then all intervals stored in the middle and the left subtree of p are reported and the right son of p is visited recursively.

THEOREM 3.1. *For the one-dimensional rectangle intersection searching problem involving a set S of n intervals there exists a data structure which requires $0(n)$ space and $(n \log n)$ time for construction such that $0(\log n+t)$ time suffices to report the t intervals which intersect a query interval.*

Proof We show the assertion by analyzing the time and space requirements of the 1-fold rectangle tree for S.

In order to answer a query at most $0(\log n)$ primary nodes of the tree are visited. For each such node either the associated middle subtree is examined or all intervals stored in at most two of its subtrees are reported. Since the time needed for each such activity is proportional to the number of intervals detected, $0(\log n+t)$ time suffices to answer a query where t intervals are reported.

The 1-fold rectangle tree for n intervals can be constructed in $0(n \log n)$ time by a recursive procedure which follows the definition of the tree: $0(n)$ time is required for each level of the primary structure and $0(n \log n)$ time suffices to construct the various middle subtrees.

The space required by the primary structure is clearly $0(n)$, and since each interval is stored exactly once in the secondary structure, the total space is $0(n)$. This completes the argument.

3.2. The d-fold rectangle tree

This section extends the methods presented to d dimensions. The so-called d-fold rectangle tree to be described is a straightforward generalization of the 1-fold rectangle tree and is used to accommodate d-rectangles.

Let S denote a set of d-rectangles, for $d \geqq 2$. The *d-fold rectangle tree* for S consists of a 1-fold rectangle tree which stores the dth intervals of the d-rectangles in S. No doubly-chained organization of the leaves and no additional pointers for the secondary roots are required this time. This tree is referred to as the dth *component tree.* For each node p of this tree, we define set(p) as the set of d-rectangles which have at least one of their endpoints stored in the subtree of p. Then p is assigned the $(d-1)$-fold rectangle tree which stores the $(d-1)$-rectangles which are the orthogonal projections onto the first $d-1$ coordinates of the d-rectangles in set(p).

The algorithms for creating a d-fold rectangle tree and searching in it are very similar to the algorithms for the 1-fold rectangle tree. In order to adapt the construction algorithm we have to add the recursive mechanism that associates $(d-1)$-fold trees to the nodes of the dth component tree. The modifications necessary for the search algorithm are:

i) The command of reporting all intervals stored in some subtree T' is replaced by the command to investigate the appropriate lower dimensional subtree.

ii) The examination of a middle subtree which is done by traversing the leaves exploiting the doubly-chained organization is replaced by the determination of $0(\log n)$ disjoint subtrees of this middle subtree whose leaves are exactly the leaves which would be encountered during the traversal. For each of these subtrees the lower dimensional subtree associated with its root is investigated.

THEOREM 3.2. *For the d-dimensional rectangle intersection searching problem involving a set of n d-rectangles there exists a data structure which requires* $0(n \log^{d-1} n)$ *space and* $0(n \log^{d} n)$ *time for construction such that* $0(\log^{2d-1} n + t)$ *time suffices to report the t rectangles which intersect a query rectangle.*

Proof The asserted bounds are shown to be correct for the d-fold rectangle tree T for S.

A query in T visits $0(\log n)$ primary nodes of the dth component tree of T. For each of these nodes either at most two $(d-1)$-fold trees associated with its sons or $0(\log n)$ $(d-1)$-fold trees associated with nodes in the middle subtree are investigated. Let $Q(n,d)$ denote the query time for T not regarding the time required to report the intervals determined. Then the observation above implies $Q(n,d) = 0(\log^2 n)Q(n,d-1)$. Since $Q(n,1)=0(\log n)$, see Theorem 3.1, we have $Q(n,d)=0(\log^{2d-1} n)$ as desired.

Each d-rectangle in S has its endpoints in the subtrees of at most $0(\log n)$ nodes of the dth component tree of T. Thus, the $(d-1)$-fold trees associated with the nodes of the dth component tree store $0(n\log n)$ $(d-1)$-rectangles, altogether. Let $P(n,d)$ denote the time required to construct T. Then

$$P(n,d)=0(n\log n)+P(0(n\log n),d-1)$$

and since $P(n,1)=0(n\log n)$ we have $P(n,d)=0(n\log^d n)$. Let $S(n,d)$ denote the space required by T. By the above observation, we have

$$S(n,d)=0(n)+S(0(n\log n),d-1)$$

and since $S(n,1)=0(n)$, we conclude $S(n,d)=0(n\log^{d-1} n)$ as asserted. This completes the argument.

The investigation of algorithms which determine intersections on d-rectangles is continued in Part II of this paper. There, the data structure developed in this section is used for reporting all intersecting pairs of a set of d-rectangles.

References

[1] J. L. Bentley, Decomposable searching problems, *Inf. Proc. Lett.* **8**, 1979, 244–251.

[2] J. L. Bentley and D. Wood, An optimal worst case algorithm for reporting intersections on rectangles, *IEEE Tr. on Comp.* **C-29**, 1980, 571–577.

[3] H. Edelsbrunner, Dynamic rectangle intersection searching, *Report F47*, Inst. for Inf. Proc. Techn. Univ. of Graz, Austria, 1980.

[4] H. Edelsbrunner, A time- and space-optimal solution of the planar all intersecting rectangles problem, *Report F50*, Inst. for Inf. Proc., Techn. Univ. of Graz, Austria, 1980.

[5] H. Edelsbrunner, Dynamic data structures for orthogonal intersection queries, *Report F59*, Inst. for Inf. Proc., Techn. Univ. of Graz, Austria, 1980.

[6] M. L. Fredman, A lower bound on the complexity of orthogonal range queries, *J. of the ACM* **28,** 1981, 696–705.

[7] D. T. Lee and C. K. Wong, Finding intersections of rectangles by range search, *J. of Algorithms* **2,** 1981, 337–347.

[8] E. M. McCreight, Efficient algorithms for enumerating intersecting intervals and rectangles, *Report CSL-80-9,* XEROX Parc, Palo Alto, Cal., 1980.

[9] H.-W. Six and D. Wood, The rectangle intersection problem revisited, *BIT 20,* 1980, 426–433.

[10] H.-W. Six and D. Wood, Counting and reporting intersections of *d*-ranges, *IEEE Tr. on Comp.* **C-31,** 1982, 181–187.

[11] D. E. Willard, New data structures for orthogonal queries, *Report TR-22-78,* Aiken Comp. Lab., Harvard Univ., Cambr., Mass., 1978.