Intern. J. Computer Math., 1983, Vol. 13, pp. 221–229
0020-7160/83/1304-0221 $18.50/0

Printed in Great Britain

A New Approach to Rectangle Intersections

Part II

HERBERT EDELSBRUNNER

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

(*Received September 1982*)

The study begun in Part I is completed by providing an algorithm which reports all intersecting pairs of a set of rectangles in d dimensions. 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 algorithms and problems—*geometrical problems and computation, sorting and searching*; H.3.3 [Information Storage and Retrieval]: Information search and retrieval—*search process*.

4. THE ALL INTERSECTING RECTANGLES PROBLEM

This section applies the d-fold rectangle tree developed in Section 3 to the all intersecting rectangles problem in $d+1$ dimensions. This problem involves a set S of $(d+1)$-rectangles and requires the determination of all intersecting pairs in S. To this end, a modified version of the d-fold rectangle tree is developed. This so-called offline

dynamic d-fold rectangle tree in conjunction with the plane sweep technique which is described, e.g. in Bentley and Wood [2] leads to efficient solutions for this problem.

4.1. The offline dynamic d-fold rectangle tree

A data structure storing a set of objects is called *dynamic* if it supports at little cost insertions and deletions of objects. We call a data structure *offline dynamic* if only insertions and deletions of objects from some prespecified and usually small set are allowed. Using the d-fold rectangle tree as the underlying data structure, we derive an offline dynamic data structure which stores d-rectangles and permits us to answer d-dimensional rectangle intersection queries efficiently. Intuitively, this data structure for a set S of d-rectangles consists of a skeleton which determines the way how a subset of S is stored. The actually stored subset is indicated by additional information with which the skeleton is augmented. Let us discuss the one-dimensional case first.

The *skeleton* of the *offline dynamic 1-fold rectangle tree* T for a set S of intervals is the 1-fold rectangle tree for S without the doubly-chained organization of the secondary leaves and without the auxiliary pointers of the secondary roots. Yet, this skeleton stores none of the intervals in S. The intervals which are stored in T are indicated by various additional pointers the totality of which we term the *auxiliary structure* of T. The flexibility of T is due to the flexibility of its auxiliary structure while the rigidity of its skeleton guarantees that the structure remains well balanced.

We call a leaf of T *active* if it stores an endpoint of an interval which is actually present in the tree. The auxiliary structure organizes the active leaves in a doubly-chained list. The order in which the active leaves occur in this list is the same as in T. In addition, the auxiliary structure equips each inner node p of T with two pointers leftfinger(p) and rightfinger(p). The former points to technique employed in Bentley and Wood [2] which reduces the the latter points to the rightmost active leaf in this subtree (if it exists). The auxiliary structure is constructed, destroyed, and adjusted while insertions and deletions of intervals are performed. The following actions are taken in order to insert an interval i from S into T:

i) The middle subtree is determined which is ready to store the two endpoints of i.

ii) The leaves of this subtree which store the endpoints of i are active now and therefore are included in the doubly-chained list of active leaves. To this end, the active leaves in between which they have to be inserted are determined exploiting the leftfinger and rightfinger pointers.

iii) The leftfinger and rightfinger pointers of the ancestors of the two new active leaves are adjusted or created.

A deletion is analogous, hence we omit its detailed description. Obviously, each of the three tasks can be accomplished in $0(\log n)$ time.

LEMMA 4.1. *Let S denote a set of n intervals. There exists a data structure which requires $0(n)$ space and $0(n \log n)$ time for construction such that insertions and deletions of intervals from S can be performed in $0(\log n)$ time and $0(\log n + t)$ time suffices to report the t intervals currently in the data structure which intersect a query interval.*

Proof We will show that the asserted bounds are correct for the offline dynamic 1-fold rectangle tree T for S. The space for the skeleton of T is clearly $0(n)$ since this structure requires less space than the original 1-fold rectangle tree for S, see Theorem 3.1. The space required for the auxiliary structure is also $0(n)$ since each node of the skeleton is equipped with at most two additional pointers. The time for construction which is the time required to build up the skeleton of T is also clear from Theorem 3.1. The method of obtaining $0(\log n)$ time for an insertion and deletion is sketched above. In order to answer an intersection query, an almost identical algorithm as the one outlined for ordinary 1-fold rectangle trees can be used. It is readily seen that this algorithm works in $0(\log n + t)$ time which completes the argument.

The generalization to $d \geqq 2$ dimensions is straightforward and analogous to the generalization leading to the d-fold rectangle tree, see Section 3.2. The auxiliary information is only necessary for the first component trees. We omit further details and state without proof:

LEMMA 4.2. *Let S be a set of n d-rectangles. Then 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 an insertion and deletion of a d-rectangle from S can be accomplished in $0(\log^{d}n)$ time and $0(\log^{2d-1}n+t)$ time suffices to report the t d-rectangles currently stored which intersect a query d-rectangle.*

4.2. Computing all intersecting pairs

Let $S=\{r_1, r_2, \ldots, r_n\}$ denote a set of n d-rectangles. Using the offline dynamic d-fold rectangle tree described in Section 4.1, we are able to give a naive algorithm for the d-dimensional all intersecting rectangles problem. The task is reduced to a sequence of insertions and intersection queries.

Algorithm NAIVE METHOD:

First, the skeleton of the offline dynamic d-fold rectangle tree T is constructed which is ready to represent any subset of S.

Then, for j running from 1 to n, the d-rectangles stored in T which intersect r_j are determined and r_j is inserted into T.

Due to Lemma 4.2, Algorithm NAIVE METHOD finds the t intersecting pairs of S in $0(n\log^{2d-1}n+t)$ time and $0(n\log^{d-1}n)$ space. Another more sophisticated approach is the plane sweep technique employed in Bentley and Wood [2] which reduces the d-dimensional task to a sequence of $0(n)$ $(d-1)$-dimensional intersection queries, insertions, and deletions involving only projections of rectangles in S. This method is now described for two dimensions and then sketched for higher dimensions.

Let S denote a set of n 2-rectangles in the sequel called rectangles. Figure 4.1 displays a set of six rectangles which we use as a running example. Now imagine a vertical line L sweeping from left to right through the rectangles, see Figure 4.1.

At any instant of time L divides the set S of rectangles into three disjoint subsets: The set of *dead* rectangles that lie completely to the left of L, the set of *active* rectangles that are cut by L, and the set of *sleeping* rectangles that lie completely to the right of L. In the example depicted in Figure 4.1, A is the only dead rectangle, B and C are active, and D, E and F are sleeping.

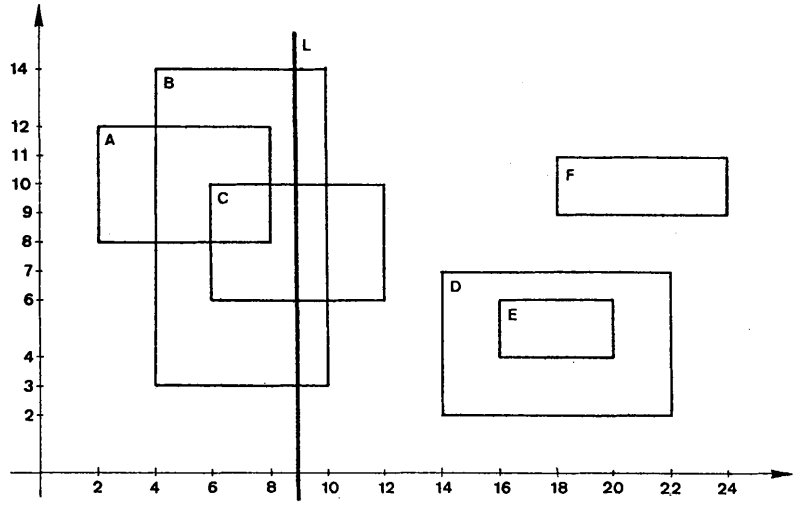

FIGURE 4.1. *L* sweeping through six rectangles.

While L sweeps from left to right, the three subsets of S change following the rules below:

1) Initially, the sets of dead and active rectangles are empty and all rectangles are sleeping.
2) When L reaches the left border of a rectangle then this rectangle becomes active.
3) When L reaches the right border of a rectangle then this rectangle becomes dead.
4) At the end of L's sweep, the sets of active and sleeping rectangles are empty and all rectangles are dead.

Let us first give an informal description of the algorithm that detects the intersecting pairs by use of the plane sweep technique. The three subsets of S change only at *critical x-values*, that is, when L meets the left or right border of a rectangle in S. The $2n$ critical x-values are stored in ascending order in an array and the plane sweep is carried out by scanning this array from left to right. At each critical x-value, certain actions are performed such as activating a rectangle, disactivating a rectangle, or searching for active rectangles which intersect a rectangle which is to become active.

Since the set of active rectangles plays an important role in our algorithm, we store the set in some data structure which allows us to perform insertions, deletions, and searching efficiently.

The formal description of the algorithm follows: Let $x_1, \ldots, x_{2n}$ denote the critical x-values such that $x_i \leqq x_j$ for $i<j$. In degenerate cases, some of the critical x-values may coincide. In these cases, the values indicating left borders come before those indicating right borders. If critical values coincide and are of the same type then their order is immaterial.

Algorithm PLANE SWEEP:

First, the skeleton of the offline dynamic 1-fold rectangle tree T for the y-intervals of the rectangles in S is constructed.

Then, for j running from 1 to $2n$, the following actions are taken:

Case 1 x_j indicates that L meets the left border of a rectangle r in S. Then a query with the y-interval of r as query object is carried out in T. This gives all active rectangles which intersect r. In addition, the y-interval of r is inserted into T.

Case 2 x_j indicates that L meets the right border of a rectangle r. Then the y-interval of r is deleted from T.

Before analyzing the algorithm, let us consider a snap-shot of the data structure used for the set of rectangles depicted in Figure 4.1. L intersects the rectangles C and D which is reflected by the auxiliary structure whose pointers are denoted by dotted lines, see Figure 4.2. Note that only the four leaves of the middle subtree associated with the root of the tree are active.

LEMMA 4.3. *Algorithm PLANE SWEEP reports each pair of intersecting rectangles exactly once and only reports such pairs.*

Proof First, we show that each intersecting pair is reported exactly once. An intersecting pair (r, s) can be detected when r becomes active or when s becomes active. W.l.o.g. we assume that r becomes active before s. Hence, the y-interval of s is not stored in T when r becomes active which implies that (r, s) is not detected at this time. Since r and s intersect by assumption, r must be active when s

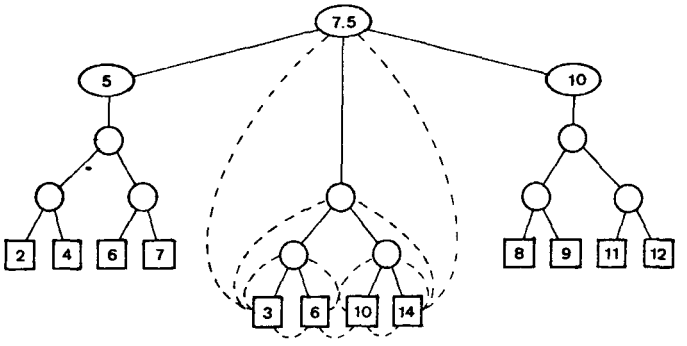

FIGURE 4.2. T storing two intervals.

becomes active. Since the y-intervals of r and s intersect the pair (r, s) is detected when s becomes active.

Assume now that a non-intersecting pair (r, s) is reported by the algorithm. Since r and s do not intersect, at least one of two cases occurs: (1) The x-intervals of r and s do not intersect, or (2) the y-intervals of r and s do not intersect. In the former case there is no point in time when both r and s are active, in the latter case the searches in T with the y-intervals of r and s do not report the pair (r, s). This completes the argument by contradiction.

THEOREM 4.4. *There exists an algorithm which reports the t intersecting pairs of a set S of n rectangles in $O(n \log n + t)$ time and $O(n)$ space.*

Proof We show that the assertion is correct for Algorithm PLANE SWEEP. Lemma 4.3 guarantees that the algorithm reports exactly the t intersecting pairs. The skeleton of the offline dynamic 1-fold rectangle tree T requires $O(n)$ space and $O(n \log n)$ time for construction, see Lemma 4.1. Each one of the insertions, deletions, and intersection queries can be carried out in $O(\log n)$ time not regarding the time for reporting intersections. Since $3n$ such operations are performed we conclude that $O(n \log n)$ time suffices for all these activities. The space remains $O(n)$ since the auxiliary structure constructed while intervals are inserted into T requires $O(n)$ space, see Lemma 4.1. This completes the argument.

The plane sweep technique can easily be extended to three and higher dimensions as done in Six and Wood [10]. The d-dimensional space is swept from left to right by a $(d-1)$-dimensional hyperplane H perpendicular to the x-axis. Thus, the intersecting pairs of a set S of d-rectangles are detected by (1) constructing the skeleton of the offline dynamic $(d-1)$-fold rectangle tree for the orthogonal projections onto H of the d-rectangles in S, and (2) performing $0(n)$ insertions, deletions, and intersection queries involving $(d-1)$-rectangles. This finally implies:

THEOREM 4.5. *There exists an algorithm which reports the t intersecting pairs of a set S of n d-rectangles in $0(n\log^{2d-3} n+t)$ time and $0(n\log^{d-2} n)$ space.*

5. CONCLUSIONS

The contributions of this paper are threefold: (1) The close relationship between rectangle intersection searching in d and range searching in $2d$ dimensions is described which gives us a nice intuition of how rectangles can intersect. (2) A new data structure for d-dimensional rectangles is introduced which is efficient in time and in space. In particular, the solution provided for the one-dimensional rectangle intersection searching problem is optimal in both respects. In addition, an offline dynamic version of this data structure is developed which supports insertions and deletions of d-dimensional rectangles from some prespecified set. (3) This offline dynamic version of the new data structure is used to determine all intersecting pairs of a set of n rectangles. The solution obtained is optimal in two dimensions where $0(n\log n+t)$ time and $0(n)$ space suffice to report the t intersecting pairs.

We briefly compare these results with earlier ones. The two-dimensional all intersecting rectangles problem was studied in Bentley and Wood [2] and in Six and Wood [9]. Their algorithms achieve the same time bound as ours but they require $0(n\log n)$ space for the task. The problem in three and higher dimensions was considered in Six and Wood [10]. Their method dominates ours in time but requires more space, that is, they need $0(n\log^{d-1} n+t)$ time and $0(n\log^{d-1} n)$ space to report the t intersecting pairs. Since the study presented in this paper took place new results have been

obtained by McCreight [8], Lee and Wong [7], and Edelsbrunner [5] which is mentioned in Section 1 of this paper.

Let us finally give a few questions and open problems which come from our investigations. (1) The analysis of the d-fold rectangle tree as presented in Section 3.2 is best possible in the worst case. Nevertheless, we feel that it is terribly pessimistic in the expected case. (2) No other than trivial lower bounds are known for the various rectangle intersection problems presented in this paper. Recent results due to Fredman [6] seem to be a first step towards an appropriate understanding of these issues.

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.* **C31**, 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.