THE MEASURE PROBLEM FOR RECTANGULAR RANGES IN d-SPACE

Jan van Leeuwen and Derick Wood

RUU-CS-79-6

July 1979

Rijksuniversiteit Utrecht

Vakgroep informatica

Budapestlaan 6
Postbus 80.012
3508 TA Utrecht
Telefoon 030–53 1454
The Netherlands

THE MEASURE PROBLEM FOR RECTANGULAR RANGES IN d-SPACE

Jan van Leeuwen
Department of Computer Science
University of Utrecht
P.O. Box 80.012, 3508 TA Utrecht
the Netherlands

and

Derick Wood
Unit for Computer Science
McMaster University
1280 Main Street West
Hamilton, Ontario
Canada L8S 4K1

Technical Report RUU-CS-79-6

July 1979

Department of Computer Science
University of Utrecht
P.O. Box 80.012, 3508 TA Utrecht
the Netherlands

THE MEASURE PROBLEM FOR RECTANGULAR RANGES IN d-SPACE*

Jan van Leeuwen** and Derick Wood***

Abstract. In the Research Problems section of the Am. Math. Monthly V. Klee recently posed the question to find an efficient algorithm for computing the measure of a set of n intervals on the line, and its analog for n hyper-rectangles (ranges) in d-space. The 1-dimensional case is easily solved in $O(n \log n)$, Bentley has proved an $O(n^{d-1} \log n)$ algorithm for dimension $d \geq 2$. We present an algorithm for Klee's measure problem which has a worst case running time of only $O(n^{d-1}+n \log n)$. For d=2 the solution is Bentley's, using his novel segment tree. For $d \geq 3$ we devise a more involved data-structure, based on quad trees, to gain efficiency.

1. Introduction.

In the Research Problems section of the Am. Math. Monthly, Klee [5] recently posed the following problem:

> "Given n intervals on the line, it is desired to find the measure of their union. How efficiently can that be done?"

Klee gave an $O(n \log n)$ algorithm for it based on sorting end-points, but felt one might do better if only sorting could be avoided in favor of a technique more "natural" to the problem. Fredman and Weide [4] soon showed this intuition wrong and proved a lowerbound of $\Omega(n \log n)$ in a general decision tree model permitting comparisons between arbitrary linear functions of the inputs at the nodes.

For Klee's problem in d-dimensional space ($d \geq 2$), the situation is rather more complex. Let us define a (rectilinearly oriented) hyperrectangle or "range" in d-space to be any set of type

---

* This work was carried out while the second author was visiting the University of Utrecht, sponsored by a grant from the Netherlands Organisation for the Advancement of Pure Research (ZWO). The second author was also supported by Natural Sciences and Engineering Research Council of Canada Grant No. A-7700.

** Author's address: Department of Computer Science, University of Utrecht, P.O. Box 80.012, 3508 TA Utrecht, the Netherlands.

***Author's address: Unit for Computer Science, McMaster University, Hamilton, Ontario, Canada L8S 4K1.

$$\{(x_1,\ldots,x_d) \mid l_1 \le x_1 \le u_1 \ \& \ \ldots \ \& \ l_d \le x_d \le u_d\}$$

for bounding parameters l_i and u_i. Now consider the following question, again suggested by Klee [5]:

"MEASURE PROBLEM. Given a set of n ranges in d-space, find the d-measure of their union."

One might want to speak of the fair volume (fair because regions covered more than once should not be counted again) rather than of the d-measure of a set of ranges, as we do not want to give the impression that we treat an integration problem with an arbitrary weighting function on the space. Note that ranges in 2 dimensions are simply rectangles and that the measure problem is the same as asking for the total area covered by the rectangles in this case.

Bentley [1] was apparently the first to launch a major attack on Klee's measure problem for $d \ge 2$. He obtained a surprising $O(n \log n)$ algorithm for the rectangle measure problem (i.e. with d=2), exploiting a novel datastructure called the segment tree. Segment trees have since proved very helpful in solving other problems for sets of rectangles (see e.g. Bentley and Wood [2] and Vaishnavi and Wood [7]) and we will probably see more applications of them soon. Given the efficient algorithm in 2 dimensions, Bentley [1] immediately noted that it could be "lifted" to an $O(n^{d-1}\log n)$ algorithm in $d \ge 3$ dimensions. He suspected that it was not the best one could do, but concluded that at least it settled Klee's question (see [5]) of whether the measure problem in d dimensions could be solved "in a number of steps bounded by a polynomial in d and n". In this paper we shall prove that in dimensions $d \ge 3$ one can indeed do better than Bentley's algorithm and solve the measure problem in only $O(n^{d-1})$ steps.

In section 2 of this paper we shall describe the idea of segment trees and their application to the 2-dimensional measure problem in some detail, partly because Bentley's write-up [1] is not widely available but largely because we shall build on a good understanding of how they function in developing our extended datastructures for $d \ge 3$.

Like Bentley, we shall concentrate on low dimensional problems first and try to pull whatever efficient solution we can find into higher dimensions. In section 3 we shall prove that a quad-tree based datastructure can be constructed which allows the 3-dimensional measure problem to be solved in only $O(n^2)$ steps. It will generalize to the $O(n^{d-1})$ algorithm we claimed for $d > 3$ and thus be the currently most efficient solution for the problem we know.

Again it is hard to formulate a conjecture related to the question of the algorithm's optimality. For the approach we take it seems the best we can hope to get, but we have no argument that it actually is.

2. The area of a set of rectangles in the plane.

Given a set of n rectilinearly-oriented rectangles in the plane, we are asked to compute the total size of the area they cover. To get a feel for the apparent intricacy of the question, figure 1 shows a typical configuration of rectangles we may have to consider. The task is to measure the shaded region.

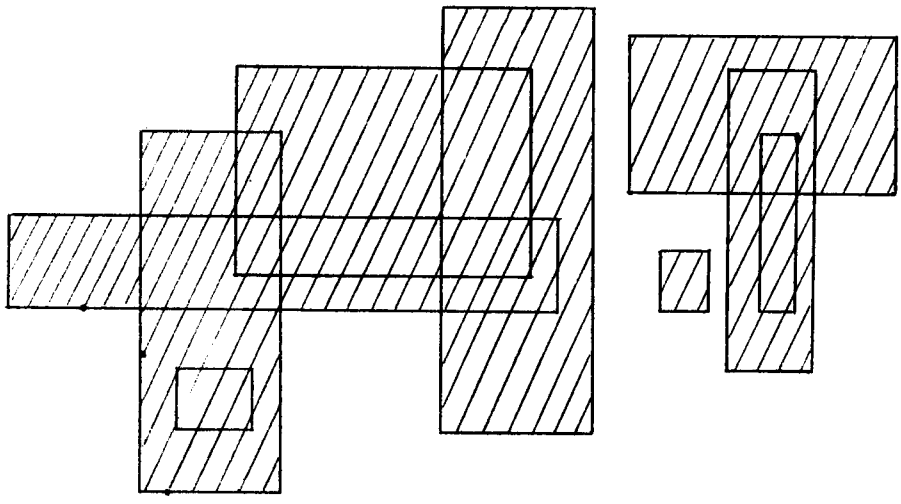


Figure 1.

We shall tacitly assume that the coordinate axes are chosen such that all rectangles are located in the first quadrant. Each rectangle will be given by a "named" quadruple

q = (x-low, x-high, y-low, y-high)

which fully specifies its boundaries (see figure 2). The naming of rectangles guarantees that there will be no confusion between "different" rectangles occupying the same region of the plane.

In a first move we throw out all rectangles which have x-low = x-high or y-low = y-high, as their contribution to the measure will be zero. Thus we can assume from now on that all rectangles in the set are non-trivial.

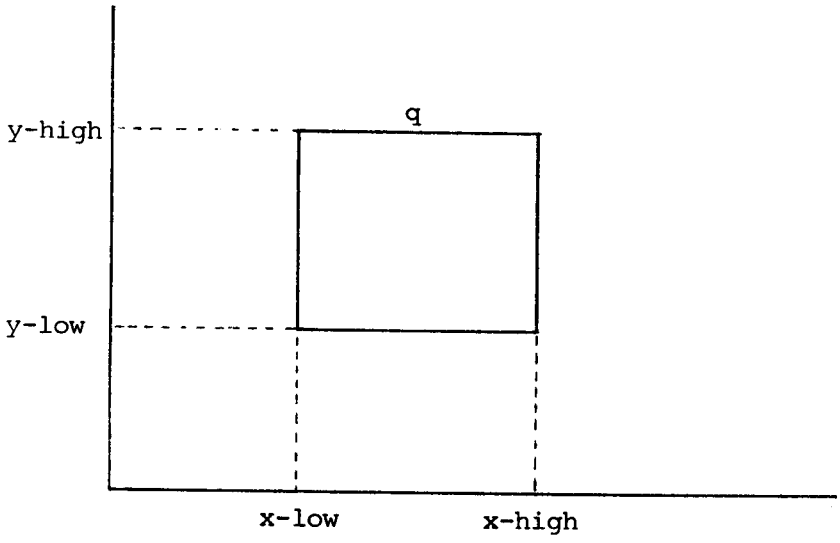


Figure 2.

Separately sort the 2n x-values and 2n y-values we have, while for each value retaining the name of the rectangle it belongs to. Go down the sorted x-list and "merge" identical values, to obtain one list of distinct x-values

$$u_1 < u_2 < \ldots < u_k$$

in which with each u_i a table is associated containing the names of those rectangles which "begin" (when x-low = u_i) or "end" (when x-high = u_i) at u_i. Likewise trim the y-list to

$$v_1 < v_2 < \ldots < v_l$$

but ignore (and indeed, drop) the names of the rectangles they belong to. We won't have a need for this extra information. So far the work has cost us O(n log n) for sorting and a mere O(n) for trimming.

Consider the position of the rectangles in the plane. Each one will be locked in between two vertical lines, which hit the x-axis in some u_i (= x-low) and u_j (= x-high). These lines, and all lines in between, will "sense" the rectangle as a segment. Our algorithm (or rather, Bentley's) is based on the use of a scan-line, which moves across the plane from left to right. See figure 3. Think of the line as scanning a dark radar screen, with segments lighting up whenever the line intersects some rectangles. For a while the intersection will be stable, but things can change when (and only when) the scan-line gets to the "next" u_i position. After all, at no other points can rectangles begin or end! Thus we shall let the scan-line make brief stops at each of the u_i to see what happens.

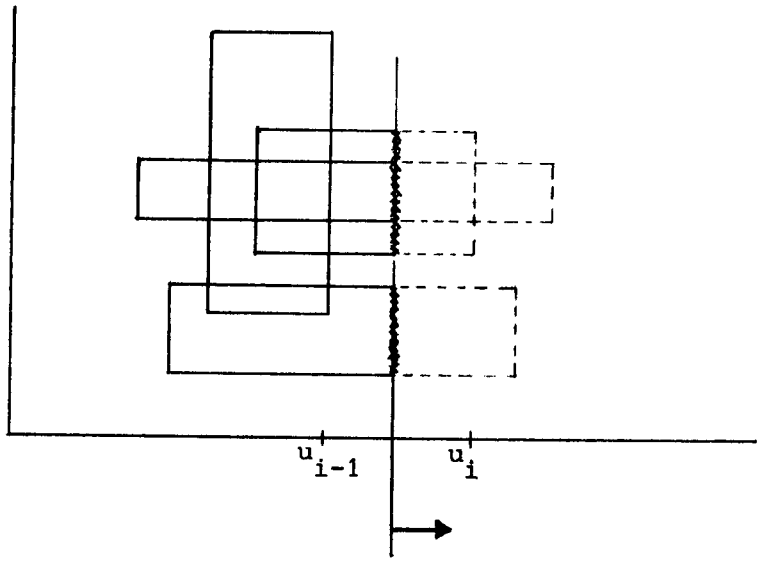


Figure 3.

When the scan-line is positioned at u_i, the segments "cut out" by intersecting rectangles can be of two different types (see figure 4). A segment is active when it is the "front" of a new rectangle or the current intersection with a non-ending rectangle. A segment is non-active if it is the "end" of a rectangle or a segment not on the present line of intersection. In figure 4 the segments α and β are active, γ and δ are non-active. We

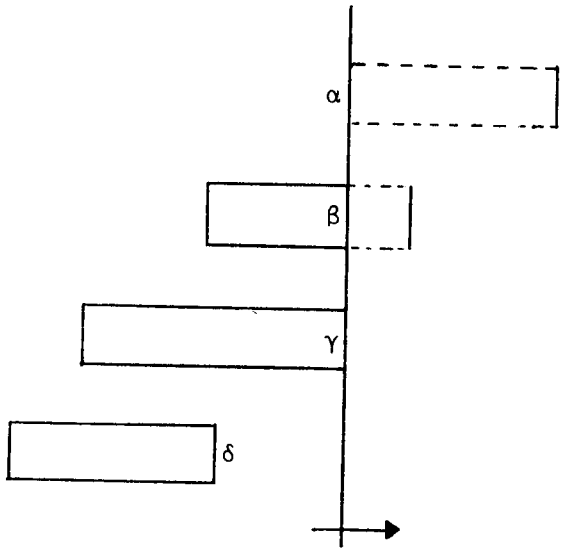


Figure 4.

shall only want to maintain the active segments on the current scan-line.

Definition. Let scan(i) be the scan-line as it is positioned at u_i, with its active segments. Let m(i) be the 1-dimensional measure of the active segments on scan(i).

Let M be the measure of the entire 2-dimensional region covered by the n rectangles. The use of the scan-line rests on the following simple, yet fundamental fact

Lemma 2.1. $M = \sum_{1}^{k-1} m(i).(u_{i+1}-u_i)$

It suggests a method to calculate M: as the scan-line moves right and makes its regular stops, keep track of the values of m(i) and accumulate the indicated sum!

When we move the scan-line we can indeed obtain scan(i+1) from scan(i), by deleting the segments of the rectangles we now "leave" (as they become non-active) and inserting the segments of rectangles we now "bump into" (as they become active). Fortunately we kept the necessary information about rectangles ending and beginning at u_{i+1} in a list right at this point, so we'll have no difficulty in determining which segments to delete or insert. For later use we might want to keep segments ordered by their left-point. Insertions and deletions could still be processed at O(log n) per transaction, to a total of O(n log n) as the scan-line moves right and each "side" of the rectangles gets inserted once and deleted later.

The straightforward way to compute m(i) would seem to apply the algorithm for the 1-dimensional measure problem to the currently active segments on scan(i). When segments are given by ordered left-points, it is easy to see that one sweep over scan(i) will do and that m(i) follows in a "mere" O(n) steps. It works, but can be costly. The total amount of work adds up to O(n log n + kn), which could be as bad as $O(n^2)$ when k gets large. On the face of it, we must have lost a great deal of efficiency when we decided to recalculate the entire measure on the scan-line at each scan-position. Intuitively we must be able to limit the work at u_{i+1} and keep it "proportional" to the number of segments deleted and inserted. We shall see how m(i+1) can be computed by applying a correction to m(i).

Observe that all segments which can appear on the scan-line must be composed of a collection of consecutive fragments $[v_j, v_{j+1}]$. We shall exploit the fragments as common, atomic building blocks. It would again be too expensive to tag all constituent fragments of active segments individually, as suggested in the structure of figure 5.a. The measure can be very easily maintained, but in the worst case we might have to place and (later) remove

about O(l.n), hence possibly up to $O(n^2)$, tags to do it right and we are still no better off than before. To save, we shall explain the idea of Bentley to mark "highest" nodes in a covering tree (figure 5.b).

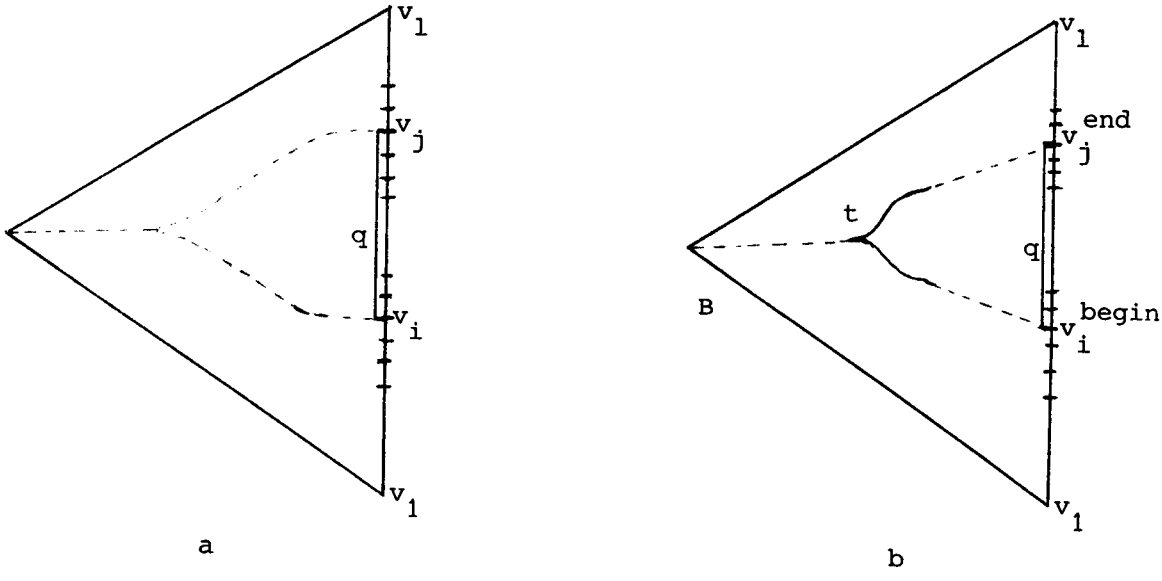


Figure 5.

Imagine that the (ordered) fragments are all entered as leaves in a balanced search tree B. Each v_j (except v_1 and v_l) will occur twice, once as the "end" (v_j^{end}) of the fragment $[v_{j-1}, v_j]$ and once as the "beginning" (v_j^{begin}) of the fragment $[v_j, v_{j+1}]$. Let a segment $q \equiv [v_i, v_j]$ be given. A node of B is said to be q-full if it covers a segment fully contained in q, q-partial if it is not q-full but has a son which is q-partial or q-full. Consider the search paths for v_i^{begin} and v_j^{end} through B (see figure 5.b). Let the node where the two paths diverge be t. All nodes encountered before t must have been q-partial (why?). Prune the two paths at the first q-full node they run into (see figure 6). The net result is that we can distinguish a "low line" from t to low-tip and a "high line" from t to high-tip. Note that low-tip and high-tip could very well coincide with t, the tip. In fact, if one does they both do (why?).

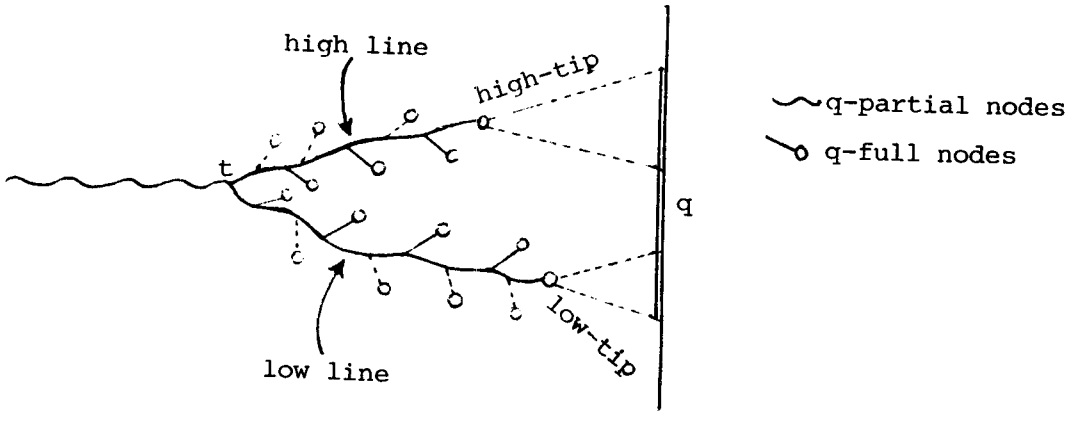


Figure 6.

Definition. The 1-umbrella of a segment q is the subtree of B consisting of

(i) the tip t,

(ii) the nodes on the high line from t to high-tip (all q-partial except high-tip, which is q-full) and the nodes on the low line from t to low-tip (likewise q-partial except for low-tip itself, which is q-full),

(iii) the q-full nodes which are directly connected to a node on the high or low line.

A 1-umbrella will have at most 4 nodes in each level, hence O(log n) nodes in all. It should be clear that the 1-umbrella can in fact be constructed in O(log n) time, but to prepare for a later generalization we shall prove it in a slightly different way.

Lemma 2.2. The 1-umbrella of a segment q can be built in O(log n) steps.

Proof

Start at the root of B and repeat the following routine for each node x visited

step 1

if x is q-full, then x is the umbrella and stop

step 2

let x_{low} and x_{high} be the low and high son of x, resp.
if only one is q-partial or q-full then
    x := the q-partial or q-full son
    goto step 1

step 3

{we get here when both sons are q-partial or q-full,
 hence when x is the tip}
initialize the umbrella by setting tip to x
let q_{low} be the subsegment of q covered by x_{low}
let q_{high} be the subsegment of q covered by x_{high}
call $L(q_{low}, x_{low})$
call $H(q_{high}, x_{high})$

step 4

stop

If we keep with each node the segment it covers, then the routine goes down the tree at O(1) cost per node visited. The routine L(q, x) builds the low-line of the umbrella by presenting a segment q to node x and carrying out the following steps thereafter

step 1

if x is q-full, then link x to the umbrella as its low-tip and stop

step 2

let x_{low} and x_{high} be the low and high sons of x, resp.
{note that x_{low} cannot be q-full!}
if x_{low} is q-partial, then
link x to the umbrella, x_{high} to x (as q-full node it is linked to the umbrella too)
$x := x_{low}$
goto step 2

step 3

{we get here when x_{low} is not q-partial and the low line must be continued towards x_{high}}
$x := x_{high}$
goto step 1

The routine H(q, x) likewise constructs the high line of the umbrella.

It easily follows that the total time spent is O(log n).

□

The idea of Bentley's segment tree is not to mark the individual fragments from which a segment is composed, but to mark the nodes of its 1-umbrella. We shall prove that the following information can be maintained in B, for any set of segments:

(*) with each node, the measure of the fragments in its subtree which are covered by 1-umbrellas through it or below it.

If (*) can be rightly maintained, then it follows that the measure of the entire set of segments currently in B can be read off at its root. Note that we do not explicitly keep track of what the current 1-umbrellas are. There is a hidden danger in not doing so. If we insert, we must be careful not to add "measure" to nodes which have already been declared full before. If we delete, we must be careful not to take "measure" away from nodes which still have other umbrellas through it or below it. But how do we avoid

errors! It appears to be sufficient to maintain the following additional information in B

(**) with each node, a count of the number of times it figures as a full node of some umbrella.

Let val(x) and count(x) be the values at node x as indicated in (*) and (**) respectively. It is important to realize that only the full nodes of 1-umbrellas really determine what we must compensate for at other nodes.

Lemma 2.3. One can insert a segment and maintain the information in B in only O(log n) steps.

Proof

Suppose we want to insert a segment q. We first determine its 1-umbrella (using O(log n) steps). See figure 7.

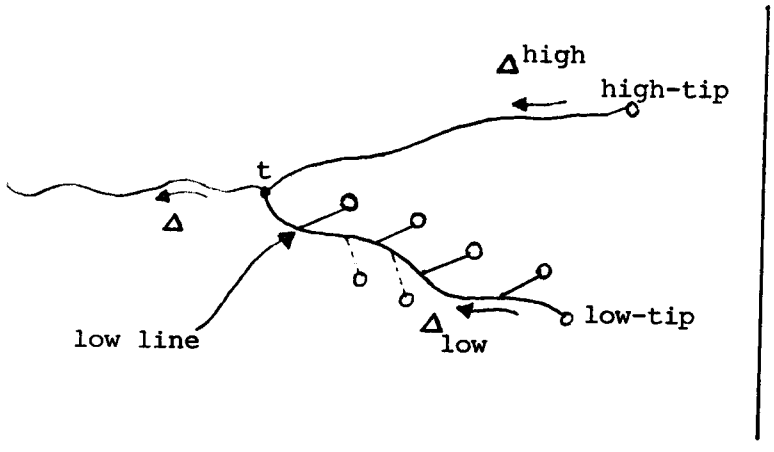


Figure 7.

We shall assume that t ≠ low-tip (hence ≠ high-tip), to treat the hardest case first.

It is useful to have a special routine DELTA(x) for updating full nodes of the umbrella first. It updates the measure information and the count at x (as side effects) and returns the increment in the measure of its subtree incurred by the added "full coverage".

```
procedure DELTA(x)
begin
   if val(x) = total size segment spanned by x' subtree
      then
      {entire segment already covered}
      count(x) := count(x)+1;
      return(0)
      else
```

```
            {the segment wasn't entirely covered}
            f := size segment - val(x)
            val(x) := size segment;
            count(x) := count(x)+1;
            return(f)
        fi
    end;
```

We shall begin by calling DELTA at low-tip and gradually work our way towards t.

```
    incr := DELTA(low-tip);
    x := father(low-tip);
    while x≠ t do
        begin
        {f will become the potential increment to be added in from
         the side}
        f := 0;
        if x has a q-full son y (nót low-tip)
            then
            {take care of it and see what increment it contributes}
            f := DELTA(y)
        fi;
        if val(x) = size segment spanned by x's subtree
            then
            {the maximum size has already been reached, if f≠ 0 it
             certainly must be because x did occur as a full node
             before}
            incr := 0
            else
            {it hasn't been, which means x has at most been partial
             before. The increments from both directions can validly
             be added in}
            incr := incr+f;
            val(x) := val(x)+incr
        fi;
        x := father(x)
    od;
```

Let the resulting value of incr along the low line be Δ_{low}. We can likewise determine the ultimate increment Δ_{high} resulting from working down the high

line. As we did above, the increments must be combined into t and the resulting total increment Δ be percolated upward towards the root. Note that Δ can be "killed" if we run into a node which has count ≥ 1.

The degenerate case in which t = low-tip follows likewise.

A (mind twisting) verification shows that (*) is maintained, (**) trivially is. The amount of work adds to a total of O(log n).

□

Lemma 2.4. One can delete a segment and maintain the information in B in only O(log n) stpes.

Proof

Suppose we wish to delete a segment q. Again we determine its 1-umbrella first, at a cost of only O(log n). The measure information, and the counts, must now be decremented at various nodes. Again we shall work down the low line and high line, then "combine" the decrements at t and percolate it towards the root. This time decrements can be killed too, if we run into nodes which keep having their count ≥ 1!

Again it is useful to have a separate routine DELTA(x) to update (decrement) the information at a full node.

```
procedure DELTA(x)
begin
    count(x) := count(x)-1;
    S := the sum of the val's of x' sons (0 if a leaf);
    if count(x) ≥ 1 or val(x) = S
       then
       {the size area covered below x remains unaffected by the
        removal of q}
       return(0)
       else
       {the size area covered is affected, x is no longer full and we
        must reset val(x)}
       f := val(x) - S;
       val(x) := S;
       return(f)
    fi
end;
```

To show how the decrements are calculated, we shall present the steps for working along the low line from low-tip towards t. It shows all the

necessary criteria to be applied, to determine whether to "kill" or to "propagate" the iterated decrement at a node.

```
decr := DELTA(low-tip);
x := father(low-tip);
while x≠ t do
      {we try to let f be the potential decrement to be added in
       from the side, caused by the removal of a full node}
      f := 0;
      if x has a q-full son y
         then
         f := DELTA(y)
      fi;
      {now we only need to know that x has count ≥ 1 in order to
      kill the decrement communicated from its sons. Otherwise x is
      at best partial, and the total decrement can be rightly added
      in to modify its val}
      if count(x) ≥ 1
         then
         decr := 0
         else
         decr := decr+f;
         val(x) := val(x)-decr
      fi;
      x := father(x)
od;
```

One may verify that (*) and (**) are correctly maintained. The entire process takes only O(1) time per visited node, hence the total count adds up to O(log n).

□

The reader may find it instructive to compare the steps taken to insert and delete a segment, respectively. The understanding of the routines will be aided by observing that the following, additional property remains invariant at each node x

(***) count(x) ≥ 1 or val(x) is the sum of the val's of x' sons (0 if x is a leaf).

The deletion routine should be applied only for segments of which we know they have previously been inserted. Fortunately our main application will do exactly this.

Theorem 2.5. The measure problem for a set of n rectangles in the plane can be solved in only O(n log n).

Proof

Returning to the discussion following lemma 2.1, it is clear how we should proceed. The currently active segments of scan(i) should be put in a segment tree. The value of m(i) is readily available at the root. If the scan-line moves one position, then we should delete the segments which have now become inactive and insert the segments which now become active. There will be 2n transactions of this sort in all, at a cost of O(log n) each.

□

3. The measure problem in three and more dimensions.

Let us now consider the (apparently) harder problem to determine the measure of a set of n, possibly overlapping ranges in d-space for $d > 2$. Given the O(n log n) solution in two dimensions, it readily follows that one can do it with $O(n^{d-1} \log n)$ steps in d dimensions (cf. Bentley [1]). It is important to note that any improvement over this bound in some dimension will imply a better bound for all higher dimensions too.

Theorem 3.1. Suppose that the α-dimensionsl measure problem can be solved in O(g(n)) steps, for some α and $g(n) \geq n$. Then one can solve the d-dimensional measure problem in $O(n^{d-\alpha} g(n))$ steps for any $d \geq \alpha$.

Proof

By bootstrapped induction it is sufficient to prove the statement for $d = \alpha+1$ only. Let a set of n $(\alpha+1)$-dimensional ranges be given, each range by the concrete bounding values for its coordinates. Sort the bounding values of the first coordinates and weed out duplicates, to obtain a list

$$u_1 < u_2 < \ldots < u_k$$

while, as for the rectangle case, with each u_i a table is kept containing the names of those ranges (viz. their projection) which end or begin at this point. The α-dimensional hyperplane through u_i is the perfect analogue of our earlier scan-line. Let scan(i) be the scan-plane through u_i with all active intersections on it, let m(i) be the measure of these intersections viewed as α-dimensional ranges. The "total" volume we wish to compute is again determined by $\sum_{1}^{k-1} m(i)(u_{i+1}-u_i)$.

It should be obvious that the total measure can be computed by solving $O(n)$ instances of the α-dimensional measure problem and only $O(n)$ additional steps. The number of steps adds up to $O(n \log n) + O(n.g(n))$, which is $O(n.g(n))$.

□

Theorem 3.1 immediately explains how the $O(n^{d-1} \log n)$ algorithm for the d-dimensional measure problem is obtained. To improve it, we shall examine the 3-dimensional problem more carefully. The acclaimed $O(n^2 \log n)$ algorithm in 3 dimensions resulted (cf. the proof of theorem 3.1) by moving a scan-plane from left to right over the x-axis and solving the 2-dimensional measure problem for the active "projections" each time the scan-line was positioned at another u_i. See figure 8.

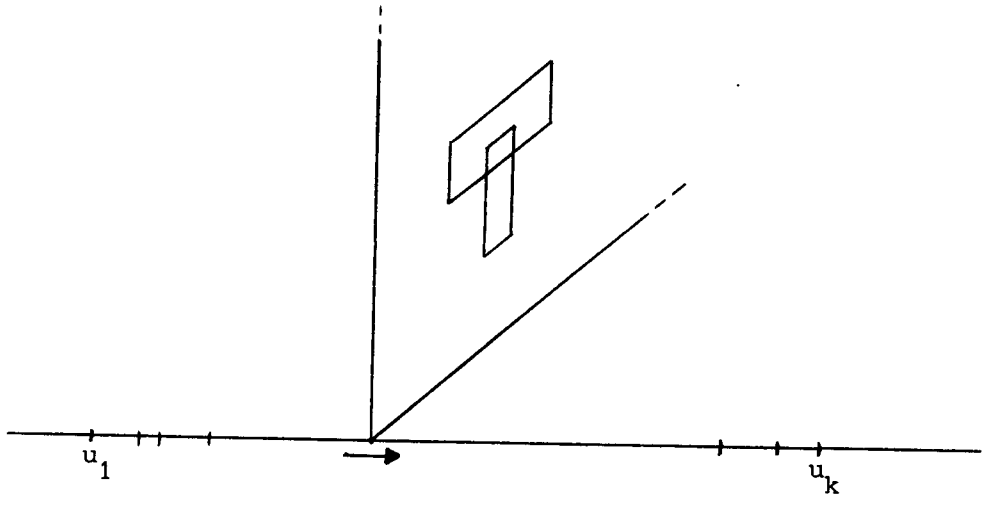


Figure 8.

Again, efficiency seems to be lost by not maintaining information from a previous scan-position. Note that we made the very same observation when we considered our "initial" solution for the rectangle measure problem in section 2. We shall try to resolve the inefficiency by following the same recipe, making use of a representation of all projected ranges by planar fragments.

The first step will be to determine what the fragments are. Thus, imagine that all 3-dimensional ranges are projected on an abstract y-z plane. Separately sort the bounding values in the y-coordinate and in the z-coordinate, and eliminate duplicates in each list. The result is a 2-dimensional roster (see figure 9), precisely delineating all pertinent fragments.

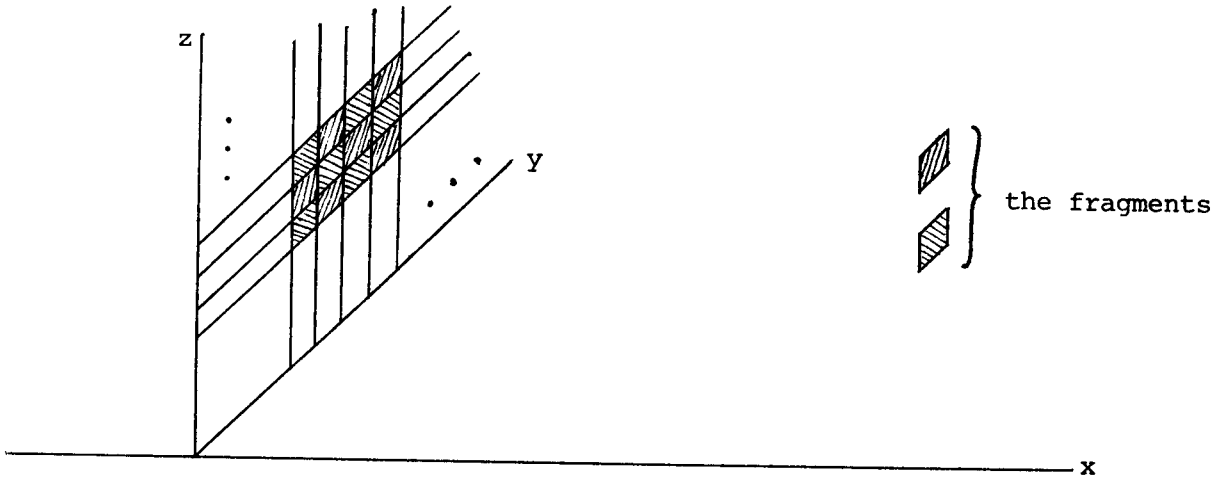


Figure 9.

It would still not resolve anything if we would now mark the presence of a rectangle by "tagging" all individual fragments it is composed of. There are $O(n^2)$ distinct fragments and we could end up tagging all of them for each rectangle in the projection! Thus we need an appropriate version of the umbrella concept for plane rectangles. To prepare for it, we note that it is indeed possible to build a natural balanced tree with the fragments at its leaves. The reader is advised to familiarize him/herself with quad-trees (see Finkel and Bentley [3]), which will be just the tool we need.

To grasp the idea of a quad-tree, consider a rectangular area which is subdivided once (figure 10). We can set up a "root" node A representing the rectangle and provide it with 4 sons, each representing a distinct rectangular part. If we subdivide each rectangular part once again, then each son will "split" in turn and obtain 4 sons also. And so on.

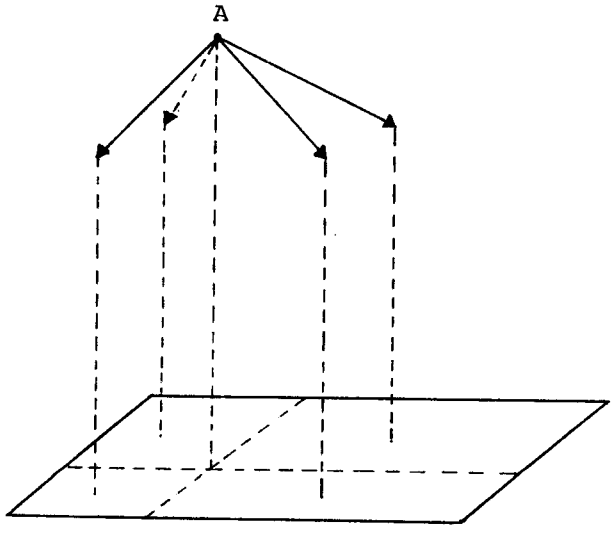


Figure 10.

If we continue subdividing parts h times, then we get a complete quaternary tree C with h levels and (hence) 4^h rectangle fragments as leaves.

Returning to the collection of up to n^2 fragments we determined as atomic parts of the measure problem, let us add enough dividing points along the y- and z-axis to make a total of some 2^h+1 points on both. Clearly we can keep $h \leq \lceil \log 2n \rceil = \lceil \log n \rceil + 1$. It will make for a (small) extension of the original set of fragments, but the succinct advantage is that we can now build a nice perfect quad-tree on top of them. Note that C can have no more than

$$4^{\lceil \log n + 1 \rceil} = O(n^2)$$

leaves in all. The quad-tree can be built in a number of steps proportional to n log n (for sorting the y- and z-points) + n^2 (to set up the node linkages), hence in $O(n^2)$.

If we keep with each node of the quad-tree the information telling what rectangular region it covers, then we can very easily determine for a given rectangle q whether a visited node is q-partial or q-full. It will be of help in determining the "umbrella" of a rectangle in the quad-tree, which we shall do in very much the same way as the construction in lemma 2.2.

To determine the 2-umbrella of a rectangle q we proceed as follows. Start at C's root and go down the quad-tree as long as there is exactly one q-partial or q-full son ahead. The first node you hit which is q-full or which has ≥ 2 q-partial or q-full sons, will be the umbrella's tip. If the tip is q-full, then the umbrella is completed. Otherwise link all q-full sons to the tip and, after linking them to the tip also, proceed with the following routine for each of the q-partial sons

```
procedure UMB(x)
begin
    {x has been linked to the umbrella}
    link all q-full and q-partial sons of x to x, making them
    part of the umbrella;
    call UMB for each of the q-partial sons
end
```

The resulting structure will be a quaternary, connected subtree embedded in C, which "resembles" an umbrella. It will come as a surprise that 2-umbrellas can't be very big, as the following lemma shows.

Lemma 3.2. The 2-umbrella of a rectangle can be determined in only O(n) steps and (hence) has at most O(n) nodes.

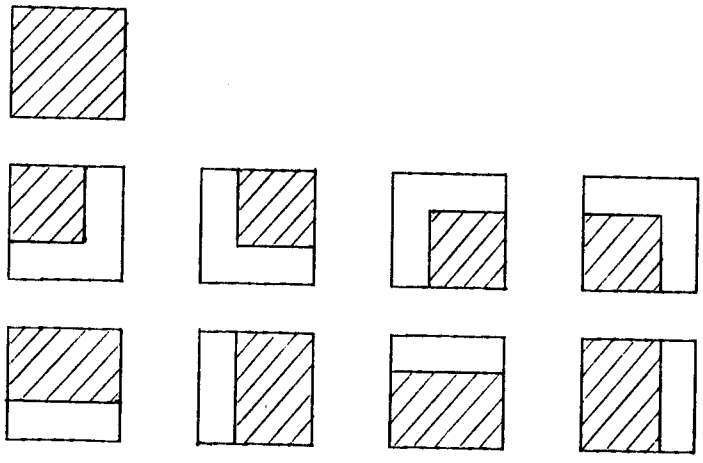
Proof

We shall argue that the given construction must terminate within O(n) steps. When the tip of the umbrella is reached (after at most log n steps), we know q can be embedded in the rectangular region below in only one of three different ways:

type A

type B

type C



For our analysis it is unimportant to keep track of the size of the area q occupies in each of these corner-types. All we need to know is that, at each node visited by UMB, q's embedding can be classified as being of one of these three types.

One may observe that

(i) nodes of type A are "terminal",

(ii) nodes of type B (as UMB inspects its 4 parts) can lead into at most 1 son of type A, 1 of type B and 2 sons of type C,

(iii) nodes of type C can likewise lead into at most 2 sons of type A and 2 of type C again.

We might conveniently represent the "growth" of the number of nodes visited by UMB as a parallel rewriting system with axiom A, B or C (depending on the cornertype at the tip) and production rules

$$A \rightarrow A$$

$$B \rightarrow A\,B\,C\,C$$

$$C \rightarrow A\,A\,C\,C$$

It models the worst case only and the real number of nodes visited will not be as large as these growth rules let us predict. Observe that there can be at most one B node at each instant in time, that the number of A's cannot be larger than twice the number of C's there can be in the worst case. This number of C's is easily seen to be

$$\leq 2 \cdot 2^{h} = O(n)$$

where we note that UMB's recursion cannot go deeper than the quad-tree's depth $h \leq \lceil \log n \rceil + 1$.

This will do to obtain the O(n) bounds claimed.

□

From this point on, we can proceed in exactly the same way as we did for segment trees! We shall demonstrate (this time only briefly) that the following information can be maintained in the quad-tree, after a set of rectangles has been inserted

(*) with each node x a value val(x), which is the measure of the total region covered by 2-umbrellas through it or below it,

(**) with each node x a number count(x), which keeps track of the number of times x currently figures as a full node of some 2-umbrella.

Again the full nodes carry the show. A quad-tree as constructed, augmented with the val- and count-information at the nodes, will be called a rectangle tree.

Lemma 3.3. One can insert a rectangle into the rectangle tree and maintain (*) and (**) in only O(n) steps.

Lemma 3.4. One can delete a (currently present) rectangle from the rectangle tree and maintain (*) and (**) in only O(n) steps.

In both lemmas we first build the (quaternary) 2-umbrella of the rectangle we want to insert or delete in O(n) steps and then update the information at each node during a pre-order traversal of the umbrella in exactly the same way as in lemmas 2.3 and 2.4. After each transaction the total "measure" of the current set of rectangles can be read off at the root.

We conclude

Theorem 3.5. The 3-dimensional measure problem can be solved in only $O(n^2)$ steps.

Proof

Setting up the initial quad-tree for the fragments in the scan-plane takes $O(n^2)$ steps. If we move the scan-plane across the x-axis, then we need to make no more than 2n insertions (of active rectangles) and deletions (of rectangles which become inactive) as we go along, at a cost of O(n) each.

□

As an immediate corollary to theorems 3.1 and 3.5 we get the main result of this paper.

Theorem 3.6. The measure of a set of n rectangular ranges in dimension $d \geq 3$ can be computed in only $O(n^{d-1})$ steps.

This result seems to establish the best currently known bound on Klee's rectangular measure problem.

4. Final comments.

The main result of our analysis is an improvement of Bentley's $O(n^2 \log n)$ bound for the 3-dimensional measure problem to $O(n^2)$. We argued that the d-dimensional measure problem for $d \geq 3$ could be solved in $O(n^{d-1})$ steps, essentially by extending the efficient solution in 3 dimensions to higher dimensions in the very same blunt manner we previously criticized for its lack of efficiency.

It would seem that, just as in the 2- and 3-dimensional cases, we should build a proper analogue of a rectangle tree for the (d-1)-dimensional fragments in a (d-1)-dimensional scan-plane and represent ranges dynamically by means of a (d-1)-umbrella. The efficiency of this method will depend very heavily on our estimates of the umbrella sizes to be expected. We know of no better bound than $O(n^{d-2})$ for it, as is also confirmed by a very similar result Lee and Wong [6] obtained for the number of nodes visited by range querying in a balanced quad-tree (which is very much the same problem).

Thus the question remains whether our bounds and methods are optimal or not. In fact, we do not even know whether the n^2 bound for 3 dimensions is best or not.

5. References

[1] Bentley, J.L., Solutions to Klee's rectangle problems, unpublished manuscript, Dept. of Computer Science, Carnegie-Mellon University (1977).

[2] Bentley, J.L. and D. Wood, An optimal worst-case algorithm for reporting intersections of rectangles, Techn. Rep. 79-CS-13, Unit for Computer Science, McMaster University, Hamilton, Ont. (1979).

[3] Finkel, R.A. and J.L. Bentley, Quad-trees: a data structure for retrieval on composite keys, Acta Informatica 4 (1974) 1-9.

[4] Fredman, M. and B. Weide, On the complexity of computing the measure of $U[a_i,b_i]$, C.ACM 21 (1978) 540-544.

[5] Klee, V., Can the measure of $U[a_i,b_i]$ be computed in less than O(n log n) steps, Research Probl. Sect., Amer. Math. Monthly 84 (1977) 284-285.

[6] Lee, D.T. and C.K. Wong, Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees, Acta Informatica 9 (1977) 23-29.

[7] Vaishnavi, V. and D. Wood, Data structures for the rectangle containment and enclosure problems, Techn. Rep. 79-CS-19, Unit for Computer Science, McMaster University, Hamilton, Ont. (1979).