New upper bounds in Klee's measure problem
(extended abstract)

Mark H. Overmars and Chee-Keng Yap

RUU-CS-88-22
May 1988

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

New upper bounds in Klee's measure problem
(extended abstract)

Mark H. Overmars and Chee-Keng Yap

RUU-CS-88-22
May 1988

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

New upper bounds in Klee's measure problem
(extended abstract)

Mark H. Overmars and Chee-Keng Yap

Technical Report RUU-CS-88-22
May 1988

Department of Computer Science
University of Utrecht
P.O.Box 80.089
3508 TB Utrecht
the Netherlands

New upper bounds in Klee's measure problem (Extended Abstract)

Mark H. Overmars and Chee-Keng Yap

May 1988

Abstract

We give new upper bounds for the measure problem of Klee which significantly improve the previous bounds for dimensions greater than 2. We obtain an $O(n^{d/2} \log n, n \log n)$ time-space upper bound to compute the measure of a set of n boxes in Euclidean d-space. The solution requires several new ideas including application of the inclusion/exclusion principle, the concept of trellises, streaming, and a partition of d-space.

1 Introduction

Around 1977, Klee [5] posed the 'measure problem': given a set of n intervals, find the length of their union. He gave an $O(n \log n)$ time solution and asked if this was optimal. This generated considerable interest in the problem, and shortly after, Fredman and Weide [4] proved that $\Omega(n \log n)$ is a lower bound under the usual model of computation. Bentley [2] considered the natural extension to d-dimensional space where we ask for the d-dimensional measure of a set of d-rectangles. He showed that the $O(n \log n)$ bound holds for $d = 2$ as well, and for $d > 2$, the result generalizes to an upper bound of $O(n^{d-1} \log n)$. Thus the results are optimal for $d = 1, 2$. We refer to the book [6] for an account. Concerning these results for $d \geq 3$, Preparata and Shamos remarked in their book ([6] pp.328-9):

> What is grossly unsatisfactory about the outlined method for $d \geq 3$ is the fact that there is a "coherence" between two consecutive sections in the sweep that we are unable to exploit. ... Although it seems rather difficult to improve on this result, no conjecture about its optimality has been formulated.

The only progress made since was a small improvement by van Leeuwen and Wood [6,7] who removed the $\log n$ factor from Bentley's upper bound for $d \geq 3$. The test case seems to be $d = 3$: is $O(n^2)$ really necessary for computing the volume of a set of n boxes in 3-space? In this paper we show that $O(n^{1.5} \log n)$

suffices. This immediately implies that in d-dimensions ($d \geq 3$), the bound becomes $O(n^{d-1.5} \log n)$.

The idea is to use a plane-sweep approach and dynamically maintain the measure of a set of 2-dimensional rectangles in time $O(\sqrt{n})$ per update.

Such a result means that we can maintain the area of a set of rectangles *implicitly* without having to represent the full boundary structure. This is because any explicit representation of the boundary of n rectangles requires $\Omega(n^2)$ time in the worst case because of the simple 'trellis' example: it consists of n long vertical rectangles which are pairwise disjoint, superposed on n long horizontal rectangles also disjoint among themselves.

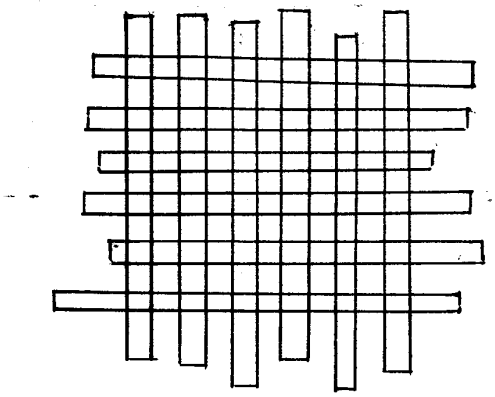

Figure 1: Trellis

The first idea is to exploit the regularity of such trellis structures by maintaining only $O(n)$ amount of information (at the boundary of the box containing the trellis) to keep track of the area of the trellis rectangles. Of course, a union of rectangles is too irregular to be consistently exploited in this way, so the next idea is to partition the plane into a collection of trellises. Using a generalization of the k-d tree of Bentley[1], we are able to form such a partition with only $O(n)$ trellises each of size $O(\sqrt{n})$. As we shall see, extending this to higher dimensions requires a partition with interesting properties that might be useful for other applications as well.

The rest of this paper is organized as follows: Section 2 describes the basic space sweep algorithm we use and introduces the generalized k-d tree for storing the boxes. Section 3 contains the solution to the 3-dimensional measure problem. Section 4 generalizes this solution to a d-dimensional method, using an interesting partition scheme of the d-dimensional space. This results in a $(O(n^{d/2} \log n, n^{d/2})$ time-space upper bound. In Section 5 we exploit a streaming technique of Edelsbrunner and Overmars[3] to reduce the amount of storage required to $O(n \log n)$ only, for any dimension d. Finally, in Section 6 some conclusions, extensions and open problems are given.

During the paper we will use the following terminology. A d-box is the cartesian product of d intervals in d-dimensional space. An i-boundary of a d-box is a boundary hyperplane that is perpendipular to the i-th coordinate axis. Each d-box

has two i-boundaries for $1 \leq i \leq d$. The i-interval is the projection of the d-box on the x_i-axis.

Definition 1.1 *A rectangle R_1 is said to* partially cover *R_2 if the boundary of R_1 intersects the interior of R_2.*

Definition 1.2 *For two d-boxes R_1 and R_2 let R be their intersection. We say that R_1 is an* i-pile *w.r.t. R_2 iff for all $1 \leq j \leq d$ with $j \neq i$ the i-interval of R is equal to the i-interval of R_2.*

In other words, in each direction, except direction i, R_1 completely covers R_2. i-piles will play an important role throughout this paper.

2 General framework

The basic method for solving the d-dimensional measure problem is as follows. Let V be the set of n d-boxes for which we want to compute the measure. Let $V' = \{a_1, ..., a_{n'}\}$ be the set of all different x_d-coordinates of vertices of the boxes, i.e., all different endpoints of d-intervals. We sort the boxes both by left and right d-boundary. We solve the measure problem using a plane sweep approach. We sweep a hyperplane along the d-th coordinate axis stopping at each value in V'. During the sweep we maintain the $(d-1)$-dimensional measure of the boxes intersected by the sweep plane. The algorithm looks as follows.

S:=$\emptyset$;
$MEAS$:=0;
FOR i:=1 TO $n'-1$ DO
    Insert all d-boundaries of boxes that start at a_i in S;
    M:=$(d-1)$-dimensional measure of boxes in S;
    $MEAS$:=$MEAS$+$(a_{i+1} - a_i)$*M;
    Delete all d-boundaries of boxes that end at $a_i + 1$ from S
END;

At termination $MEAS$ will contain the measure of the set of boxes. S will be a dynamic data structure for maintaining the $(d-1)$-dimensional measure. If insertions and deletions in S can be performed in time $F_{d-1}(n)$ the method will take time $O(n \log n + nF_{d-1}(n))$. This approach is due to Bentley.

To maintain the measure of the set of boxes intersected by the sweep-plane we introduce a generalized version of the k-d tree.

Definition 2.1 *A d-dimensional orthogonal partition tree is a balanced binary tree. With each internal node δ is associated a part C_δ of the d-dimensional space, with the following properties:*

- *C_{root} is the whole d-dimensional space.*
- *For each node δ C_δ is a (possibly unbounded) d-box.*
- *For each node δ with sons δ_1 and δ_2, $Int(C_{\delta_1}) \cap Int(C_{\delta_2}) = \emptyset$ and $C_{\delta_1} \cup C_{\delta_2} = C_\delta$.*

C_δ will be called the region associated with δ. When δ is a leaf we refer to C_δ as a cell. It immediately follows that for each full level of the orthogonal partition tree all regions are essential disjoint and their union is the d-dimensional space. From now on we drop the qualifying word "orthogonal" and speak only of "partition trees", which is not to be confused with the "non-orthogonal" partition trees of e.g. Willard[9] and Welzl[8].

To use partition trees for maintaining the measure of a set of d-boxes we store the following extra information in the partition tree: With each leaf δ we store all boxes that intersect C_δ but do not cover the region associated with the father of δ. For each internal node δ we store a counter TOT that contains the number of d-boxes that completely cover C_δ but only partially cover $C_{father(\delta)}$. Finally, with each node δ we associate a field M that is defined as follows: If δ is a leaf M contains the measure of the boxes stored at δ restricted to C_δ. Otherwise, if $TOT > 0$ then M is the measure of C_δ, otherwise $M = M_{lson(\delta)} + M_{rson(\delta)}$. It is easy to verify that M_{root} is the measure of the set of d-boxes.

To maintain the measure in a dynamically changing set we have to be able to insert and delete d-boxes in the partition tree. The basic insertion algorithm is the following:

```
PROCEDURE Insert(box,δ);
    IF δ is a leaf THEN
        Store box at δ;
        Compute M_δ
    ELSIF box covers C_δ THEN
        INCR(TOT_δ);
        M_δ:=measure of C_δ
    ELSIF box partially covers C_δ THEN
        Insert(box,lson(δ));
        Insert(box,rson(δ));
        IF TOT_δ > 0 THEN
            M_δ:= measure of C_δ
        ELSE
            M_δ:=M_lson(δ) + M_rson(δ)
        END
    END;
```

The routine is called as Insert(box,$root$). The deletion routine will be similar. Note the similarity with the methods of Bentley[2] and van Leeuwen and Wood[7] for the 1- and 2-dimensional case. It is immediately clear that the amount of time

required depends on the number of nodes visited and the amount of time required for computing the measure at the leaves. In the sequel of this paper we will show that partition trees exist in which both are small.

3 Dynamic measure problem in two dimensions

To illustrate the general solution we will develop in the next section, we first concentrate on the 3-dimensional measure problem. Solving the 3-dimensional problem means that we have to design a 2-dimensional partition tree with good performance. To obtain such a partition tree we first define a subdivision of the plane in rectangular cells with some interesting properties.

Let V be the set of the rectangles that will be inserted and deleted in the partition tree. Let V_1 be the set of different x_1-coordinates of 1-boundaries of rectangles. First we split the x_1-axis into $\sqrt{n}$ intervals such that each interval contains $\leq 2\sqrt{n}$ coordinates. This defines $\sqrt{n}$ slabs in the plane. Each slab s will be split by horizontal line segments into a number of cells. Let V_s^1 be the set of rectangles that have a 1-boundary inside s. Let V_s^2 be the set of rectangles that only have a 2-boundary in s. (Note that the size of V_s^1 is bounded by $2\sqrt{n}$ but the size of V_s^2 can be almost n.) We draw a line segment through each 2-boundary of a rectangle in V_s^1. Moreover, we draw a line segment through each $\sqrt{n}$-th 2-boundary of a rectangle in V_s^2. In this way s is partitioned into $\leq 6\sqrt{n}$ rectangular cells.

Lemma 3.1 *The partition has the following properties:*

1. *There are $O(n)$ cells.*
2. *Each rectangle of V partially covers at most $O(\sqrt{n})$ cells.*
3. *No cell contains vertices in its interior.*
4. *Each cell has at most $O(\sqrt{n})$ rectangles partially covering it.*

Proof. Can easily be verified. □

We will use the cells of this partition as leaves of the partition tree. It is easy to see how the rest of the tree can be built on top of it. As long as there are more than one cell in a slab, merge neighbouring cells into one (creating the father of the two cells). After that merge neighbouring slabs. Hence, in the resulting tree, the regions associated with nodes in the upper levels are split on x_1-coordinate. The regions associated with nodes in the lower levels are split on x_2-coordinate. Details are left to the reader.

Lemma 3.2 *Let V be a set of n rectangles in the plane. There exists a partition tree for storing any subset of V such that*

1. *The tree has $O(n)$ nodes.*

2. *Each rectangle is stored in $O(\sqrt{n})$ leaves.*
3. *Each rectangle influences $O(\sqrt{n}\log n)$ TOT fields.*
4. *No cell of a leaf does contain vertices of rectangles in the inside.*
5. *Each leaf stores no more than $O(\sqrt{n})$ rectangles.*

Proof. Properties 1, 2, 4 and 5 follow immediately from the above lemma. The third property follows from the first two. If the tree has $O(n)$ nodes its depth is bounded by $O(\log n)$. When a rectangle influences the TOT field of a node δ it intersects $C_{father(\delta)}$, and there must be a leaf below $father(\delta)$ that is intersected by the rectangle. Hence, the number of internal nodes intersected by a rectangle is bounded by $O(\log n)$ times the number of leaves where the rectangle is stored. As a result the rectangle can only influence that number of TOT fields. □

It remains to show how the measure at a leaf is maintained when inserting and deleting rectangles. To this end we use the inclusion/exclusion principle. Note that, due to property 4, the rectangles stored at a leaf δ are 1-piles or 2-piles w.r.t. C_δ. In other words, they form a trellis. The measure of such a trellis can be maintained in the following way. Let V_1 be the projection of the 1-piles on the x_1-axis and let V_2 be the projection of the 2-piles on the x_2-axis. Let M_1 be the (1-dimensional) measure of V_1 and M_2 the (1-dimensional) measure of V_2. Assume that the cell C_δ has size $L_1 \times L_2$. Now it is easy to see that the measure of the trellis is $M_1 * L_2 + M_2 * L_1 - M_1 * M_2$. Hence, we just have to maintain the 1-dimensional measure of V_1 and V_2. For this we can use a simple segment tree that uses linear storage and maintains the measure in time $O(\log n)$ per insertion and deletion.

Theorem 3.3 *The measure of a set of n 3-boxes in 3-dimensional space can be computed in time $O(n\sqrt{n}\log n)$ using $O(n\sqrt{n})$ storage.*

Proof. We use the plane sweep approach and maintain the partition tree described above. To insert or delete a rectangle we have to update $O(\sqrt{n}\log n)$ TOT fields. This takes time $O(\sqrt{n}\log n)$. Next, we have to insert or delete the rectangle at $O(\sqrt{n})$ leaves. At each such leaf this causes an insertion or deletion in a segment tree which takes $O(\log n)$. Hence, the total update time of the partition tree is $O(\sqrt{n}\log n)$.

The bound on the amount of storage required follows from the fact that the tree itself takes $O(n)$ storage and each leaf stores $O(\sqrt{n})$ information. □

4 Dynamic measure in multi-dimensional space

We will now generalize this method to d-dimensional space. To this end we will describe a d-dimensional partition tree, based on a cell decomposition of the d-dimensional space.

Let V be the set of all d-boxes that will be inserted or deleted in the partition tree. Let V_1 be the set of different x_1-coordinates of 1-boundaries of boxes. We split the x_1-axis in $\sqrt{n}$ intervals, each containing $\leq 2\sqrt{n}$ coordinates. This splits the d-dimensional space in $\sqrt{n}$ slabs. For each slab s let V_s be the set of d-boxes that partially cover s. We split V_s in two subsets: V_s^1 of d-boxes that have a 1-boundary inside s and V_s^2 of d-boxes that do not have a 1-boundary inside s. Note that $|V_s^1| \leq 2\sqrt{n}$. Each cell (slab) s we now split with respect to second coordinate. We split it at the 2-boundaries of each d-box in V_s^1 and we split it at every $\sqrt{n}$-th 2-boundary of d-boxes in V_s^2. As a result we split each slab s into $O(\sqrt{n})$ cells. For each cell c let V_c be the set of d-boxes that partially cover c. We again split V_c into two subsets: V_c^1 of boxes that have a 1- or 2-boundary inside c and V_c^2 of boxes that do not have a 1- or 2-boundary inside c. (Note that there are no boxes that have both a 1- and 2-boundary inside c.) Again $|V_c^1| = O(\sqrt{n})$. We split c into subcells with respect to the third coordinate. Again, we split at each 3-boundary of boxes in V_c^1 and at every $\sqrt{n}$-th 3-boundary of boxes in V_c^2. In this way we continue for all coordinates.

Lemma 4.1 *The partition has the following properties:*

1. *There are $O(n^{d/2})$ cells.*
2. *Each d-box of V partially covers at most $O(n^{(d-1)/2})$ cells.*
3. *Each cell only contains piles in its interior.*
4. *Each cell has at most $O(\sqrt{n})$ d-boxes partially covering it.*

Proof. See the full paper for a precise proof. □

We will use the cells of this partition as leaves of the partition tree. It is easy to see how the rest of the tree can be built on top of it. See the full paper for a detailed description of the structure.

Lemma 4.2 *Let V be a set of n d-boxes in d-dimensional space. There exists a partition tree for storing any subset of V such that*

1. *The tree has $O(n^{d/2})$ nodes.*
2. *Each d-box is stored in $O(n^{(d-1)/2})$ leaves.*
3. *Each d-box influences $O(n^{(d-1)/2} \log n)$ TOT fields.*
4. *Each cell of a leaf only contains piles.*
5. *Each leaf stores no more than $O(\sqrt{n})$ d-boxes.*

Proof. Properties 1, 2, 4 and 5 follow immediately from the above lemma. The third property follows from the first two as the depth is again bounded by $O(\log n)$. □

It remains to show how the measure at a leaf is maintained when inserting and deleting d-boxes. To this end we again use the inclusion/exclusion principle. As stated in property 4, the d-boxes stored at a leaf δ are piles and form a d-dimensional trellis. Let V_i be the projection of the i-piles on the x_i-axis for each $1 \leq i \leq d$. Let M_i be the 1-dimensional measure of V_i. Let L_i be the length of C_δ in direction x_i. The following result is easy to proof:

Lemma 4.3 *The measure of the trellis is*

$$\sum_{1 \leq k \leq d} (-1)^{k+1} S_k$$

where

$$S_k = \sum_{1 \leq j_1 < \ldots < j_k \leq d} \prod_{1 \leq i \leq k} M_{j_i} \prod_{l \neq j_i \text{ for any } i} L_l$$

Although this might look quite complicated it is simply the inclusion/exclusion principle. When M_i is known for each i the measure can be computed in constant time (assuming d is a constant).

Hence, we just have to maintain the 1-dimensional measure of V_i for each i. For this we use d segment trees, one for each dimension.

Lemma 4.4 *Updates in the d-dimensional partition tree take time $O(n^{(d-1)/2} \log n)$ and the tree uses $O(n^{(d+1)/2})$ storage.*

Proof. Follows from the above lemma's. □

Theorem 4.5 *The measure of a set of n d-boxes in d-dimensional space can be computed in time $O(n^{d/2} \log n)$ using $O(n^{d/2})$ storage.*

Proof. We use the plane sweep approach and maintain a $(d-1)$-dimensional partition tree. So we have to perform $O(n)$ updates, each taking time $O(n^{(d-1-1)/2} \log n)$. The time bound follows. According to the preceeding lemma, the structure uses $O(n^{(d-1+1)/2})$ storage. □

5 Reducing the amount of storage

In this section we will show how the amount of storage required can be reduces to $O(n \log n)$. To this end we use an instance of the streaming technique introduced in Edelsbrunner and Overmars[3]. We will only briefly describe the ideas and results. See the full paper for a more extensive description.

The idea of streaming is the following: Beforehand we know what updates have to be performed and in what order. We can view the space sweep method as traversing in time (being the d-th coordinate). Each update in the structure has to be performed at a specific moment in time. Before each update we check what the current

measure is and we multiply it by the time passed since the last update. Rather than building the structure and performing the updates one after the other, we will perform them simultaneously and construct parts of the data structure when we need them.

To formalize this, at any moment we are given a sequence of updates L over time and a region of the space C. This region corresponds to some node in the tree and L is the sequence of updates that will pass through this node. With each update in L we have stored the time at which it has to be performed. In the beginning C is the whole d-space and L is the complete list of updates. A counter $MEAS$ will be used to collect all the measure found. In the beginning it will be set to 0.

The technique now works as follows: When all d-boxes in L are piles with respect to C (i.e., we are at a leaf in the partition tree) we construct d segment trees. We perform all the updates on the segment trees and compute the measure in the cell after each update. These measures we multiply with the time passing to obtain the total measure in C over time. This measure we add to $MEAS$. This will take time $O(|L| \log n)$.

When not all boxes are piles (i.e., we are at an internal node) we first compute during which periods of time C will be completely covered by one box. (This corresponds to the time when $TOT \neq 0$.) We multiply the measure of C with this total amount of time and add it to $MEAS$. Next we change time by collapsing the covered periods into a single moment, performing all the updates in that period at the same moment. (This is neccessary to avoid that measure will be reported in these periods later again.) Boxes that are now inserted and deleted at the same moment are removed from L. Next we split C into two cells C_1 and C_2 in the same way in which it would have been split in the partition tree. (How to split can be determined by presorting. We won't go into details here.) We make two lists L_1 and L_2 out of L containing the updates that influence C_1 and C_2. We discard C and L and call the routine recursively with C_1 and C_2. The work can be performed in $O(|L|)$ time.

The method does essentially the same work us the original technique in which all updates are performed one after the other. In fact, it is more efficient because of two reasons. When the whole list consists of piles we immediately solve the problem rather than splitting till the list contains less than $\sqrt{n}$ boxes. Secondly, we don't consider boxes anymore when during their whole period of existance they are covered by some other box.

Theorem 5.1 *The measure of a set of n d-boxes in d-dimensional space can be computed in time $O(n^{d/2} \log n)$ using $O(n \log n)$ storage.*

Proof. The amount of time used is essentially the same as in the case we performed the updates one after the other.

To estimate the amount of storage, note that the depth of recursion will be bounded by $O(\log n)$. At each level we use $O(n)$ storage. The bound follows. □

6 Conclusions

We have given a new solution to Klee's measure problem that is much more efficient than previously known results, improving the time bound from $O(n^{d-1})$ to $O(n^{d/2} \log n)$. The technique uses many new ideas, including a result on partitioning space, a generalization of k-d trees and the use of the inclusion/exclusion principle. Streaming was used to reduce the amount of storage used to $O(n \log n)$.

The dynamic data structure we presented for dynamically maintaining the measure can be used for other problems as well. It is very simple to maintain e.g. the perimiter. The method can also be used for computing contours and e.g. i-contours. Moreover, the structure gives a compact representation of the shape of the set of d-boxes. In the full paper some of these applications will be described.

Some open problems do remain. First of all, it should be possible to shave off the factor of $\log n$. But, in fact, there is no guarantee that the method is even near optimal. Improvements or lowerbounds should be worked on.

References

[1] Bentley, J.L., Multidimensional binary search trees used for associated searching, *Comm. ACM* 18 (1975), 509-517.

[2] Bentley, J.L., Algorithms for Klee's rectangle problem, Unpublished notes, Dept. of Computer Science, CMU, 1977.

[3] Edelsbrunner, H., and M.H. Overmars, Batched dynamic solutions to decomposable searching problems, *J. Algorithms* 6 (1985), 515-542.

[4] Fredman, M.L., and B. Weide, The complexity of computing the measure of $\cup[a_i, b_i]$, *Comm. ACM* 21 (1978), 540-544.

[5] Klee, V., Can the measure of $\cup[a_i, b_i]$ be computed in less than $O(n \log n)$ steps?, *Amer. Math. Montly* 84 (1977), 284-285.

[6] Preparata, F.P., and M.I. Shamos, *Computational Geometry,* Springer-Verlag, 1985.

[7] van Leeuwen, J., and D. Wood, The measure problem for rectangular ranges in d-space, *J. Algorithms* 2 (1980), 282-300.

[8] Welzl, E., Partition trees for triangle counting and other range searching problems, *Proc. 4th ACM Symp. on Computational Geometry,* 1988.

[9] Willard, D.E., Polygon retrieval, *SIAM J. Computing* 11 (1982), 149-165.