On Klee's Measure Problem for Grounded Boxes

Hakan Yıldız

Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106, USA
hakan@cs.ucsb.edu

Subhash Suri

Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106, USA
suri@cs.ucsb.edu

ABSTRACT

A well-known problem in computational geometry is Klee's measure problem, which asks for the volume of a union of axis-aligned boxes in d-space. In this paper, we consider Klee's measure problem for the special case where a 2-dimensional orthogonal projection of all the boxes has a common corner. We call such a set of boxes 2-*grounded* and, more generally, a set of boxes is k-*grounded* if in a k-dimensional orthogonal projection they share a common corner.

Our main result is an $O(n^{(d-1)/2}\log^2 n)$ time algorithm for computing Klee's measure for a set of n 2-grounded boxes. This is an improvement of roughly $O(\sqrt{n})$ compared to the fastest solution of the general problem. The algorithm works for k-grounded boxes, for any $k \geq 2$, and in the special case of $k = d$, also called the *hypervolume indicator problem*, the time bound can be improved further by a $\log n$ factor. The key idea of our technique is to reduce the d-dimensional problem to a semi-dynamic *weighted volume* problem in dimension $d-2$. The weighted volume problem requires solving a combinatorial problem of maintaining the *sum of ordered products*, which may be of independent interest.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms

Algorithms, Theory

Keywords

Grounded boxes, hypervolume indicator, Klee's measure, sum of ordered products, weighted volume

SCG'12, June 17–20, 2012, Chapel Hill, North Carolina, USA.

1. INTRODUCTION

Klee's Measure Problem [14] is a classical problem in computational geometry, dating back to 1970s: Given a set of n axis-aligned boxes in d-space, compute the volume of their union. The first non-trivial solution, with running time $O(n^{d-1}\log n)$, was given by Bentley using his space-sweep approach [3], which was quickly improved to $O(n^{d-1})$ time by van Leeuwen and Wood [16]. Several years later, Overmars and Yap [15] made a breakthrough and achieved the bound of $O(n^{d/2}\log n)$, which essentially remains unbeaten more than 20 years later [8]. All the improvements during these intervening years have come in the form of reducing the log factor (the best bound achieved by Chan [8]) or for cubes, unit hypercubes and "fat" boxes [1, 2, 5, 7, 9]. The problem is known to be $\#P$-hard when the dimension is part of the input [6], so the exponential dependence on the dimension seems unavoidable.

In this paper, we present a new result for Klee's measure under the assumption that a 2-dimensional orthogonal projection of all the boxes has a common corner. We call such a collection of boxes 2-*grounded*. (When the boxes share a common corner in a k-dimensional projection, for $1 \leq k \leq d$, we call the set k-*grounded*. Naturally, the algorithm for 2-grounded boxes work trivially for k-grounded, where $k \geq 2$.) Our approach has the following general form. We transform the d-dimensional problem into one of maintaining its $(d-1)$-dimensional cross-section with a sweeping plane. The sweeping is a standard step used in algorithms for the Klee's measure, but instead of solving the cross-section problem directly in $(d-1)$-space, we exploit the grounding property and transform the problem into a $(d-2)$-dimensional *weighted volume* problem. In this problem, each box has a non-negative weight and, in computing the volume, the contribution of each point in space equals the weight of the heaviest box containing it. Solving the weighted volume problem efficiently is the main result of our paper. In particular, we show that the d-dimensional weighted volume can be maintained under insertion at the amortized cost of $O(n^{(d-1)/2}\log^2 n)$, which leads to an $O(n^{(d-1)/2}\log^2 n)$ algorithm for the d-dimensional Klee's Measure Problem on 2-grounded boxes. In solving the weighted volume problem, we also introduce and solve a combinatorial problem of maintaining the *sum of ordered products*, which may be of independent interest.

Besides their intrinsic theoretical interest as a special case of the general Klee's problem, the k-grounded boxes also arise frequently in applications when some of the coordinate axes have a natural "start" or "end" position for ranges. For

instance, if one of the axes represents time recording the duration of some event, then the origin marks the natural start point, and therefore a source of grounding on that axis. Similarly, in sensor databases, physical attributes such as temperature, humidity etc. naturally have a common "lower bound," and the measured quantity is really the deviation from this "grounded" value. In these cases, as long as two or more dimensions are grounded, our algorithm leads to an improved bound for computing the volume of the union.

Finally, if the boxes are d-grounded, namely they all share a common corner, our bound improves by a $\log n$ factor. This particular case of Klee's measure is also known as the *hypervolume indicator*, which is a metric frequently used in multi-objective optimization and evolutionary computing [10, 12, 17]. This particular special case has been studied extensively on its own, as well as a special case of unit cubes. Nevertheless, our new algorithm achieves a new improved bound for the following dimensions: $d = 4, 5, 6$.

The paper is organized in seven sections. In Section 2, we explain how to reduce the d-dimensional problem to a $(d-2)$-dimensional weighted volume problem. In Sections 3 and 4, we solve a special case of the weighted volume involving *halfspaces*, which turns out to be the key problem. In Section 5, we show how the general weighted problem can be solved based on the solution of the halfspace problem. In Section 6, we briefly mention our results for the hypervolume indicator. In Section 7, we conclude with a summary and discussion.

2. DIMENSION REDUCTION: SWEEPING AND WEIGHTING

The classical approach of Overmars and Yap [15] solves the d-dimensional Klee's Measure Problem by reducing it to a dynamic $(d-1)$-dimensional problem. Our key idea is to reduce the d-dimensional problem to a $(d-2)$-dimensional *weighted* volume problem with *insert-only* updates. We reduce one dimension by plane sweep, and another by converting the unweighted problem to a weighted problem. We begin with the high level ideas behind the plane sweep, and the weighting.

Reducing a Dimension by Plane Sweep. Consider a set of axis-aligned boxes $\mathcal{B} = \{B_1, \ldots, B_n\}$ in d-space, for which we want to compute the volume of their union. Without loss of generality, we assume that all the boxes are contained in the positive orthant $\mathbb{R}^d_+$—otherwise, we can divide the problem into 2^d groups, one for each quadrant. We say that a box B is *grounded* with respect to dimension k if B's extent along the kth dimension has the form $(0, B^k)$, where $B^k > 0$ is the length of B along dimension k. Supposing that all boxes in $\mathcal{B}$ are grounded with respect to dimension d, we can compute the volume of their union as follows. We sort the boxes in $\mathcal{B}$ in the descending order of their dth coordinate. Without loss of generality, let the resulting order be $B_1, B_2, \ldots, B_n$, meaning that $B_1^d > B_2^d > \cdots > B_n^d > 0$. We further assume that $B_{n+1}^d = 0$. Then, it is easy to see that the total volume covered by $\mathcal{B}$ is given by the formula

$$\sum_{i=1}^{n} V_{i,d-1} \times (B_i^d - B_{i+1}^d),$$

where $V_{i,d-1}$ is the volume of the set $\{B_1, \ldots, B_i\}$ *projected* onto the plane $\mathbf{x}^d = 0$.[1] This is easily seen by visualizing a plane parallel to $\mathbf{x}^d = 0$, sweeping the space from B_1^d to $B_{n+1}^d = 0$, and by observing that the intersection of this plane with the boxes of $\mathcal{B}$ is constant between two consecutive coordinates in the sorted list, determined entirely by the boxes that precede B_i. If we write $V_{i,d-1}$ for the $(d-1)$-dimensional volume of this intersection, then the d-dimensional volume that lies between B_i^d and B_{i+1}^d is precisely $V_{i,d-1} \times (B_i^d - B_{i+1}^d)$.

The $(d-1)$-dimensional slice changes only when the sweeping plane encounters a new box B_i, and thus we can compute the d-dimensional volume of n boxes by maintaining the $(d-1)$-dimensional volume of their projections subject to n insert-only updates. If the *amortized* update and query cost is $O(F(n))$, then the d-dimensional problem can be solved in time $O(nF(n))$ plus the preprocessing cost including the initial sorting.

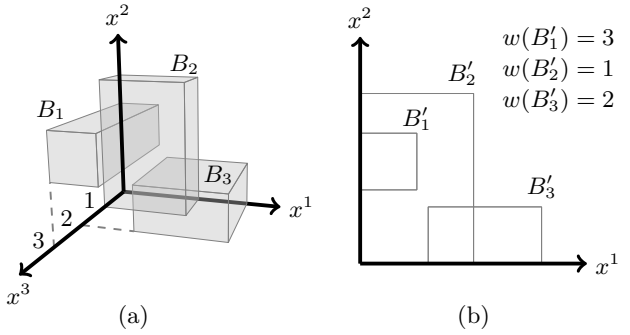


Figure 1: (a) A set of 3 boxes B_1, B_2, B_3 in 3-space, grounded with respect to the x^3 axis, and (b) their 2-dimensional weighted projections.

Reducing a Dimension by Weighting. Another conceptual way to reduce the dimension of the volume problem is the following. Assuming again that the boxes of $\mathcal{B}$ are grounded with respect to the dth dimension, we orthogonally project each box B_i in $\mathcal{B}$ onto the plane $\mathbf{x}^d = 0$, obtaining a $(d-1)$-dimensional box, denoted B'_i, and assign a *weight* $w(B'_i) = B_i^d$ to this projected box. The *weighted volume* of B'_i is defined as $w(B'_i)$ times the (ordinary, $(d-1)$-dimensional) volume of B'_i. See Figure 1 for illustration.

This transformation converts the set $\mathcal{B}$ into a set $\mathcal{B}' = \{B'_1, B'_2, \ldots, B'_n\}$ of $(d-1)$-dimensional weighted boxes. For each point $\mathbf{x} \in \mathbb{R}^{d-1}_+$ on the plane $\mathbf{x}^d = 0$, assign to it the weight of the heaviest box containing $\mathbf{x}$. That is,

$$w(\mathbf{x}) = \max\{w(B'_i) \mid \mathbf{x} \in B'_i\}$$

Then the volume covered by the boxes in $\mathcal{B}$ is given by the integral

$$\int_{\mathbf{x} \in \mathbb{R}^{(d-1)}} w(\mathbf{x}) \, \mathrm{d}\mathbf{x}$$

We call this expression the *weighted volume* of the union of the set $\mathcal{B}'$. Intuitively, this expression weights each point $\mathbf{x}$ on the projection plane by the maximum *height* of a box in $\mathcal{B}$ that contains it, thus correctly computing the volume contribution of each box in $\mathcal{B}$.

[1] $\mathbf{x}^d$ denotes the dth coordinate of point $\mathbf{x}$.

Joint Sweep and Weighting for 2-grounded Boxes. Suppose $\mathcal{B}$ is a set of boxes, grounded along dimensions d and $d-1$, and that their orthogonal projection onto the dimensions $(d-1)$ and d has origin as the common corner. We can now apply both of the above dimension-reduction techniques simultaneously, as follows. We sweep the space along the dth dimension, which requires us to dynamically maintain the $(d-1)$-dimensional volume of the set of boxes intersecting the sweep plane. The updates are *insert only* because boxes are only inserted, and never deleted. Depending on that the boxes are grounded with respect to $(d-1)$th dimension, we maintain the dynamic $(d-1)$-dimensional volume by converting it to a *weighted* $(d-2)$-dimensional volume.

Thus, the d-dimensional Klee's Measure Problem is transformed into a $(d-2)$ dimensional problem of maintaining the weighted volume under insertion of boxes. Solving this problem efficiently is the main contribution of this paper, and the focus of the next three sections. In fact, the crux of the problem proves to be the following special case:

Under insert-only updates, maintain the weighted volume of a set of axis-parallel halfspaces.

In reality, we require the weighted volume of axis-parallel *strips*, but generalizing the halfspace solution to strips is relatively straightforward, incurring an extra $\log n$ factor in complexity. In the interest of simplicity, therefore, we focus on halfspace in the next two sections, and return to the strips only in Section 4.4. For ease of presentation, we first describe the halfspace algorithm in two dimensions (Section 3), and then its generalization to higher dimensions. (Section 4). Section 5 describes how to solve the general weighted volume by combining the strips problem with a space partition technique.

3. WEIGHTED VOLUME OF HALFSPACES: THE PLANAR CASE

We have a set $\mathcal{H}$ of n axis-parallel weighted halfplanes in 2-space, each containing the origin, and a positive orthant axis-aligned rectangle R anchored at the origin. For a point $\mathbf{p}$ in R, define the weight of $\mathbf{p}$ with respect to a subset $\mathcal{H}' \subseteq \mathcal{H}$, denoted $w(\mathbf{p}, \mathcal{H}')$, as the weight of the heaviest halfplane in $\mathcal{H}'$ that contains $\mathbf{p}$; if no such halfplane exists, then the weight is zero. The weighted volume (area) of $\mathcal{H}'$ over R is $\int_{\mathbf{p}\in R} w(\mathbf{p}, \mathcal{H}')\ \mathrm{d}\mathbf{p}$. Our goal is to maintain the weighted volume as $\mathcal{H}'$ undergoes insert-only updates. We will show a data structure with amortized cost of $O(\log n)$ per insertion. The set of axis-aligned halfplanes in $\mathcal{H}'$ is naturally divided into two groups: vertical and horizontal. The vertical halfplanes have the form $0 \le \mathbf{x}^1 \le a$, and the horizontal ones have the form $0 \le \mathbf{x}^2 \le b$. We maintain the weight distribution imposed by these two classes separately, and then show how to compute the joint weight implicitly and efficiently.

3.1 Vertical and Horizontal Gradients

The intersection of R with a halfplane H is a "strip," either vertical or horizontal, containing the origin. Let us focus on the vertical halfplanes of $\mathcal{H}'$, and consider the *partition* they induce on R where each point of R is "claimed" by the maximum weight halfplane containing it. This partition is a sequence of vertical strips in which each strip belongs entirely to one halfplane, each halfplane contributes at most one strip, and the strips are ordered in *descending* weight order from left to right. This follows because all halfplanes contain the origin, and a larger weight halfplane completely overrides the smaller weight halfplane to its left. Visually, the resulting structure looks like a "waterfall", and for ease of reference, we call it the *vertical gradient*.[2] Similarly, the horizontal halfplanes, considered in isolation, induce a *horizontal gradient* of strips ordered in descending order of weights from bottom to top. (Figure 2(a) shows an example, where the vertical gradient consists of three strips of respective weights $w_7 > w_4 > w_1$, and respective widths 4.5, 3.5, and 5.)

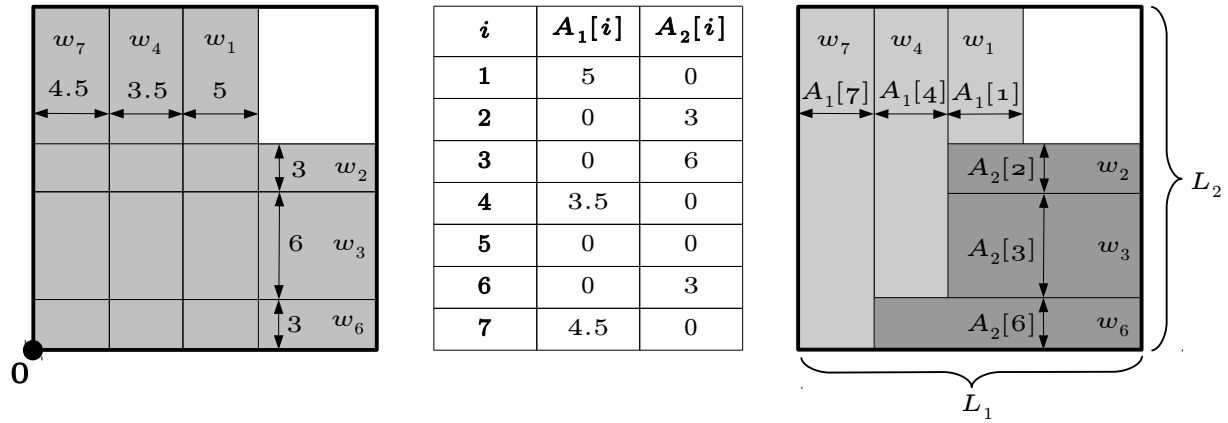
The gradients give a nice and compact representation of the weight structure imposed by the vertical and the horizontal halfplanes *separately*. In order to compute the weighted volume of $\mathcal{H}'$ over R, however, we need to *intersect* the two gradients. An explicit intersection entails $\Omega(n^d)$ complexity in d-dimensions, and so the key is to perform this intersection *implicitly*.

Let w_i be the weight of the ith halfplane $H_i \in \mathcal{H}$, and assume that the halfplanes are ordered in the increasing order of their weights. Without loss of generality, we assume that the weights w_i's are distinct. We maintain two arrays, A_1 and A_2, storing the *widths* of the strips contributed by vertical and horizontal halfplanes, respectively. Both the arrays have size n, whose ith entries correspond to the halfplane H_i. The entry $A_1[i]$ or $A_2[i]$ stores the width of the strip contributed by H_i: if H_i is vertical, it contributes to $A_1[i]$, otherwise to $A_2[i]$. (A halfplane contributes to neither gradient if it is dominated by heavier halfplanes, in which case both the entries are zero.) (See Figure 2(b) for an illustration.)

Let L_1, L_2 be the dimensions of the rectangular region R. Then it is easy to see that $\sum_i A_1[i] \le L_1$ and $\sum_i A_2[i] \le L_2$. (The vertical strips whose widths populate A_1 are disjoint, and their sum cannot exceed the width of R.) Furthermore, *considering each gradient in isolation*, the weighted volume contribution of H_i is precisely $w_i \times (A_1[i] \cdot L_2 + A_2[i] \cdot L_1)$: this follows because only one of $A_1[i]$ or $A_2[i]$ can be nonzero, and so this term is precisely the weighted area of the rectangular strip of H_i.

The next problem is to determine how much of each gradient is claimed by the other. We do that in the next subsection, but first let us consider how to maintain the arrays A_1 and A_2 under *insertion of new halfplanes*. Consider an update to A_1; the horizontal case A_2 is entirely symmetric. When a vertical halfplane H, with weight w, is to be inserted, we first determine its *rank*, namely, the *index* i for the weight w in the ordered sequence $w_1, \ldots, w_n$, which can be done in $O(\log n)$ time. Suppose the rank of H is i, namely, $H \equiv H_i$. Two changes occur in the gradient structure: (1) some of the vertical strips that overlap with H_i are deleted—in particular, those with weights less than w_i, and (2) the strip containing the vertical line defining H_i "shrinks" in width. We can locate the strip to be shrunk in logarithmic time by maintaining a binary search tree on the strips, keyed by their position, and then appropriately update its array value. Starting with that strip, we then traverse the list of vertical strips to the left, deleting each as long as their weights are less than w_i. These strips are deleted also from the search tree, in logarithmic time per

[2]This name is inspired by color gradients, which are images with decreasing color intensity in one direction.



i	$A_1[i]$	$A_2[i]$
1	5	0
2	0	3
3	0	6
4	3.5	0
5	0	0
6	0	3
7	4.5	0

(a) (b) (c)

Figure 2: (a) Illustrating the gradient structures. The vertical gradient has three strips, with weights in descending order $w_7 > w_4 > w_1$, and widths $4.5, 3.5$ and 5, respectively. (b) The array representation of the two gradients shown in (a). The halfplane H_5 is either not inserted yet or covered by halfplanes with higher weights, and so its array entries are zero. (c) Illustrating the "intersection" of the two gradients. The "light gray" (resp., "dark gray") section shows the portion where the weight of the vertical (resp., horizontal) gradient dominates. Recall that the halfplane with larger index has higher weight than the one with smaller index.

strip. (We note that once a strip is deleted from the gradient, it is never reinserted.) We can easily compute the width of the strip produced by H_i and write this value to $A_1[i]$, leading to the following lemma.

LEMMA 1. *The arrays A_1 and A_2 representing the vertical and horizontal gradients can be updated in $O(\log n)$ amortized time after the insertion of a halfplane. The number of array entries whose values change is $O(1)$ amortized.*

3.2 Intersecting Gradient Volumes via Partial Sums

The key problem now is to deduce the correct weighted volume by overlapping the two gradients. Any point that is covered by both the vertical and the horizontal gradients should only be counted once, and receive the weight of the heavier halfplane containing it. In particular, let V_1 denote the weighted volume *contributed* by the vertical gradient: this is the weighted sum over the points where the vertical halfplanes prevail. Similarly, V_2 is the contribution of the horizontal gradient. We claim that V_1 has the following form:

$$V_1 \;=\; \sum_{i=1}^{n} \left(w_i \times A_1[i] \times \left(L_2 - \sum_{j=i+1}^{n} A_2[j] \right) \right)$$

This follows because $\sum_{i=1}^{n} w_i \cdot A_1[i] \cdot L_2$ is the weighted volume *without* considering the horizontal gradient, and the subtracted term is precisely what the horizontal gradient claims away from vertical strips. More precisely, for the halfplane H_i, the portion claimed by the horizontal gradient involves only those strips whose weight is larger than w_i, and if such a strip has "thickness" $A_2[j]$, then the area claimed by the vertical strip away from H_i is $A_1[i] \cdot A_2[j]$. (See Figure 2(c).) By subtracting the total over all such horizontal strips leaves the portion that H_i contributes to the final weighted volume. The complementary term V_2 has the similar form:

$$V_2 \;=\; \sum_{i=1}^{n} (w_i \times A_2[i] \times (L_1 - \sum_{j=i+1}^{n} A_1[j]))$$

We need to maintain these weighted volumes under insert-only updates, which we do by using a dynamic partial sums structure. Recall that the *partial sums* problem is the following:

> Maintain an array $\mathcal{A}$ of size n under an intermixed sequence of *update* and *query* operations, where $update(i, \Delta)$ changes $A[i]$ to $A[i] + \Delta$, and $query(k)$ reports the partial sum $\sum_{i=1}^{k} A[i]$.

Using a balanced binary tree, one can easily support these operations in $O(\log n)$ time each using linear space. The partial sums structure allows us to maintain our weighted volumes V_1 and V_2 as the gradient structures change due to insertion of new halfplanes. In particular, when a halfplane insertion changes $A_1[i]$ by Δ, the value V_1 changes by

$$w_i \times \Delta \times (L_2 - \sum_{j=i+1}^{n} A_1[j])$$

This requires computing the partial sum $\sum_{i=j+1}^{n} A_1[j]$, which is easily done by computing the partial sum $query(j)$, and subtracting it from $query(n)$, in $O(\log n)$ time. On the other hand, if the halfplane H changes the entry $A_2[j]$, then the change in V_1 is $-(\Delta \times \sum_{i=1}^{j-1} w_i \times A_1[i])$. By maintaining a partial sums structure for the array $A[i] = w_i \times A_1[i]$, this update can also be implemented in logarithmic time. In summary, the disjoint weighted volume contributions V_1 and V_2 of the two gradients can be maintained at the cost of $O(\log n)$ time per update, and we have the following key result.

THEOREM 1. *The weighted volume of a set of axis-aligned halfplanes in a rectangle R can be maintained in $O(\log n)$*

amortized time per insertion, with a linear-space data structure.

4. MULTIDIMENSIONAL WEIGHTED VOLUME

In dimension $d > 2$, the basic idea is the same: we organize the halfspaces in d independent gradient structures, which can be updated efficiently when a new halfspace is added. The key difference from the two-dimensional problem arises in how we compute the final weighted volume by intersecting the d gradients structures. Unlike the planar case, where we only needed to compute partial sums, the higher dimensional problems involves a more complex *sum of ordered products*. We discuss the details of these sums in the next two subsections, but first let us express the general form of the d-dimensional gradients.

We have a set $\mathcal{H}$ of n axis-parallel weighted halfspaces, each containing the origin, in d-space. We also have an orthogonal region R in the positive quadrant, anchored at the origin. We wish to maintain the weighted volume of a subset $\mathcal{H}' \subseteq \mathcal{H}$ over R, under insertions of halfspaces from $\mathcal{H}$. We maintain d gradient structures, where the jth structure, denoted G_j, is formed by the halfspaces normal to the jth axis, for $j = 1, 2, \ldots, d$. The gradient G_j consists of strips, in descending weight order, along the positive $\mathbf{x}^j$ direction. We represent G_j as an array A_j of size n, where the entry $A_j[i]$ contains the width of the strip contributed by the halfspace H_i to the gradient G_j, with weight w_i. The array indices are arranged in the increasing weight order of the halfspaces. Each of these d arrays can be maintained at the amortized cost of $O(\log n)$, per insertion of a halfspace, requiring modifications to a constant (amortized) number of array entries, using the technique of the previous section.

4.1 Intersecting the Gradients

Let us now consider how to maintain the weighted volume of the current set of halfspaces $\mathcal{H}'$, given the d gradient arrays. Write the weighted volume as the sum $V_1 + \cdots + V_d$, where V_j is the contribution by G_j to the total volume. We show how to maintain V_1; the others are analogous. In the absence of the other gradients, the weighted volume induced by a strip in G_1 is simply the product of its weight, its width and the $(d-1)$-dimensional volume of its projection on the plane $\mathbf{x}^1 = 0$. Formally, the weighted volume of ith strip is $w_i \times A_1[i] \times \prod_{\alpha=2}^{d} L_\alpha$, where L_α is the length of R along the αth coordinate axis. But some of this volume is *lost* to other gradients because of their higher weight. In particular, for a nonempty set $S = \{\alpha_1, \ldots, \alpha_m\} \subseteq \{2, \ldots, d\}$, let $Loss_i(S)$ be the weighted volume of the ith strip in G_1 that is claimed by all of the gradients in the set $\{G_{\alpha_1}, \ldots, G_{\alpha_m}\}$. In other words, $Loss_i(S)$ is the intersection of the weighted volumes claimed by the gradients $G_{\alpha_1}, \ldots, G_{\alpha_m}$. We can write $Loss_i(S)$ as $w_i \times A_1[i]$ times the $(d-1)$-dimensional volume on the plane $\mathbf{x}^1 = 0$ that is covered by the intersection of heavier subsections of the gradients $G_{\alpha_1}, \ldots, G_{\alpha_m}$. Formally, we write

$$Loss_i(S) = w_i \times A_1[i] \times \prod_{\alpha \in \{2,\ldots,d\} \setminus S} L_\alpha \times \sum_{i < j_{\alpha_1} \le n} A_{\alpha_1}[j_{\alpha_1}] \times \cdots \times \sum_{i < j_{\alpha_m} \le n} A_{\alpha_m}[j_{\alpha_m}]$$

Let $Loss(S)$ be the portion of G_1's weighted volume (over all of its strips) that is claimed by the intersection of the set of gradients $\{G_{\alpha_1}, \ldots, G_{\alpha_m}\}$. Then, we clearly have

$$\begin{aligned} &Loss(S) \\ &= \sum_{1 \le i \le n} Loss_i(S) \\ &= \sum_{1 \le i \le n} \left(w_i \times A_1[i] \times \prod_{\alpha \in \{2,\ldots,d\} \setminus S} L_\alpha \times \sum_{i < j_{\alpha_1} \le n} A_{\alpha_1}[j_{\alpha_1}] \times \cdots \times \sum_{i < j_{\alpha_m} \le n} A_{\alpha_m}[j_{\alpha_m}] \right) \\ &= \prod_{\alpha \in \{2,\ldots,d\} \setminus S} L_\alpha \times \sum_{1 \le i < j_{\alpha_1}, j_{\alpha_2}, \ldots, j_{\alpha_m} \le n} \left(w_i \times A_1[i] \times A_{\alpha_1}[j_{\alpha_1}] \times \cdots \times A_{\alpha_m}[j_{\alpha_m}] \right) \end{aligned}$$

The volume contributed by G_1, namely, V_1 can be written as the quantity *not* claimed by any of the other gradients. Using the inclusion-exclusion principle, we get

$$V_1 = \prod_{2 \le \alpha \le d} L_\alpha \times \sum_{1 \le i \le n} w_i \times A_1[i] \quad - \sum_{S \subseteq \{2,\ldots,d\} \wedge S \neq \emptyset} (-1)^{|S|+1} Loss(S)$$

In this expression, the product involving L_α's is a constant, while the sum $\sum_{1 \le i \le n} w_i \times A_1[i]$ is easily maintained. The non-trivial term is the summation involving $Loss$, and we observe that it contains $(2^{d-1} - 1)$ inner terms of the following general form:

$$C \times \sum_{1 \le i_1 < i_2, i_3, \ldots, i_k \le n} \mathcal{A}_1[i_1] \times \cdots \times \mathcal{A}_k[i_k],$$

where C is constant, the number of indices k is at most d, and each array $\mathcal{A}_j[\,]$ corresponds to either a gradient array or the array $A[i] = w_i \times A_1[i]$. We further decompose each term of the above form into $(k-1)!$ terms by grouping the inner terms of the summation by the ordering of their indices $i_2, \ldots, i_k$. This yields at most $(d-1) \times (d-1)!$ terms of the following form:[3]

$$C \times \sum_{1 \le i_1 < \cdots < i_k \le n} \mathcal{A}_1[i_1] \times \cdots \times \mathcal{A}_k[i_k].$$

Let us call this expression a *sum of ordered products*. All we need now is a data structure that can maintain the sum of ordered products efficiently as array entries are modified. We show in the following subsection (Section 4.2) how to maintain the sum of ordered products in $O(\log n)$ time per update and $O(1)$ time per query, which is sufficient for the following key theorem for the weighted volume of halfspaces in any fixed dimension d.

THEOREM 2. *The weighted volume of a set of axis-aligned halfspaces in a rectangular box R in d dimensions can be*

[3]By distinct weight assumption, all inner terms that contain equal indices are 0, and so we can safely ignore them.

maintained in $O(\log n)$ amortized time per insertion, with a linear-space data structure.

4.2 Maintaining the Sum of Ordered Products

Given d arrays $\mathcal{A}_i$, $i = 1, 2, \ldots, d$, each of size n, we want to efficiently maintain the following sum of ordered products, under updates to individual array entries:

$$\sum_{1 \le i_1 < \ldots < i_d \le n} \mathcal{A}_1[i_1] \times \ldots \times \mathcal{A}_d[i_d]$$

For simplicity, let us assume that n is a power of two. For $0 \le j \le \log n$ and $1 \le k \le n/2^j$, let $S(j,k)$ denote the set of consecutive integers in the range $[(k-1)2^j + 1, k2^j]$. Observe that $S(0,k) = \{k\}$ and $S(\log n, 1) = \{1, \ldots, n\}$. Moreover, $S(j,k)$ is the concatenation of $S(j-1, 2k-1)$ and $S(j-1, 2k)$, for $j \ge 1$. Conceptually, S represents a hierarchically ordered binary partition of the set $\{1, \ldots, n\}$ into singleton integers. If the partition is viewed as a tree, then $S(j,k)$ refers to the kth node from the left at the jth level and it is the parent of $S(j-1, 2k-1)$ and $S(j-1, 2k)$.

Let $T(j,k,l,r)$ denote the following sum of ordered products

$$\sum_{i_l < \ldots < i_r \ \wedge \ i_l, \ldots, i_r \in S(j,k)} \mathcal{A}_l[i_l] \times \ldots \times \mathcal{A}_r[i_r],$$

where $0 \le j \le \log n$, $1 \le k \le n/2^j$ and $1 \le l \le r \le d$. We observe that $T(\log n, 1, 1, d)$ is the sum of ordered products that we want to maintain. Additionally, $T(0,k,l,l) = \mathcal{A}_l[k]$ and $T(0,k,l,r) = 0$ for $l \ne r$. For $j \ge 1$, we can write $T(j,k,l,r)$ in terms of $T()$ values whose first parameter is $(j-1)$, as shown in the following lemma.

LEMMA 2. *For $j \ge 1$, we have the following recurrence:*

$$T(j,k,l,r) = T(j-1, 2k-1, l, r) + T(j-1, 2k, l, r) + \sum_{l \le c < r} \Big(T(j-1, 2k-1, l, c) \times T(j-1, 2k, c+1, r) \Big)$$

PROOF. See Appendix A. □

We maintain $T()$ values in a table: for each valid selection of (j,k,l,r), the table has an entry storing $T(j,k,l,r)$. Then, a query can be answered in $O(1)$ time by reporting $T(\log n, 1, 1, d)$. Moreover, it is easy to show that the table has $O(n)$ entries. When an array entry $A_l[k]$ is updated, we update the table entry for $T(0,k,l,l)$ and the table entries for all $T()$ values that are dependent on $T(0,k,l,l)$ through the recurrence relation given in Lemma 2. Notice that updating an entry takes constant time, assuming d is a constant. It can be easily seen that for a constant value of j, there are at most $d + \binom{d}{2}$ table entries dependent on $T(0,k,l,l)$. Thus, assuming d is a constant, $O(\log n)$ entries are updated in total. This leads to the following theorem.

THEOREM 3. *The sum of ordered products of d arrays of size n can be maintained with an update time of $O(\log n)$, query time of $O(1)$, using a linear-space data structure.*

4.3 Higher Order Partial Sums and Sum of Ordered Products

The reader will notice that the partial sum problem utilized in the halfplane solution is replaced by sum of ordered products in higher dimensions, suggesting a link between the two problems. In fact, the sum of ordered products can be viewed as an *iterated* or *higher-order* generalization of the classical partial sum problem.

Given an array $\mathcal{A}$ of n numbers, the basic *partial sum* problem, addresses the following query operation: report $query(k) = \sum_{i=1}^{k} \mathcal{A}[i]$. There are n different partial sum queries, for $1 \le k \le n$, and one can view them as forming another array $\mathcal{A}'[k] = query(k)$. We can then ask the partial sum problem on $\mathcal{A}'$, which could be considered the *second order* partial sum of $\mathcal{A}$. An iterated application of this process leads to higher order partial sums, with the following general form.

> Given an array $\mathcal{A}$ of size n, its *kth partial sum of order d*, denoted $P_d(k)$, is defined recursively as follows:
>
> $$P_d(k) = \begin{cases} \sum_{i=1}^{k} \mathcal{A}[i] & \text{if } d = 1 \\ \sum_{i=1}^{k} P_{d-1}(i) & \text{if } d > 1 \end{cases}$$

Considering the d arrays $\mathcal{A}_1, \ldots, \mathcal{A}_d$ of size n, the following definition is a further generalization of the iterated partial sums problem for $\mathcal{A}_1$:

> Given d arrays $\mathcal{A}_1, \ldots, \mathcal{A}_d$ of size n, *their kth weighted partial sum*, denoted $W_d(k)$, is defined as follows:
>
> $$W_d(k) = \begin{cases} \sum_{i=1}^{k} \mathcal{A}_1[i] & \text{if } d = 1 \\ \sum_{i=1}^{k} \mathcal{A}_d[i] \times W_{d-1}(i) & \text{if } d > 1 \end{cases}$$

The following lemma shows that the sum of ordered products is actually a weighted partial sum.[4]

LEMMA 3. *For d arrays $\mathcal{A}_1, \ldots, \mathcal{A}_d$,*

$$W_d(n) = \sum_{1 \le i_1 \le \cdots \le i_d \le n} \mathcal{A}_1[i_1] \times \cdots \times \mathcal{A}_d[i_d]$$

PROOF. By induction. The lemma clearly holds for $d = 1$. For $d > 1$,

$$\begin{aligned} W_d(n) &= \sum_{i=1}^{n} \mathcal{A}_d[i] \times W_{d-1}(i) \\ &= \sum_{i=1}^{n} \Big(\mathcal{A}_d[i] \times \sum_{1 \le i_1 \le \cdots \le i_{d-1} \le i} \mathcal{A}_1[i_1] \times \cdots \times \mathcal{A}_{d-1}[i_{d-1}] \Big) \\ &= \sum_{1 \le i_1 \le \cdots \le i_d \le n} \mathcal{A}_1[i_1] \times \cdots \times \mathcal{A}_d[i_d] \end{aligned}$$

□

Our sum of ordered products structure can be easily used to maintain weighted (or iterated) partial sums of d arrays $\mathcal{A}_1, \ldots, \mathcal{A}_d$. Let $\mathcal{A}_{d+1}$ be an additional array such that $\mathcal{A}_{d+1}[i] = 0$ for all i. Then, the weighted partial sum $W_d(i)$ can be obtained by simply incrementing $\mathcal{A}_{d+1}[i]$ by 1 and then querying for the sum of ordered products of the set $\{\mathcal{A}_1, \ldots, \mathcal{A}_{d+1}\}$.

[4]For simplicity of presentation only, we allow the indices in the sum of products to be equal. The original sum of products with strictly ordered indices can be easily reduced to this form by shifting arrays.

4.4 Weighted Volume of Axis-parallel Strips

The weighted volume algorithm given in Section 5 requires us to maintain the weighted volume of a structure known as a *trellis*, which consists of axis-parallel strips in the form $a \leq \mathbf{x}^k \leq b$, rather than halfspaces of the form $\mathbf{x}^k \leq b$. Our technique for maintaining the volume of the halfspaces also works for trellises, by maintaining separate arrays for strips orthogonal to each axis. We utilize a standard data structure known as a *segment tree* to maintain each of these arrays. In particular, the segment tree used to maintain the array for the strips orthogonal to the ith axis stores the projections of these strips on the ith coordinate axis as one-dimensional *weighted* intervals. Note that the segment tree stores each interval in $O(\log n)$ of its nodes, which conceptually partitions the interval into $O(\log n)$ subintervals. (See [4] for details on segment trees.)

The length dominated by each interval (which corresponds to the array entry of its strip) can then be maintained in $O(\log n)$ amortized time per strip insertion. Briefly, this is achieved by deleting the subintervals that are subsumed (both in extent and in weight) by the inserted interval during insertions. The implication of this policy is an invariant that along a root-to-leaf path in the tree, the interval entries are stored in ascending weight order. It then becomes trivial to compute the contribution of each interval to the array via a recursive relation on the nodes. We defer the details of this algorithm to Section 5, where we apply it to a "binary space partition tree" rather than a segment tree.

Each strip insertion affects $O(\log n)$ array entries amortized. This follows from the fact that each inserted interval is stored on $O(\log n)$ nodes in the tree, with $O(\log n)$ total ancestors. The array contribution of the lower weight intervals stored in these $O(\log n)$ ancestors are possibly modified, whereas the contributions of the lower weight intervals in all remaining nodes are *deleted*, for which we charge to their corresponding insertions.

Each array entry change is coupled with an $O(\log n)$ time update in our sums of ordered products structure. It follows that we can maintain the weighted volume of strips in $O(\log^2 n)$ amortized time per insertion. We can now state the following theorem.

THEOREM 4. *The weighted volume of a set of axis-aligned strips in a rectangular box R in d dimensions can be maintained in $O(\log^2 n)$ amortized time per insertion, with a linear-space data structure.*

We finally note that we also need an elimination operation as part of the main algorithm described in Section 5. In this elimination operation, we delete all strips with weights less than a given weight w. This can be easily achieved by doing a binary search on the weights to identify the strips to delete and then deleting the corresponding intervals from the segment trees. As usual, we can charge the deletions to the corresponding insertions, thus this operation can be achieved in $O(\log^2 n)$ amortized time as well.

5. DYNAMIC WEIGHTED VOLUME FOR ARBITRARY BOXES

We now argue that the weighted volume of n arbitrary boxes in d-space, can be maintained under insert-only updates in $O(n^{(d-1)/2} \log^2 n)$ amortized time. For simplicity, we describe a detailed solution only for the two-dimensional case. The extension of this solution to higher dimensions is straightforward as we mention later.

Let $\mathcal{B}$ be a set of n weighted boxes in 2-space. Our objective is to maintain the weighted volume of a dynamic set of boxes that undergoes insertions from $\mathcal{B}$. We propose a structure that achieves this in $O(\sqrt{n} \log^2 n)$ amortized time per update.

The main idea of our solution is to partition the space into rectangular regions such that the set of boxes, $\mathcal{B}$, forms a set of axis-parallel strips inside each region. This enables us to use the specialized structure defined in Section 4.4 to efficiently maintain the overall weighted volume. To form this partition, we follow a classical technique by Overmars and Yap [15].

The partition is formed as follows. We draw a vertical line through each $\sqrt{n}$-th vertical boundary of the boxes along the first coordinate (horizontal) axis. As a result, the space is divided into $\Theta(\sqrt{n})$ slabs, each of which contain $\Theta(\sqrt{n})$ boundaries. We then individually divide each slab further by drawing a horizontal line through: (1) each box corner in the slab and, (2) each $\sqrt{n}$-th horizontal boundary of boxes passing through the slab. Consequently, each slab is partitioned into $\Theta(\sqrt{n})$ final regions. We call these final regions *cells*. The following lemma summarizes the properties of the partition and its proof can be found in [15].

LEMMA 4. *The partition has the following properties.*

- *There are $O(n)$ cells.*
- *Any box intersects $O(\sqrt{n})$ cells on its boundary.*
- *Each cell intersects the boundary of $O(\sqrt{n})$ boxes in $\mathcal{B}$.*
- *The boxes of $\mathcal{B}$ contain no corner in the interior of any cell.*

The last property implies that $\mathcal{B}$ forms a set of strips inside each cell. As the first key part of our algorithm, we maintain, for each cell C, the weighted volume on C contributed *only* by the boxes that intersect C on their boundary. We do this by utilizing an instance of the strips structure defined in Section 4.4 for each cell. By Lemma 4, there are $O(\sqrt{n})$ boxes that intersects any cell on its boundary, thus, the size of each strip structure is $O(\sqrt{n})$. Also, during the insertion of a box B, we can update all strip structures in $O(\sqrt{n} \log^2 n)$ amortized time, because B intersects $O(\sqrt{n})$ cells on its boundary and updating the strip structure of each of these cells takes $O(\log^2 n)$ time. We note that the cells lying on the boundary of B can be efficiently obtained by using the partition tree that we describe below.

Clearly, the weighted volumes maintained in the cells exclude the contributions of the boxes that *entirely* contain the cells. The second key part of our algorithm is to include these contributions so that the overall weighted volume is correctly maintained. To do this efficiently, we utilize a *binary space partition tree.* In particular, we maintain a balanced binary tree in which every node v is associated with a rectangular region of the space R_v, such that for all internal nodes v with children v_l and v_r, R_v is divided into R_{v_l} and R_{v_r} by an axis-aligned hyperplane. Moreover, the root is associated with the whole space and each leaf is associated with one cell of our partition bijectively. It is trivial to show that such a balanced space partition tree can be constructed in $O(n \log n)$ time.

When a box B is inserted to the structure, we store an entry for B at every node v such that B subsumes R_v but not $R_{parent(v)}$. This storage scheme conceptually partitions B into a set of maximal fragments, where each fragment is the intersection $B \cap R_v$ for a node v that B is stored. Note that this fragmentation excludes the sections of B that partially overlap the cells, which are already stored in the corresponding strip structures. It can easily be shown that B is stored in $O(\sqrt{n}\log n)$ nodes in the tree and these nodes can be traversed in $O(\sqrt{n}\log n)$ steps. [15]

For efficient weighted volume maintenance, we remove any box fragments that are entirely contained by boxes of higher weight. In particular, when we insert a box entry B to a node v, we delete all box entries B' stored below v such that $w(B') < w(B)$. Note that the deleted fragments have no contribution to the weighted volume because they are contained by a heavier box, and thus they are safe to be deleted. Similarly, for each cell that a newly inserted box contains, we delete all lower weight boxes in the corresponding strip structure. Finally, we keep only the highest weight box at each node. The result of this policy is an invariant that for any particular box B at a node v, all boxes stored above v have lower weights while all boxes stored below (including the strip structures) have higher weights. Consequently, the weighted volumes of the strip structures directly contribute to the overall. Moreover, we can write the weighted volume contribution of a box B stored at node v as

$$w(B) \times (|R_v| - A(v))$$

where $|R_v|$ is the ordinary volume (area) of region R_v and $A(v)$ is the ordinary volume of the union of the boxes stored below v (including the strip structures). Note that $A(v)$ equals the volume that is claimed from B by higher weight fragments, which are all stored below v. The maintenance of $A(v)$ for leaf nodes can be done trivially due to the simplicity of the strips arrangement [15]. For each internal node v with children v_l and v_r, $A(v)$ can be written recursively in terms of $A(v_l)$ and $A(v_r)$, and thus can easily be updated during the tree traversals. (See [15] for details.)

The overall weighted volume is easily maintained as the sum of weighted volume contributions of each strip structure and each box entry in the tree. It remains to show that removing lower weight box entries during insertions do not increase the overall amortized cost of $O(\sqrt{n}\log^2 n)$ per insertion. Recall that the tree traversal performed to insert a box takes $O(\sqrt{n}\log n)$ steps. To efficiently delete the lower weight boxes after an insertion, we maintain at each node v, the weight of the lowest weight box stored below v. This makes it possible to directly traverse to the box entries to be deleted. The additional traversals performed to delete the entries can be charged to the traversals which inserted these entries. Specifically, the traversal of within a subtree that contains entries of a lower weight box B is charged to the traversal of the same subtree during the insertion of B. Thus, the amortized cost of a box insertion remains $O(\sqrt{n}\log^2 n)$.

One can easily prove that the partition tree and the strip structures can be constructed in $O(n\sqrt{n}\log n)$ time and consume $O(n\sqrt{n})$ space. This yields to the following theorem.

THEOREM 5. *There exists a data structure that maintains the weighted volume of n 2-dimensional boxes in $O(\sqrt{n}\log^2 n)$ amortized time per insertion. This structure can be constructed in $O(n\sqrt{n}\log n)$ time and consumes $O(n\sqrt{n})$ space.*

The partition technique of [15] is generalized to higher dimensions as given in the following lemma.

LEMMA 5. *Given a set $\mathcal{B}$ of n axis-aligned boxes in d-space, one can partition the space into cells such that*

- *There are $O(n^{d/2})$ cells.*
- *Any box intersects $O(n^{(d-1)/2})$ cells on its boundary.*
- *Each cell intersects the boundary of $O(n^{(d-1)/2})$ boxes in $\mathcal{B}$.*
- *No cell contains a $(d-2)$-dimensional facet of any box in its interior.*

By applying the partition tree algorithm together with d-dimensional strip structures, we deduce the following result.

THEOREM 6. *We can maintain the weighted volume of a set of n d-dimensional boxes in $O(n^{(d-1)/2}\log^2 n)$ amortized time per box insertion. This data structure can be constructed in $O(n^{(d+1)/2}\log n)$ time using $O(n^{(d+1)/2})$ space.*

Computing the Klee's measure for 2-grounded boxes in d-space requires n box insertions into a $(d-2)$-dimensional structure, for the total complexity of $O(n \cdot n^{(d-3)/2}\log^2 n) = O(n^{(d-1)/2}\log^2 n)$, giving us the main result of our paper.

THEOREM 7. *Klee's measure for n 2-grounded boxes in d-space can be computed in worst-case time $O(n^{(d-1)/2}\log^2 n)$ and space $O(n^{(d-1)/2})$.*

6. KLEE'S MEASURE FOR d-GROUNDED BOXES

When the boxes are d-grounded, namely, they are all anchored at a common corner, the time complexity of our algorithm improves by a factor of $\log n$. This follows from the fact that the Overmars-Yap partition, when applied on d-grounded boxes, yields to regions that contain half-spaces rather than strips. By Theorem 2, we can maintain weighted volume of each region in $O(\log n)$ time per update (improving on $O(\log^2 n)$ for strips), reducing the running time of our algorithm to $O(n^{(d-1)/2}\log n)$. The d-grounded case of the problem is also known as the hypervolume indicator problem, which is utilized in evolutionary computing often to assess the quality of multi-objective optimization algorithms. Several techniques in the computational geometry literature can be used solve the problem efficiently. The current best bounds with respect to the number of dimensions are as follows:

- For $d \leq 3$, $O(n\log n)$, based on space-sweep [11].
- For $d = 4$, $O(n^{3/2}\text{polylog } n)$ by Chan [7], using reduction to unit-cubes.
- For $d = 5$, $O(n^2\text{polylog } n)$ by Kaplan et al. [13].
- For $d \geq 6$, $O(n^{(d+2)/3})$ by Bringmann [5], using reduction to fat-boxes.

Compared to the above results, our algorithm is faster in dimensions 4, 5 and 6, improving the bound for computing the hypervolume indicator in these dimensions, and using a simpler approach.

7. CONCLUSION

We have proposed a new method for computing Klee's measure on grounded boxes, which includes the hypervolume indicator as a special case. In particular, we obtained a bound of $O(n^{(d-1)/2} \log^2 n)$ for the k-grounded problem for $2 \leq k < d$, which is an improvement of roughly $\sqrt{n}$ over the general Klee's bound. Our technique also leads to a faster algorithm for the hypervolume indicator, which is a special case of the grounded boxes, in dimensions 4, 5 and 6. Given the long and distinguished history of Klee's measure problem, where all the previous improvements have been limited to cube-like boxes, the grounded boxes offer an interesting new direction to pursue.

8. REFERENCES

[1] P. Agarwal. An Improved Algorithm for Computing the Volume of the Union of Cubes. In *Proceedings of the 26th Annual Symposium on Computational Geometry*, pages 230–239, 2010.

[2] P. Agarwal, H. Kaplan, and M. Sharir. Computing the Volume of the Union of Cubes. In *Proceedings of the 23rd Annual Symposium on Computational Geometry*, pages 294–301, 2007.

[3] J. L. Bentley. Solutions to Klee's rectangle problems. *Unpublished manuscript*, 1977.

[4] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2008.

[5] K. Bringmann. Klee's Measure Problem on Fat Boxes in Time $O(n^{(d+2)/3})$. In *Proceedings of the 26th Annual Symposium on Computational Geometry*, pages 222–229, 2010.

[6] K. Bringmann and T. Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. In *Proceedings of 19th International Symposium on Algorithms and Computation*, pages 436–447, 2008.

[7] T. Chan. Semi-online Maintenance of Geometric Optima and Measures. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 474–483, 2002.

[8] T. M. Chan. A (slightly) faster algorithm for Klee's measure problem. *Computational Geometry*, 43(3):243–250, 2010.

[9] L. Chew, D. Dor, A. Efrat, and K. Kedem. Geometric Pattern Matching in d-Dimensional Space. *Discrete and Computational Geometry*, 21(2):257–274, 1999.

[10] M. Fleischer. The measure of pareto optima. applications to multi-objective metaheuristics. In *Evolutionary Multi-Criterion Optimization, Second International Conference*, pages 519–533, 2003.

[11] C. Fonseca, L. Paquete, and M. López-Ibáñez. An Improved Dimension-Sweep Algorithm for the Hypervolume Indicator. In *IEEE Congress on Evolutionary Computation*, pages 1157–1163, 2006.

[12] S. Huband, P. Hingston, L. While, and L. Barone. An evolution strategy with probabilistic mutation for multi-objective optimisation. In *The 2003 Congress on Evolutionary Computation*, volume 4, pages 2284–2291, 2003.

[13] H. Kaplan, N. Rubin, M. Sharir, and E. Verbin. Counting colors in boxes. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 785–794, 2007.

[14] V. Klee. Can the measure of $\cup_1^n [a_i, b_i]$ be computed in less than $O(n \log n)$ steps? *American Mathematical Monthly*, 84(4):284–285, 1977.

[15] M. H. Overmars and C.-K. Yap. New upper bounds in Klee's measure problem. *SIAM Journal on Computing*, 20(6):1034–1045, 1991.

[16] J. van Leeuwen and D. Wood. The measure problem for rectangular ranges in d-space. *Journal of Algorithms*, 2(3):282–300, 1981.

[17] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *Proceedings of 8th International Conference on Parallel Problem Solving from Nature*, pages 832–842, 2004.

APPENDIX

A. PROOF OF LEMMA 2

A member of $S(j,k)$ is either a member of $S(j-1,2k-1)$ or $S(j-1,2k)$. We can, therefore, decompose $T(j,k,l,r)$ into sums of products based on the sets to which the indices belong, as follows:

$$T(j,k,l,r) = \sum_{i_l<\ldots<i_r \ \wedge \ i_l,\ldots,i_r \in S(j-1,2k-1)} A_l[i_l] \times \ldots \times A_r[i_r]$$

$$+ \sum_{i_l<\ldots<i_r \ \wedge \ i_l,\ldots,i_r \in S(j-1,2k)} A_l[i_l] \times \ldots \times A_r[i_r]$$

$$+ \sum_{l \le c < r} \left(\sum_{i_l<\ldots<i_r \ \wedge \ i_l,\ldots,i_c \in S(j-1,2k-1) \ \wedge \ i_{c+1},\ldots,i_r \in S(j-1,2k)} A_l[i_l] \times \ldots \times A_r[i_r] \right)$$

By the distributive property of multiplication over addition, we get

$$T(j,k,l,r) = \sum_{i_l<\ldots<i_r \ \wedge \ i_l,\ldots,i_r \in S(j-1,2k-1)} A_l[i_l] \times \ldots \times A_r[i_r]$$

$$+ \sum_{i_l<\ldots<i_r \ \wedge \ i_l,\ldots,i_r \in S(j-1,2k)} A_l[i_l] \times \ldots \times A_r[i_r]$$

$$+ \sum_{l \le c < r} \left(\begin{array}{c} \sum_{i_l<\ldots<i_c \ \wedge \ i_l,\ldots,i_c \in S(j-1,2k-1)} A_l[il] \times \ldots \times A_c[i_c] \\ \times \\ \sum_{i_{c+1}<\ldots<i_r \ \wedge \ i_{c+1},\ldots,i_r \in S(j-1,2k)} A_{c+1}[i_{c+1}] \times \ldots \times A_r[i_r] \end{array} \right)$$

This expression is equivalent to

$$T(j,k,l,r) = T(j-1,2k-1,l,r) \ + \ T(j-1,2k,l,r) + \sum_{l \le c < r} (T(j-1,2k-1,l,c) \times T(j-1,2k,c+1,r))$$

The lemma follows.