Algorithmica
DOI 10.1007/s00453-013-9797-9

Computing Klee's Measure of Grounded Boxes

Hakan Yıldız · Subhash Suri

Received: 30 April 2012 / Accepted: 23 May 2013

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.

Keywords Grounded boxes · Hypervolume indicator · Klee's measure · Sum of ordered products · Weighted volume

1 Introduction

Klee's Measure Problem [14] is a classical problem in computational geometry, dating back to the 1970s: Given a set of n axis-aligned boxes in d-space, compute the

H. Yıldız (✉) · S. Suri
Department of Computer Science, University of California, Santa Barbara, CA 93106, USA
e-mail: hakan@cs.ucsb.edu

S. Suri
e-mail: suri@cs.ucsb.edu

Published online: 08 June 2013

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 [15]. Several years later, Overmars and Yap [16] 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 insertions at the amortized cost of $O(n^{(d-1)/2} \log^2 n)$, which results in an $O(n^{(d-1)/2} \log^2 n)$ time 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 Sect. 2, we explain how to reduce the d-dimensional problem to a $(d-2)$-dimensional weighted volume problem. In

Sects. 3 and 4, we solve a special case of the weighted volume involving *halfspaces*, which turns out to be the key problem. In Sect. 5, we show how the general weighted problem can be solved based on the solution of the halfspace problem. In Sect. 6, we briefly mention our results for the hypervolume indicator. In Sect. 7, we conclude with a summary and discussion.

2 Dimension Reduction: Sweeping and Weighting

The classical approach of Overmars and Yap [16] 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 (a standard technique used in previous algorithms for Klee's Measure [3, 8, 15, 16]), 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 orthant. 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 set $B_{n+1}^d = 0$. Then, it is easy to see that the total volume covered by $\mathcal{B}$ is given by the formula

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

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 the plane $\mathbf{x}^d = 0$, 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

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

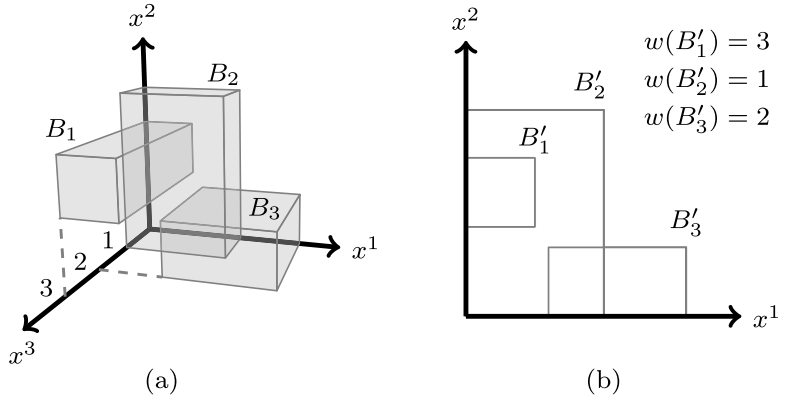


Fig. 1 (**a**) A set of 3 boxes B_1, B_2, B_3 in 3-space, grounded with respect to the x^3 axis, and (**b**) their 2-dimensional weighted projections

problem can be solved in time $O(nF(n))$ plus the preprocessing cost including the initial sorting.

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 Fig. 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 $\mathbf{x}$ the weight of the heaviest box containing $\mathbf{x}$. That is,

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

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

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

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 $\mathcal{B}$.

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 the 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.

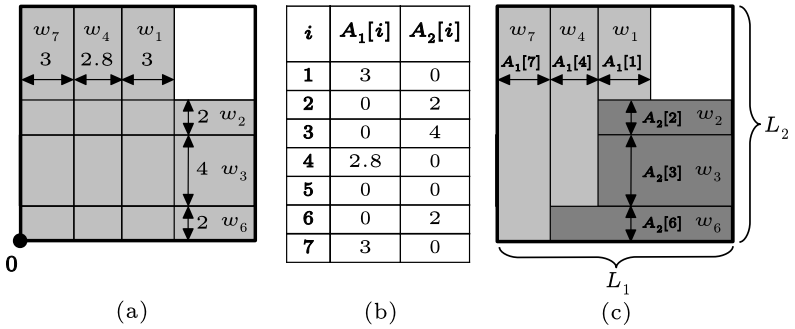
Our algorithm needs to compute the weighted volume of axis-parallel *strips*, not halfspaces. However, the solution is easier to describe for the halfspace case, and generalizes to strips with minor modifications that incur only an extra $\log n$ factor in time complexity. Therefore, in the next two sections, we focus on halfspaces, and then return to the weighted volume of arbitrary boxes in Sect. 5. For ease of presentation, we first describe the halfspace algorithm in two dimensions (Sect. 3), and then its generalization to higher dimensions (Sect. 4). Section 5 describes how to solve the general weighted volume by combining the halfspaces problem with a space partition technique.

3 Weighted Volume of Halfspaces: The Planar Case

Consider a set $\mathcal{H}$ of n axis-parallel weighted halfplanes in 2-space, each containing the origin, and an (arbitrary) axis-aligned rectangle R on the positive orthant 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 $\{\mathbf{x} \in \mathbb{R}^2 \mid 0 \le \mathbf{x}^1 \le a\}$, and the horizontal ones have the form $\{\mathbf{x} \in \mathbb{R}^2 \mid 0 \le \mathbf{x}^2 \le b\}$, where a and b are arbitrary reals. 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



i	$A_1[i]$	$A_2[i]$
1	3	0
2	0	2
3	0	4
4	2.8	0
5	0	0
6	0	2
7	3	0

Fig. 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 3, 2.8 and 3, 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

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 3, 2.8, and 3.)

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 of two size n gradient structures entails $\Omega(n^2)$ complexity [16], 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 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 arrays have size n, and the ith entry of each corresponds 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 entries are zero.) (See Fig. 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 $\sum_i A_1[i] \leq L_1$ and $\sum_i A_2[i] \leq L_2$. (The vertical strips whose widths populate A_1

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

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 non-zero, 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 (through the binary search tree), zeroing out the array entry of each strip as long as their weights are less than w_i. These strips are deleted also from the search tree, in logarithmic time per 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 Fig. 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:

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

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, Δ) 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

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

This requires computing the partial sum $\sum_{j=i+1}^{n} A_2[j]$, which is easily done by computing the partial sum *query*(i), and subtracting it from *query*(n), in $O(\log n)$ time. On the other hand, if the halfplane H changes the entry $A_2[j]$ by Δ, then the change in V_1 is

$$-\left(\Delta \times \sum_{i=1}^{j-1} w_i \times A_1[i] \right)$$

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.*

Proof We use the array data structure described above, with the total weighted volume maintained explicitly, so the query time is $O(1)$. By Lemma 1, inserting a new hyperplane into the array representations requires $O(\log n)$ amortized time, and affects $O(1)$ array entries amortized. For each affected entry, we update two partial-sum structures (one each for V_1 and V_2) and then compute the change in V_1 and V_2, which suffices to recompute the new weighted volume, in $O(\log n)$ time via partial-sums queries. Thus, the total amortized cost of an insertion is $O(\log n)$. Finally, all data structures consume linear space. □

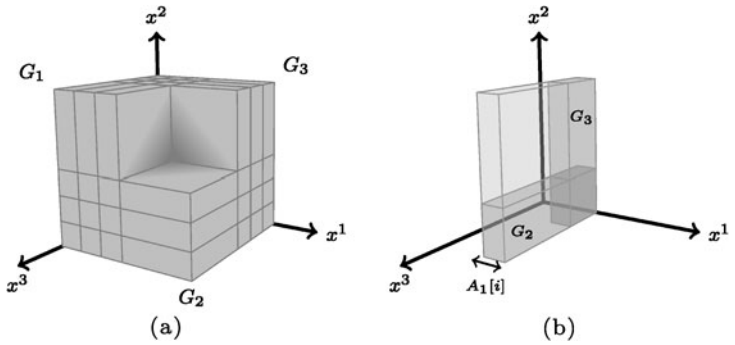


Fig. 3 (**a**) A three-dimensional halfspace arrangement forming three gradients G_1, G_2 and G_3. (**b**) The ith strip on G_1 (*light gray*) and the portion of its volume claimed by gradients G_2 and G_3 (*areas with darker gray*). *The darkest area* is the intersection of the claims of G_2 and G_3, whose weighted volume (in G_1) is given by $Loss_i(\{2, 3\})$

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 orthant, 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. (See Fig. 3(a) for a three-dimensional example.) 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 the common intersection 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 (on the ith strip of G_1) individually claimed by each of 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 $G_{\alpha_1}, \ldots, G_{\alpha_m}$. (See the three-dimensional example in Fig. 3(b).) 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

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

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

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

Observe that this expression contains 2^{d-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 $\lfloor \mathrm{e}(d-1)! \rfloor$ terms[3] (where e is the natural logarithm base) such that each term has the following form:[4]

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

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 (Sect. 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 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:

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

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 $S(j-1, 2k)$.

[3] Note that $\sum_{k=1}^{d} \binom{d-1}{k-1} \times (k-1)! = \sum_{k=0}^{d-1} \frac{(d-1)!}{(d-1-k)!} = \sum_{k=0}^{d-1} \frac{(d-1)!}{k!} \le \mathrm{e}(d-1)!$.

[4] Since we assume that all halfspaces have distinct weights, all inner terms that contain equal indices are 0, and so we can safely ignore these terms and use a strict ordering on indices $i_2, \ldots, i_k$.

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

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

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*:

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

Proof 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:

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

By the distributive property of multiplication over addition, we get

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

This expression is equivalent to

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

The lemma follows. □

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 $A_s[k]$ is updated, we update the table entry for $T(0,k,s,s)$ and the table entries for all $T(\,)$ values that are dependent on $T(0,k,s,s)$ 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 all table entries dependent on $T(0,k,s,s)$ are in the form $T(j, \lceil \frac{k}{2^j} \rceil, l, r)$ such that $0 \le j \le \log n$ and $1 \le l \le s \le r \le d$. 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 (of order d)*, 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.[5]

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 $d > 1$,

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

□

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 $\{\mathcal{A}_1, \ldots, \mathcal{A}_{d+1}\}$.

5 Dynamic Weighted Volume for Arbitrary Boxes

Computing the Klee's measure of n two-grounded boxes in d-space requires maintaining the weighted volume, under insertion-only updates, of n boxes in $(d-2)$-dimensional space. In this section, we complete the discussion of this last step, and show how to maintain the weighted volume under insert-only updates in $O(n^{(d-1)/2} \log^2 n)$ amortized time in d dimensions. Since we require n such insertions in $(d-2)$-dimensional space, the final complexity of our algorithm will be $O(n \cdot n^{(d-3)/2} \log^2 n) = O(n^{(d-1)/2} \log^2 n)$, as desired.

[5] For simplicity of presentation only, we use a non-strict ordering of the array indices (i.e., $i_1 \le \cdots \le i_d$) in this lemma. The sum of products with strictly ordered indices (i.e., $i_1 < \cdots < i_d$, as defined in Sect. 4.2) can be easily reduced to this form by shifting arrays. In particular, the strictly ordered sum of products for d arrays $\mathcal{A}_1, \ldots, \mathcal{A}_d$ is equal to the non-strictly ordered sum of products for arrays $\mathcal{A}'_1, \ldots, \mathcal{A}'_d$ where $\mathcal{A}'_s[i]$ is defined to as $\mathcal{A}_s[i + s - 1]$.

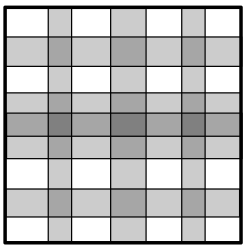


Fig. 4 An arrangement of axis-parallel strips inside a rectangular region in two dimensions

Our scheme is based on the space partition technique originally proposed by Overmars and Yap [16]. In particular, we partition the space into to a set of axis-parallel regions such that the boxes form a set of *axis-parallel strips*[6] inside each region (a structure known as a *trellis*). See Fig. 4 for an illustration. By utilizing a binary space partition tree on the partition, we reduce the main problem to the simpler case of maintaining the weighted volume of axis-parallel strips. This case can be relatively easily solved by extending our halfspace solution.

For simplicity, we first describe a solution for the two-dimensional case in Sect. 5.1, which we later extend into d dimensions in Sect. 5.2. We defer the details of how we generalize our halfspace solution to the case of axis-parallel strips to Sect. 5.3. The following theorem, which we utilize in our descriptions, is proved in Sect. 5.3 and summarizes our data structure for axis-parallel strips.

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. The data structure additionally supports an elimination operation in which all strips with weight less than a given weight are deleted. The cost of the elimination operation can be charged to the preceding insertions, and thus is also $O(\log^2 n)$ amortized.*

5.1 The Planar Case

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.

We first create a partition of the space into rectangular regions such that the set of boxes, $\mathcal{B}$, forms a set of axis-parallel strips inside each region. The partition is formed in two steps as follows. Each box in $\mathcal{B}$ has 2 vertical boundaries (left and right sides) and 2 horizontal boundaries (top and bottom sides). Thus, there are $2n$ vertical and $2n$ horizontal boundaries in total. In the first step of the partition, we sort the vertical boundaries with respect to their first (horizontal) coordinates. Then, we draw a vertical line through each $\sqrt{n}$-th boundary in the sorted list, dividing the space into $O(\sqrt{n})$ slabs, each of which contain $O(\sqrt{n})$ vertical boundaries in its interior. In the second step of the partition, we individually divide each slab along the

[6] In d dimensions, an axis-parallel strip has the form $\{\mathbf{x} \in \mathbb{R}^d \mid a \leq \mathbf{x}^k \leq b\}$ where a and b are reals and k is an integer between 1 and d.

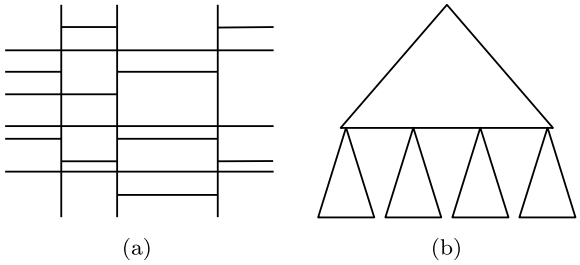


Fig. 5 (**a**) An example partition. (**b**) Its binary space partition tree

second axis. We do this by sorting the horizontal boundaries by their second (vertical) coordinates and then drawing a horizontal line through: (1) each $\sqrt{n}$-th horizontal boundary of boxes passing through the slab, and (2) each box corner inside the slab. Consequently, each slab is partitioned into $O(\sqrt{n})$ final regions containing $O(\sqrt{n})$ horizontal boundaries (see Fig. 5(a)). Note that each slab contains $O(\sqrt{n})$ box corners because there are only $O(\sqrt{n})$ vertical boundaries inside.

We call the final regions of the partition *cells*. The following lemma, which is proved in [16], summarizes the properties of the partition. We include the proof for completeness.

Lemma 4 *The partition has the following properties.*

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

Proof (1) is obvious from construction. We show (2) as follows. Each vertical boundary of a box intersects $O(\sqrt{n})$ cells that are contained in the same single slab. Moreover, each horizontal boundary of a box intersects at most one cell in each slab, thus, $O(\sqrt{n})$ cells in total. It follows that each box intersects $O(\sqrt{n})$ cells on its boundaries. (3) is due to that each cell intersects at most $O(\sqrt{n})$ vertical and horizontal boundaries. Finally, (4) holds because each slab is partitioned into cells by drawing horizontal lines through the box corners. □

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 a two-dimensional instance of the strips structure given in Theorem 4 for each cell. By Lemma 4, there are $O(\sqrt{n})$ boxes that intersect any cell on their 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

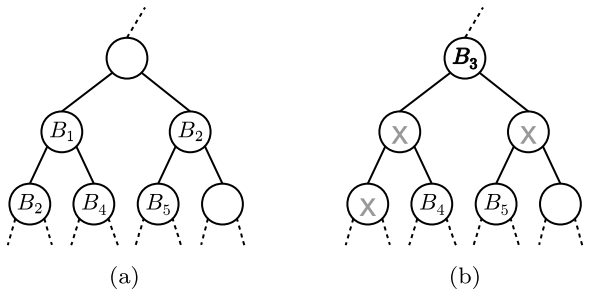


Fig. 6 (**a**) A set of boxes stored in the partition tree. *Box with larger index* has higher weight. (**b**) The tree after inserting an entry for box B_3. All lower weight entries below the inserted entry are deleted

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 a vertical or horizontal line. Moreover, the root is associated with the whole space and each leaf is associated with one cell of our partition bijectively. Such a balanced space partition tree can be easily constructed in $O(n \log n)$ time by constructing a balanced tree for the slabs first, and then for the cells within each slab (see Fig. 5(b)).

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 where 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.

The following lemma is borrowed from [16], and included here with proof only for the sake of completeness.

Lemma 5 *Each box B is stored in $O(\sqrt{n} \log n)$ nodes in the binary space partition tree and these nodes can be traversed in $O(\sqrt{n} \log n)$ steps.*

Proof By Lemma 4 (or 2), there are $O(\sqrt{n})$ leaves v in the tree such that B partially overlaps R_v, i.e., $R_v \cap B \neq \emptyset$ and $R_v \nsubseteq B$. Then, there are $O(\sqrt{n} \log n)$ nodes u (internal or leaf) such that B partially overlaps R_u. This follows from the fact that each such node u is above at least one leaf v that is partially overlapped by B and the tree height is $O(\log n)$. Since the tree is binary and each node that stores B has a parent u that is partially overlapped, B is stored at $O(\sqrt{n} \log n)$ nodes.

The nodes storing B can be traversed by recursively descending in the tree through the nodes that are partially overlapped until nodes subsumed by B are reached. Since the number of partially overlapped nodes is $O(\sqrt{n} \log n)$, so is the traversal time. □

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 $w(B') < w(B)$. See Fig. 6 for an example. 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 subsumes, we delete all lower weight boxes in the corresponding strip structure by using its elimination operation. 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

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

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 strictly 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 v can be done with no extra cost by utilizing a copy of the strips structure at v that maintains the *ordinary* volume rather than the weighted volume. (In this case, the weights of the boxes are set to 1.) 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. In particular, we have

$$A(v) = \begin{cases} |R_{v_l}| + |R_{v_r}| & \text{if } v_l \text{ and } v_r \text{ both store a box} \\ |R_{v_l}| + A(v_r) & \text{if } v_l \text{ stores a box and } v_r \text{ does not} \\ A(v_r) + |R_{v_r}| & \text{if } v_r \text{ stores a box and } v_l \text{ does not} \\ A(v_r) + A(v_l) & \text{if neither } v_l \text{ nor } v_r \text{ store a box} \end{cases}$$

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 (including the boxes stored in the strip structures). This allows us to locate the box entries that we need to delete, without searching the whole tree. In particular, we descend down the tree as long as the lowest weight stored below the current node is lower than weight of the inserted box. The additional traversals performed to locate the deleted entries in the tree can be charged to the traversals which inserted these entries. Notice that each step of an insertion traversal (i.e., traversal of an edge) is charged at most once by a deletion traversal. Finally, as we mention in Theorem 4, the time spent to eliminate the box entries in the strip structures can also be charged to the preceding insertions. It follows that the amortized cost of a box insertion remains $O(\sqrt{n}\log^2 n)$.

The binary space partition tree uses $O(n)$ space and can be constructed in $O(n\log n)$ time. Each of the $O(n)$ strip structures uses $O(\sqrt{n})$ space (by Lemma 4 (or 3)) and can be constructed in $O(\sqrt{n}\log n)$ time. This yields 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.*

5.2 The d-Dimensional Case

We now extend our weighted volume data structure to d dimensions. In d dimensions, the high-level algorithm is the same. We construct a d-dimensional partition whose cells do not contain any $(d-2)$-dimensional facets of any box, i.e., all boxes are strips inside the cells. In this section, we briefly describe how this construction is done. (Details on this construction may be found in [16].) Note that the rest of the algorithm can be adapted trivially to d-dimensions: we organize the final partition in a balanced binary space partition tree and apply our maintenance scheme. We emphasize that we utilize the d-dimensional version of our strips structure in the leaves of the partition tree.

Let B be a set of n weighted boxes in d-space. We construct the partition in d steps as follows. Each box has two $(d-1)$-dimensional boundaries that are orthogonal to the ith coordinate axis, for a particular i $(1 \le i \le d)$. For simplicity, we call these boundaries *i-bounds*. In the first step of construction, we sort the 1-bounds of the boxes with respect to their first coordinate. Then, we draw a $(d-1)$-dimensional hyperplane (orthogonal to the first axis) through each $\sqrt{n}$-th 1-bound in the sorted list, dividing the space into $O(\sqrt{n})$ sections, each of which contains $O(\sqrt{n})$ 1-bounds. In the second step, we individually divide each section along the second coordinate. We do this by sorting the 2-bounds and then drawing a hyperplane (orthogonal to the second axis) through: (1) each $\sqrt{n}$-th 2-bound in the section, and (2) each 2-bound whose owner box contains a 1-bound in the section. Consequently, each section is partitioned into $O(\sqrt{n})$ subsections containing $O(\sqrt{n})$ 1-bounds and 2-bounds. In the third step, we cut each section with respect to the third axis through: (1) each $\sqrt{n}$-th 3-bound in the section and (2) each 3-bound whose owner box contains a 1-bound or 2-bound in the section. We continue in this fashion for a total of d steps, creating a partition with $O(n^{d/2})$ cells each containing $O(\sqrt{n})$ i-bounds for any $1 \le i \le d$.

The following is the d-dimensional analog of Lemma 4, also from [16].

Lemma 6 *The partition has the following properties.*

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

Proof (1) is true because at each step of the construction, the number of sections increase by a factor of $O(\sqrt{n})$. We prove (2) as follows. Each i-bound intersects at most $O(n^{(i-1)/2})$ sections just before the ith step of the construction. After the ith step, for any section that an i-bound intersected previously, the i-bound can intersect at most one of its resulting subsections (because the cuts are parallel to the i-bound). Thus, each i-bound still intersects $O(n^{(i-1)/2})$ sections after the ith step. It follows that each i-bound intersects $O(n^{(d-1)/2})$ sections when all d steps are completed.

Since each box has $2d$ i-bounds in total, (2) follows. (3) is due to that each cell intersects at most $O(\sqrt{n})$ i-bounds for each $1 \leq i \leq d$. Finally, we prove (4) as follows. A $(d-2)$-dimensional facet is the intersection of a j-bound and a k-bound of a box for $1 \leq j < k \leq d$. By induction, after ith step of the construction, no resulting subsection contains a j-bound and a k-bound of the same box such that $1 \leq j < k \leq i$. Thus, after the dth step, no cell contains a $(d-2)$-dimensional facet. □

Given the above partition, we can utilize our partition tree algorithm of the previous section (with d-dimensional strip structures). By following similar proof steps, 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.

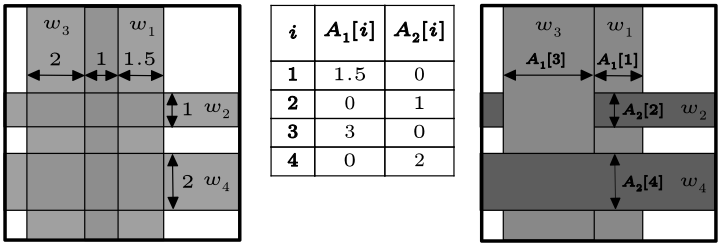
Theorem 7 *Klee's measure for n 2-grounded boxes in d-space can be computed in worst-case time $O(n^{(d-1)/2}\log^2 n)$ and space $O(n^{(d-1)/2})$.*

5.3 Weighted Volume of Axis-Parallel Strips

Our technique for maintaining the volume of halfspaces extends easily to strips, by keeping a separate array for strips orthogonal to each axis. In this section, we describe this extension, and show how to maintain the weighted volume of axis-parallel strips that are in the form $\{\mathbf{x} \in \mathbb{R}^d \mid a \leq \mathbf{x}^k \leq b\}$ (where a and b are reals). In particular, let $S_1, \ldots, S_n$ be a set of n strips such that the indices are ordered by weight, i.e., $w(S_1) < \cdots < w(S_n)$. We maintain d arrays $A_1, \ldots, A_d$ of size n. If the strip S_i is orthogonal to the kth axis, then the entry $A_k[i]$ represents the total width of S_i not claimed by the strips orthogonal to the kth axis. Otherwise, $A_k[i]$ is zero. See Fig. 7. The algorithm from Sect. 4 can then be applied on these arrays to maintain the weighted volume, at the cost of $O(\log n)$ per array entry change.

We utilize a standard data structure known as a *segment tree* to maintain the arrays. In particular, the segment tree used to maintain the array A_k stores the strips orthogonal to the kth axis as one-dimensional *weighted* intervals on the kth axis. (The intervals are the projections of these strips.) The segment tree stores each interval in $O(\log n)$ of its nodes, which conceptually partitions the interval into $O(\log n)$ subintervals in a way very similar to the binary space partition tree described in Sect. 5.1. (For more details on segment trees, see [4].)

By applying the same maintenance ideas from the space partition tree, we can maintain the length dominated by each interval (which corresponds to the array entry of its strip) in $O(\log n)$ time per interval insertion. Each interval 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



i	$A_1[i]$	$A_2[i]$
1	1.5	0
2	0	1
3	3	0
4	0	2

Fig. 7 (**a**) A two-dimensional arrangement of four strips with weights in descending order $w_4 > w_3 > w_2 > w_1$. Notice the intersection between the strips with weights w_1 and w_3, where the strip with weight w_3 dominates. (**b**) The array representation of the strips. (**c**) Illustrating the intersection of the strips. The "*light gray*" (resp.,"*dark gray*") section shows the portion where the weights of the vertical (resp., horizontal) strips dominate

modified, whereas the contributions of the lower weight intervals in all descendants 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 finally note that we also need an elimination operation as part of the main algorithm described in Sect. 5.1. 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.

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.

References

1. Agarwal, P.: An improved algorithm for computing the volume of the union of cubes. In: Proceedings of the 26th Annual Symposium on Computational Geometry, pp. 230–239 (2010)
2. Agarwal, P., Kaplan, H., Sharir, M.: Computing the volume of the union of cubes. In: Proceedings of the 23rd Annual Symposium on Computational Geometry, pp. 294–301 (2007)
3. Bentley, J.L.: Solutions to Klee's rectangle problems. Unpublished manuscript (1977)
4. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer, Berlin (2008)
5. Bringmann, K.: Klee's measure problem on fat boxes in time $O(n^{(d+2)/3})$. In: Proceedings of the 26th Annual Symposium on Computational Geometry, pp. 222–229 (2010)
6. Bringmann, K., Friedrich, T.: Approximating the volume of unions and intersections of high-dimensional geometric objects. In: Proceedings of 19th International Symposium on Algorithms and Computation, pp. 436–447 (2008)
7. Chan, T.: Semi-online maintenance of geometric optima and measures. In: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 474–483 (2002)
8. Chan, T.M.: A (slightly) faster algorithm for Klee's measure problem. Comput. Geom. **43**(3), 243–250 (2010)
9. Chew, L., Dor, D., Efrat, A., Kedem, K.: Geometric pattern matching in d-dimensional space. Discrete Comput. Geom. **21**(2), 257–274 (1999)
10. Fleischer, M.: The measure of Pareto optima. Applications to multi-objective metaheuristics. In: Evolutionary Multi-criterion Optimization, Second International Conference, pp. 519–533 (2003)
11. Fonseca, C., Paquete, L., López-Ibáñez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In: IEEE Congress on Evolutionary Computation, pp. 1157–1163 (2006)
12. Huband, S., Hingston, P., While, L., Barone, L.: An evolution strategy with probabilistic mutation for multi-objective optimisation. In: The 2003 Congress on Evolutionary Computation, vol. 4, pp. 2284–2291 (2003)
13. Kaplan, H., Rubin, N., Sharir, M., Verbin, E.: Counting colors in boxes. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 785–794 (2007)
14. Klee, V.: Can the measure of $\bigcup_1^n [a_i, b_i]$ be computed in less than O($n \log n$) steps? Am. Math. Mon. **84**(4), 284–285 (1977)
15. van Leeuwen, J., Wood, D.: The measure problem for rectangular ranges in d-space. J. Algorithms **2**(3), 282–300 (1981)

16. Overmars, M.H., Yap, C.K.: New upper bounds in Klee's measure problem. SIAM J. Comput. **20**(6), 1034–1045 (1991)
17. Zitzler, E., Künzli, S.: Indicator-based selection in multiobjective search. In: Proceedings of 8th International Conference on Parallel Problem Solving from Nature, pp. 832–842 (2004)