Klee's Measure Problem Made Easy

Timothy M. Chan[*]

August 10, 2013

Abstract

We present a new algorithm for a classic problem in computational geometry, *Klee's measure problem*: given a set of n axis-parallel boxes in d-dimensional space, compute the volume of the union of the boxes. The algorithm runs in $O(n^{d/2})$ time for any constant $d \geq 3$. Although it improves the previous best algorithm by "just" an iterated logarithmic factor, the real surprise lies in the simplicity of the new algorithm.

We also show that it is theoretically possible to beat the $O(n^{d/2})$ time bound by logarithmic factors for integer input in the word RAM model, and for other variants of the problem.

With additional work, we obtain an $O(n^{d/3} \operatorname{polylog} n)$-time algorithm for the important special case of orthants or unit hypercubes (which include the so-called "hypervolume indicator problem"), and an $O(n^{(d+1)/3} \operatorname{polylog} n)$-time algorithm for the case of arbitrary hypercubes or fat boxes, improving a previous $O(n^{(d+2)/3})$-time algorithm by Bringmann.

1 Introduction

Klee's measure problem is, without exaggeration, easy to state:

> Given a set B of n axis-parallel boxes (hyperrectangles) in $\mathbb{R}^d$, compute the volume of the union of B.

The dimension d is assumed to be a constant in this paper. When we are additionally given a domain (a box) Γ, the objective is to compute the volume within Γ. Although the combinatorial complexity of the union may be $\Theta(n^d)$ in the worst case, the hope is that we may not need to construct the union itself in order to compute its measure.

First posed by Klee [27] in 1977, the problem harkens back to the early days of computational geometry. It is a simple exercise to design an $O(n \log n)$-time algorithm for $d = 2$ [4, 32]. In higher dimensions, after initial solutions by Bentley and others [4, 34], Overmars and Yap [30] announced an $O(n^{d/2} \log n)$-time algorithm at FOCS 25 years ago. For a long time, their result had remained the record holder, until a few years ago a small improvement in the logarithmic factor was found by this author [11]: the improved algorithm takes $O(n^{d/2} 2^{O(\log^* n)})$ time for any $d \geq 3$. The first result of the present paper is a new algorithm that runs in $O(n^{d/2})$ time, thus removing the remaining iterated logarithmic factor completely.

[*]Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (tm-chan@uwaterloo.ca).

Motivation. To explain the significance of the result, one should first note that besides its intrinsic value, Klee's problem is related to many other problems about orthogonal objects, not necessarily about volumes. For example, all known algorithms for Klee's measure problem (including our new one) can be adapted to solve the *depth* problem: find a point $p \in \mathbb{R}^d$ that maximizes the number of boxes in B containing p. A special case of both the measure and the depth problem is the *coverage* problem: decide whether the union of the boxes in B covers the domain Γ. Given a set of n points in $\mathbb{R}^d$ and a number k, finding a cluster of k points with minimum L_∞-diameter can be reduced to the depth problem [9, 19]. Given two sets of n points in $\mathbb{R}^3$, finding a translation that minimizes the Hausdorff distance between the two sets can be reduced to the depth problem for $O(n^2)$ boxes [13]. The continuous p-center problem on weighted graphs can be reduced to solving a number of coverage problems for boxes in $\mathbb{R}^p$ [5]. The list goes on.

There are also other problems [17, 20, 25] that may not be directly reducible to Klee's measure problem but nonetheless can be solved by similar techniques. Also, indirectly, Klee's measure problem is linked to certain fundamental combinatorial questions about the decomposition of orthogonal objects, notably, orthogonal *binary space partitions* (BSPs) [18, 31].

The exponent $d/2$ is somewhat unusual in computational geometry, and thus makes the problem more interesting from the theoretical perspective. There is good reason to believe $d/2$ is tight. Specifically, the problem of deciding the existence of a *d-clique* in a graph with $\sqrt{n}$ vertices can be reduced to the coverage problem for $O(n)$ boxes in $\mathbb{R}^d$ [11]. Consequently, the coverage and measure problems are $W[1]$-hard with respect to the dimension d. Furthermore, since the current best algorithm for d-cliques on arbitrary n-vertex graphs that avoids fast matrix multiplication requires near-$O(n^d)$ time, with current knowledge one cannot hope for a purely combinatorial algorithm for Klee's measure problem that beats $O(n^{d/2})$ time, ignoring logarithmic factors.

Simplicity. The main virtue of the new algorithm is not that it improves (very slightly) the previous result, but is in its simplicity. The author's previous $O(n^{d/2}2^{O(\log^* n)})$-time algorithm is more complicated than Overmars and Yap's, but the new algorithm is even simpler than Overmars and Yap's! Traces of the ideas can be found in previous work, but they are distilled in the simplest form. For example, unlike in existing algorithms, no dynamic data structures are used. The divide-and-conquer strategy it employs is just a variant of the well-known *k-d tree*.

By the time one has read two pages past the introduction, one would have already seen the highlight of the entire paper. (Section 2 requires no prior background, even for a non-geometer.) In fact, the new algorithm is perfect material for teaching divide-and-conquer.

Polylogarithmic-factor speedups. In certain settings, it turns out that the $O(n^{d/2})$ bound can be surpassed by logarithmic factors, as we show in Section 3. For example, for the depth problem (and in particular the coverage problem), our new approach yields a time bound of $O((n^{d/2}/\log^{d/2} n)(\log\log n)^{O(1)})$ for $d \geq 3$, which improves the author's previous $O((n^{d/2}/\log^{d/2-1} n)(\log\log n)^{O(1)})$ bound [11].

We also obtain an $O((n^{d/2}/\log^{d/2-c} n)(\log\log n)^{O(1)})$ time bound for $d > 2c$ for the *weighted depth problem*: given a set B of weighted boxes in $\mathbb{R}^d$, find a point $p \in \mathbb{R}^d$ that maximizes the sum of the weights of the boxes in B containing p. Here, c is an absolute constant, upper-bounded by 5. This is the first $o(n^{d/2})$ result which holds for arbitrary real-valued weights. Interestingly, our algorithm uses Meiser's *point location* method in hyperplane arrangements in high dimensions [28], or alternatively Meyer auf der Heide's method about *linear decision trees* [29]. It is the first instance the author is aware of where Meiser's or Meyer auf der Heide's results were applied to obtain faster polynomial-time algorithms in a traditional real RAM model (although more generally linear decision trees have been used for polylogarithmic-factor speedups before in other context, e.g., in Fredman's algorithm for all-pairs shortest paths or min-plus matrix multiplication [22]).

For the original measure problem, we show that speedups are possible in the standard word RAM model under the reasonable setting of integer input coordinates. We obtain an $O((n^{d/2}/\log^{d/2-2} n)(\log\log n)^{O(1)})$ time bound for $d \geq 5$ under the assumption that $n \geq w$, where w is the word size. For a polynomially bounded universe, we can save one more logarithmic factor. The idea this time involves the use of the Chinese remainder theorem.

Special cases. In the most technically challenging part of the paper (Section 4), we combine our approach with additional ideas and obtain the current best results for the measure problem in a number of special cases. These cases have been actively studied by researchers in recent years.

Among the most important is the case when the boxes are orthants containing $(-\infty, \ldots, -\infty)$, i.e., orthants of the form $\{(x_1, \ldots, x_d) \mid (x_1 \leq \textvisiblespace) \wedge \cdots \wedge (x_d \leq \textvisiblespace)\}$, where each occurrence of $\textvisiblespace$ stands for a possibly different real number. The measure problem for such orthants is sometimes known by another name, the *hypervolume indicator* problem. For $d \leq 3$, the union of orthants has linear complexity and can be constructed in $O(n \log n)$ time. In higher dimensions, the union has combinatorial complexity $\Theta(n^{\lfloor d/2 \rfloor})$ in the worst case, but again the hope is to compute the measure without constructing the union. After some minor improvements in certain dimensions [10], the first true breakthrough was obtained in 2010 by Bringmann [7], who presented an impressive $O(n^{(d+2)/3})$-time algorithm. Later, Yildiz and Suri [36] gave an $O(n^{(d-1)/2} \log n)$-time algorithm, which is better only for very small d. In this paper, we present an $\widetilde{O}(n^{d/3})$-time algorithm,[1] which improves all previous algorithms for all $d \geq 4$. Our algorithm actually works for *arbitrary orthants*, of the form $\{(x_1, \ldots, x_d) \mid (x_1 \;?\; \textvisiblespace) \wedge \cdots \wedge (x_d \;?\; \textvisiblespace)\}$, where each occurrence of $\textvisiblespace$ stands for a possibly different real number and each occurrence of "?" stands for either $\leq$ or $\geq$.

Bringmann's algorithm works more generally for the case of *arbitrary hypercubes*; apparently, specialization to orthants does not make his algorithm any faster. For arbitrary hypercubes, we obtain an $\widetilde{O}(n^{(d+1)/3})$-time algorithm for any $d \geq 3$, thus strictly subsuming Bringmann's result. The improvement is more dramatic in specific small dimensions. For example, we obtain the first subquadratic algorithm for 4D hypercubes.

Note that the case of *unit hypercubes* reduces to the case of arbitrary orthants. To see this, build a grid with unit side length; each unit hypercube intersects only $O(1)$ grid cells, and inside each grid cell, a unit hypercube is equivalent to an orthant. Furthermore, the case of *similar-size fat* boxes—where all side lengths are $\Theta(1)$—easily reduces to unit hypercubes, and the case of *arbitrary fat* boxes—where the ratio of the maximum to the minimum side length of each box is $\Theta(1)$—reduces to arbitrary hypercubes. Our algorithms thus apply to these cases as well.

Both our $\widetilde{O}(n^{d/3})$ and $\widetilde{O}(n^{(d+1)/3})$ algorithms incorporate ideas similar to Bringmann's, but we explain these ideas in a more general "functional" framework which we believe provides better understanding at the conceptual level.

Remarks. For 3D cubes specifically, Agarwal, Kaplan, and Sharir [2] obtained an $\widetilde{O}(n^{4/3})$ time bound, which was later improved to $\widetilde{O}(n)$ by Agarwal [1] with a complicated algorithm. It is possible

[1]Throughout this paper, the $\widetilde{O}$ notation hides polylogarithmic factors.

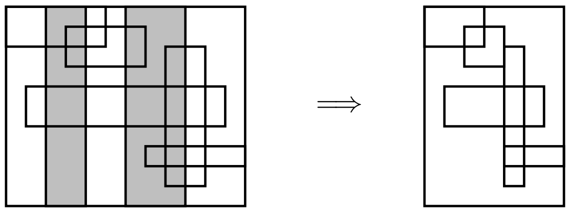

Figure 1: Simplify by readjusting x_i values. (The shaded slabs are boxes in B_i.)

to use our approach to get a relatively simple $O(n^{1+\varepsilon})$-time solution, but since the result is not an improvement, we omit the details. Also, Yildiz and Suri's algorithm works more generally, with an extra logarithmic factor, for so-called *2-grounded* boxes of the form $\{(x_1, \ldots, x_d) \mid (x_1 \leq \text{␣}) \wedge (x_2 \leq \text{␣}) \wedge (\text{␣} \leq x_3 \leq \text{␣}) \wedge \cdots \wedge (\text{␣} \leq x_d \leq \text{␣})\}$. To avoid further diversions, we will not investigate this particular case here.

As observed by Bringmann [8], the measure problem for general boxes in $\mathbb{R}^d$ can be reduced to the measure problem for orthants in $\mathbb{R}^{2d}$; thus, the problem for orthants or hypercubes remains $W[1]$-hard with respect to d.

2 A Simple $O(n^{d/2})$-Time Algorithm

For convenience, we change the objective, in the remainder of the paper, to computing the measure of the *complement* of the union of B within the given box domain Γ.

Our new algorithm actually follows one of the most obvious divide-and-conquer strategies, as outlined below:

```
measure($B, \Gamma$):
 0.  if $|B|$ is below a constant then return the answer directly
 1.  *simplify* $B$
 2.  *cut* $\Gamma$ into two subcells $\Gamma_L$ and $\Gamma_R$
 3.  return measure($\{$boxes of $B$ intersecting $\Gamma_L\}, \Gamma_L$) +
            measure($\{$boxes of $B$ intersecting $\Gamma_R\}, \Gamma_R$)
```

The twist is that we simplify the input before we cut. We now explain the meaning of "simplify" and the way we cut.

Step 1: How to simplify. For each $i \in \{1, \ldots, d\}$, let B_i be the set of all boxes of B that are equivalent to slabs of the form $\{(x_1, \ldots, x_d) \mid \text{␣} \leq x_i \leq \text{␣}\}$ when restricted to Γ. Let B_* be all the B_i's combined, and let $\widehat{B} = B - B_*$. We describe a way to eliminate the boxes in B_*. First compute the union of B_i; this reduces to computing the union of 1-dimensional intervals and can be done by a linear scan after sorting the x_i-coordinate values. The union of B_i is a collection of disjoint slabs orthogonal to x_i. Now, readjust all x_i values so that each slab's x_i-length is reduced to 0 (see Figure 1); the readjustment can be done by a linear scan over the x_i values.

Clearly, the measure of the complement of the union of B is preserved after this transformation. After the simplification, the main property to remember is that each remaining box in $\widehat{B}$ must have at least one $(d-2)$-face[2] intersecting Γ.

Step 2: How to cut. For each $(d-2)$-face f that is orthogonal to the i-th and j-th axis (i.e., lies in a $(d-2)$-flat[3] of the form $\{(x_1,\ldots,x_d) \mid (x_i = \text{\textvisiblespace}) \wedge (x_j = \text{\textvisiblespace})\}$), define the weight of f to be $2^{(i+j)/d}$ (which is a value between 1 and 4). Cut Γ into two open subcells using the hyperplane $x_1 = m$, where m is the weighted median of the first coordinates of the $(d-2)$-faces orthogonal to the first axis and intersecting Γ. Then renumber the coordinate system so that the old axes $1, 2, \ldots, d$ correspond to the new axes $d, 1, \ldots, d-1$ respectively.

Consider a $(d-2)$-face f orthogonal to the i-th and j-th axis, with $i, j \neq 1$. After the axis renumbering, its weight changes from $2^{(i+j)/d}$ to $2^{(i-1+j-1)/d}$, i.e., the weight decreases by a factor of $2^{2/d}$.

Next consider a $(d-2)$-face f orthogonal to the first and j-th axis, with $j \neq 1$. After the axis renumbering, its weight changes from $2^{(1+j)/d}$ to $2^{(d+j-1)/d}$, i.e., it increases by a factor of $2^{(d-2)/d}$. However, after the cut, the total weight of the $(d-2)$-faces orthogonal to the first axis and intersecting either (open) subcell decreases by a factor of 2, for a net decrease by a factor of $2^{2/d}$.

Thus, the total weight of all $(d-2)$-faces intersecting either subcell drops by a factor of $2^{2/d}$.

Analysis. Let $T(n, N_{d-2})$ be the running time needed for an input which has been pre-sorted in each dimension, where n is an upper bound on the input size and N_{d-2} is an upper bound on the total weight of all $(d-2)$-faces in B intersecting Γ. The simplification step yields

$$T(n, N_{d-2}) \;\leq\; T(O(N_{d-2}), N_{d-2}) + O(n) \tag{1}$$

and the cutting step yields

$$T(n, N_{d-2}) \;\leq\; 2\,T(n, N_{d-2}/2^{2/d}) + O(n). \tag{2}$$

Putting the two together and setting $T(N) = T(cN, N)$ for a suitable constant c, we obtain the recurrence

$$T(N) \;\leq\; 2\,T(N/2^{2/d}) + O(N), \tag{3}$$

which immediately solves to $T(N) = O(N^{d/2})$ for $d \geq 3$.

Remarks.

- This gives a new $O(n \log n)$-time algorithm for $d = 2$ as well. (In the $d = 2$ case, the $(d-2)$-faces are points, weights do not matter, and we can always cut vertically.) In contrast, the textbook algorithm for the 2D Klee's measure problem [4, 32] is based on a plane sweep and requires search trees with insertion and deletion operations.
- The hidden constant in the $O(n^{d/2})$ is actually polynomial in d (it appears to be $O(d)$).
- The algorithm is simple to the point that some readers should be tempted to implement it. In fact, similar divide-and-conquer strategies must have been tried by practitioners before, one would imagine.

[2] A j-face is a j-dimensional face of an input box; e.g., a $(d-1)$-face is a facet, and a 1-face is an edge.
[3] A j-flat is a j-dimensional flat (affine subspace); e.g., a $(d-1)$-flat is a hyperplane, and a 1-flat is a line.

- The space used by our recursive algorithm satisfies the recurrence $S(N) \leq S(N/2^{2/d}) + O(N)$ and is easily seen to be linear. In contrast, Overmars and Yap's paper [30] required extra tricks to keep space linear, whereas the author's previous paper [11] did not analyze space at all.

- Some form of the simplification idea for Klee's measure problem has appeared before, notably, in the author's previous paper [11] (and also [7]). Although in [11] we replace B_* with $O(n)$ extra boxes instead of readjusting coordinate values, the result of the simplification is similar.

- In the cutting step, the axis renumbering is just to ease the analysis; equivalently, we are cycling through the dimensions we choose to cut along. This is just like *k-d trees* [16, 32]. Another similar cutting approach from the literature is an old *binary space partition* (BSP) construction by Paterson and Yao [31]. Instead of sums of weights, they used an expression involving products raised to some power, resulting in a less trivial analysis.

- Our weighted-median cutting approach implies a BSP construction for n axis-parallel $(d-2)$-flats of size $O(n^{d/2})$, which easily generalizes to a BSP construction for n axis-parallel j-flats of size $O(n^{d/(d-j)})$. The new proof is simpler than previous ones [18, 31].

- Our cutting approach also implies a simple proof for the following result: the union of n orthants in $\mathbb{R}^d$ has at most $O(n^{d/2})$ complexity and can be decomposed into $O(n^{d/2})$ cells of $O(1)$ complexity. This is because in the case of orthants, each B_i can be simplified to at most two axis-aligned halfspaces. The bound is tight in even dimensions (but unfortunately not in odd dimensions). There were previous proofs that the union of orthants has $O(n^{\lfloor d/2 \rfloor})$ complexity [6]. Agarwal *et al.* [26] has shown that the union of orthants can be decomposed into $O(n^{\lfloor d/2 \rfloor})$ cells but only in the case when the orthants all contain $(-\infty, \ldots, -\infty)$.

3 Polylogarithmic-Factor Speedups

In this section, we apply the algorithm in Section 2 to different variants of Klee's measure problem, and show that under many scenarios, the algorithm can be sped up by logarithmic factors in theory. In Section 3.1, we work in a standard RAM model with word size $w \geq \log n$. In Section 3.2, we assume that the input coordinates and weights are real numbers and work in the standard real-RAM model; each word may hold an input real number or a $(\log n)$-bit pointer/index; standard arithmetic operations on real numbers take constant time. In Section 3.3, we assume that the input coordinates are integers from a bounded universe $[U] := \{1, \ldots, U\}$, and work in a standard RAM model with word size $w \geq \log U$ (i.e., each input number fits in a word); standard arithmetic operations, shifts, and bitwise-logical operations on words take constant time.

3.1 Depth

For the depth problem, we first consider a slight generalization: given a set of boxes and d univariate step functions $h_1, \ldots, h_d$, we want a point $p = (x_1, \ldots, x_d)$ that maximizes the number of boxes containing p plus $h_1(x_1) + \cdots + h_d(x_d)$.

How to simplify. We describe a different way to eliminate B_*, as defined in Section 2. First project B_i to get intervals on the i-th axis. Let $h'_i(x_i)$ be the number of such intervals containing x_i. This step function h'_i can be computed by a linear scan after sorting. Add h'_i to h_i. To avoid blowing up the complexity of the new step function h_i, let X_i denote the set of x_i-coordinate values that appear among the vertices of $\widehat{B}$. In any range between two consecutive values in X_i, we can replace the function with the maximum in the range. As a result, the complexity of each step function is reduced to $O(|\widehat{B}|)$.

Furthermore, we may assume that the difference between the maximum M_i and the minimum m_i of the function h_i is at most $|\widehat{B}|$, because points $p = (x_1, \ldots, x_d)$ with $h_i(x_i) < M_i - |\widehat{B}|$ cannot be maximal. By shifting, we may assume that h_i has values between 0 and $|\widehat{B}|$.

Thus, (1) still holds, where the "input size" n upper-bounds the number of boxes, the complexity of the d step functions, and the maximum values of the step functions.

Analysis, with better base cases. By applying (1) and terminating when the subproblem size drops below some fixed parameter b, we obtain

$$T(n) \ \le (n/b)^{d/2}(T(b) + O(b)). \tag{4}$$

Subproblems of input size b can be encoded in $O(b \log b)$ bits, since for the depth problem, coordinate values can be replaced by their ranks, and the input size bounds the maximum value of the step functions. We can precompute a table storing the answers to all subproblems of size b in $2^{O(b \log b)}$ time. Then $T(b) = O(1)$. The total time is $O((n/b)^{d/2} b + 2^{O(b \log b)})$. Setting $b = \varepsilon \log n / \log\log n$ for a sufficiently small constant $\varepsilon > 0$ yields a time bound of $O((n^{d/2}/\log^{d/2-1} n)(\log\log n)^{O(1)})$.

We can do better still by using word packing tricks. Observe that an input of size n can be packed into $O((n \log n)/w)$ words. By modifying all the linear scans to take time linear in the number of words (by keeping various lists of boxes and $(d-2)$-faces sorted along each dimension), we can obtain

$$T(N) \ \le \ 2\,T(N/2^{2/d}) + O((N \log N)/w).$$

As a result,

$$T(n) \ \le \ (n/b)^{d/2}(T(b) + O(b \log b)/w).$$

With $T(b) = O(1)$, the total time is $O((n/b)^{d/2}(b \log b)/w)$. For $b = \varepsilon \log n / \log\log n$ and $w \ge \log n$, this is $O((n^{d/2}/(w \log^{d/2-1} n))(\log\log n)^{O(1)}) = O((n^{d/2}/\log^{d/2} n)(\log\log n)^{O(1)})$.

Remarks. The previous paper [11] has already used table lookups or word operations to obtain polylogarithmic-factor improvements for the depth problem, but the new result is better by about one logarithmic factor. The improvement stems from the ability to avoid dynamic data structures in our simpler algorithm, unlike the algorithm in [11].

As mentioned in the introduction, there is a reduction from the *d-clique* problem for $\sqrt{n}$-vertex graphs to the coverage problem [11]. It is worth comparing our result with known combinatorial algorithms for the clique problem that achieve polylogarithmic-factor speedups [35].

3.2 Weighted Depth

For the weighted depth problem, a similar simplification step works, although we no longer care about the maximum values of the step functions.

By (4), we get $O(n/b)^{d/2}$ subproblems of size b. Subproblems can no longer be encoded in a small number of bits for real-weighted input, so we proceed differently. Define the *signature* of a subproblem to be its input after replacing coordinate values with ranks; the signature has $O(b \log b)$

bits. Collect all subproblems with a common signature together in a common list; the number of lists is at most $b^{O(b)}$.

Subproblems with a common signature reduce to evaluating a common $O(b)$-variate function f over the weights, where f is the upper envelope (pointwise maximum) of at most $O(b^d)$ linear functions. This is because the signature determines the combinatorial structure of the arrangement of boxes, and the weighted depth in each of the $O(b^d)$ cells in the arrangement is a sum of weights. Since we are evaluating the same function over a long list of $O(b)$-tuples, it makes sense to preprocess the function to speed up querying.

Lemma 3.1 *We can preprocess N linear functions $f_1, \ldots, f_N$ in b variables in $(bN)^{O(b)}$ time so that $f(x) := \max\{f_1, \ldots, f_N\}(x)$ can be evaluated in $O(b^c \log N)$ time for any given $x \in \mathbb{R}^b$, where c is an absolute constant.*

Proof: This problem reduces to *point location* in an arrangement of $O(N^2)$ hyperplanes $\{x \in \mathbb{R}^b \mid f_i(x) = f_j(x)\}$ over all i, j, since in each cell of the arrangement, the sign of $f_i(x) - f_j(x)$ is determined for all i, j. Building on previous work by Clarkson [14], Meiser [28] gave a point location algorithm in any high nonconstant dimension b with $O(b^5 \log N)$ query time, thus achieving $c = 5$. He stated an expected preprocessing time bound of $O(N^{O(b)})$, although this ignored hidden factors of the form $b^{O(b)}$. (The preprocessing algorithm can be derandomized using deterministic ε-*net* constructions [12].)

Alternatively, in the framework of *linear decision trees*, Meyer auf der Heide [29] gave a point location algorithm with $O(b^4 \log b)$ query time, under the assumption that the coefficients of the linear functions are integers bounded by $O(1)$ (which holds in our application). He did not state a preprocessing time bound, however. $\Box$

In our application, we need $b^{O(b)}$ invocations of Lemma 3.1, with $N = O(b^d)$. So, we obtain $T(b) = O(b^c \log b)$, after a preprocessing time of $b^{O(b)}$. The total time is $O((n/b)^{d/2} b^c \log b + b^{O(b)})$. Setting $b = \varepsilon \log n / \log\log n$ yields a time bound of $O((n^{d/2}/\log^{d/2-c} n)(\log\log n)^{O(1)})$.

3.3 Measure in the Word RAM

By (4), we get $O(n/b)^{d/2}$ subproblems of size b. For the measure problem, subproblems with a common signature reduce to evaluating a common $O(b)$-variate polynomial function with degree d and at most $O(b^d)$ terms. This is because we can form a grid with $O(b^d)$ cells from the coordinates along each dimension, and the signature determines which grid cells are part of the union of B; the measure of each grid cell is a product of d side lengths. Since we are evaluating the same function over a long list of $O(b)$-tuples, it seems conceivable that the amortized cost per query can be lowered, at least when the final objective is to report the sum. More precisely, the underlying problem is formulated in the lemma below. For integer input in the word RAM model, we present three solutions: the basic idea is to use the Chinese remainder theorem to reduce the universe size before doing table lookups; in the second and third solutions, we further reduce the number of table lookups by word packing and sorting.

Lemma 3.2 *Given a b-variate polynomial f with $O(1)$ degree and $O(1)$-bounded integer coefficients, and given m b-tuples $x^{(1)}, \ldots, x^{(m)} \in [U]^b$, where the elements of the tuples are all from a set X of n numbers, we can compute $S = \sum_{\ell=1}^{m} f(x^{(\ell)})$ in*

1. *$O(m\log U/\log\log U + mb\log b + n\log U/\log\log U + 2^{O(b\log\log U)})$ time, or*
2. *$O(mb^2\log\log U + n\log U/\log\log U + 2^{O(b\log\log U)})$ time, or*
3. *$O(mb^2\log\log\log U + mb\log\log U + n\log U/\log\log\log U + 2^{O(b\log\log\log U)}\log^2 U)$ time.*

Proof: Let $p_1,\ldots,p_k$ be k primes from a smaller universe $[u]$ so that the product exceeds U^c for a sufficiently large constant c. By the prime number theorem, there exist such primes with $u=\Theta(\log U)$ and $k=\Theta(\log U/\log u)$.

First precompute a table of $f(x)$ values for all $x\in[u]^b$ in $O(b^{O(1)}u^b)$ time. For each $x\in X$, precompute $(x \bmod p_1,\ldots,x \bmod p_k)$, which we refer to as the *Chinese-remainder (CR) code* of x. Each CR code requires $O(k\log u)=O(\log U)$ bits, i.e., $O(1)$ words. The time required is $O(nk)$.

Solution 1. Consider a fixed $\ell\in\{1,\ldots,m\}$. We take the CR codes of the b elements of $x^{(\ell)}$, and rearrange the bits to get the b-tuples $x^{(\ell)} \bmod p_j$ for each $j\in\{1,\ldots,k\}$. The rearrangement amounts to doing a "transpose" operation on $O(b)$ words, which takes $O(b\log b)$ standard word operations (e.g., see [33]). We can then compute $f(x^{(\ell)}) \bmod p_j = f(x^{(\ell)} \bmod p_j) \bmod p_j$ by table lookup for each j, and reconstruct $f(x^{(\ell)})$ by the Chinese remainder theorem; the time required over all ℓ is $O(mk)$.

Solution 2. Let $t=\log U/(b\log u)$. Consider a group of t indices ℓ. Each b-tuple $x^{(\ell)} \bmod p_j$ occupies $O(b\log u)$ bits. Rearrange the bits to get all t b-tuples $x^{(\ell)} \bmod p_j$ in $O(1)$ words, for each j. The rearrangement reduces to a transpose operation on $O(tb)$ words, which takes $O((tb)\log(tb))$ time.

Now consider a fixed $j\in\{1,\ldots,k\}$. Sort the list of all m b-tuples $x^{(\ell)} \bmod p_j$, which are packed in $O(m/t)$ words. In general, sorting b'-bit numbers packed in $O(m')$ words can be done in $O(m'b')$ time (by a modified mergesort [3], since merging takes time linear in the number of words and b' levels of merging suffice). Thus, our list can be sorted in $O((m/t)b\log u)$ time. For each of the $O(m/t)$ words in the sorted list, if all b-tuples in the word are identical to, say, x, we can do one table lookup for $f(x)$, multiply by the number of b-tuples in the word, and add the result to a running total S_j. On the other hand, if the word contains more than one distinct b-tuple, we do a table lookup for $f(x)$ for each such b-tuple x, and add to S_j. Since the number of words of the latter kind is at most $2^{O(b\log u)}$, the additional time is at most $O(m/t+2^{O(b\log u)}\log U)$.

We can then obtain $S \bmod p_j = S_j \bmod p_j$, and reconstruct S by the Chinese remainder theorem. The total time is $O(k(m/t)b\log u + k2^{O(b\log u)}\log U + mb\log(tb) + nk + b^{O(1)}u^b) = O(mb^2\log u + 2^{O(b\log u)}\log^2 U + mb\log\log U + n\log U/\log u + b^{O(1)}u^b)$.

Solution 3. We can further speed up the second solution by using a 2-level CR code. Namely, let $q_1,\ldots,q_{k'}$ be primes from $[u']$ such that the product exceeds u^c, with $u'=O(\log u)$ and $k'=O(\log u/\log u')$. The new CR code of x contains $(x \bmod p_i) \bmod q_j$ for all $i\in\{1,\ldots,k\}$ and $j\in\{1,\ldots,k'\}$. We can reconstruct S by two levels of applications of the Chinese remainder theorem. The total time is similarly $O(mb^2\log u' + 2^{O(b\log u')}\log^2 U + mb\log\log U + n\log U/\log u' + b^{O(1)}u'^b)$, now with $u'=O(\log\log U)$. (Of course, extending CR codes to more levels would yield further but smaller improvements.) □

In our application, we need $b^{O(b)}$ invocations of Lemma 3.2, where the m's sum to $O(n/b)^{d/2}$. Solution 1 takes time $O((n/b)^{d/2}\log U/\log\log U + (n/b)^{d/2}b\log b +$

$b^{O(b)} n \log U/\log\log U + b^{O(b)} 2^{O(b \log\log U)})$. Setting $b = \varepsilon \log n/\log\log U$ yields a time bound of $O((n^{d/2}/\log^{d/2} n) \log U (\log\log U)^{O(1)})$.

Alternatively, Solution 3 takes time $O((n/b)^{d/2} b^2 \log\log\log U + (n/b)^{d/2} b \log\log U + b^{O(b)}[n \log U/\log\log\log U + 2^{O(b \log\log\log U)} \log^2 U])$. Assume $n \geq w$, and thus $U \leq 2^n$. Setting $b = \varepsilon \min\{\log n/\log\log n, \log n/\log\log\log U\}$ yields $O((n^{d/2}/\log^{d/2-2} n)(\log\log n)^{O(1)})$ time for $d \geq 5$.

Remarks. Whether similar speedups are possible in the real RAM model (as opposed to the integer RAM model) would depend on the algebraic complexity of $S = \sum_{i=1}^{m} f(x^{(\ell)})$. In our application, f is a multilinear function, specifically a d-linear function over $O(b)$ variables. It is not difficult to see that computing S in this case can be reduced to the multiplication of an $O(b^{\lceil d/2 \rceil}) \times m$ and $m \times O(b^{\lfloor d/2 \rfloor})$ matrix. However, this step would require at least $\Omega(mb^{\lceil d/2 \rceil})$ time (known results on rectangular matrix multiplication [15, 23] can in fact achieve near $mb^{\lceil d/2 \rceil}$ time when $b = o(m^{\varepsilon})$); unfortunately this bound is too big to yield $o(n^{d/2})$ time at the end.

More concretely, for $d = 3$, the underlying problem is to compute a sum of the form $\sum_{\ell=1}^{m} \sum_{i=1}^{b} \sum_{j=1}^{b} \sum_{k=1}^{b} w_{ijk} x_{i\ell} y_{j\ell} z_{k\ell}$. At first glance, this expression appears different from the bilinear or trilinear forms usually encountered in the matrix multiplication literature.

In some very vague sense, the difference between the weighted depth and the measure problem is akin to the difference between min-plus matrix multiplication and standard matrix multiplication. (In fact, reductions from min-plus matrix multiplication to dynamic weighted depth in 2D [24], and from dynamic standard matrix-vector multiplication [21] to dynamic measure in 2D [11], have been mentioned in the literature. And there is also the aforementioned connection with the clique problem, which is related to matrix multiplication.) As we have been successful in getting speedups for real-weighted depth in Section 3.2, it would be intriguing if algebraic techniques for fast matrix multiplication could get polylogarithmic-factor speedups (or even more boldly, reduce the exponent $d/2$) for Klee's measure problem for real-valued input.

4 Faster Algorithms for Special Cases

We now explore how to adapt the approach in Section 2 to get faster algorithms for the measure problem in the cases of arbitary orthants and arbitrary hypercubes. As noted in the introduction, this would imply improved algorithms for similar-size fat boxes and arbitrary fat boxes respectively.

It should be noted that the algorithms in this section are not applicable to the depth problem—this is to be expected, because the depth problem for general boxes can be reduced to the depth problem for orthants (an exercise left for the reader).

In Section 2, we have shown how to eliminate all boxes B_i that are equivalent to slabs of the form $\{(x_1, \ldots, x_d) \mid \text{␣} \leq x_i \leq \text{␣}\}$. In the next subsection, we first describe a more sophisticated simplification procedure for boxes of another type. This procedure is the key behind our two subsequent algorithms.

4.1 Simplification of 2-Sided Orthants

For each $i, j \in \{1, \ldots, d\}$ with $i \neq j$, let B_{ij} be the set of all boxes in B that are equivalent to 2-sided orthants of the form $\{(x_1, \ldots, x_d) \mid (x_i \ ? \ \text{␣}) \wedge (x_j \ ? \ \text{␣})\}$ when restricted to Γ, where each occurrence

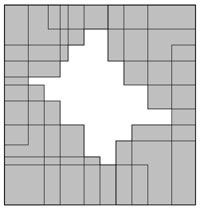

Figure 2: The union of B_{ij}.

of "?" may be $\leq$ or $\geq$. Redefine B_* as all the B_i's and B_{ij}'s combined, and let $\widehat{B} = B - B_*$. We describe a way to reduce the number of boxes in B_* while preserving the measure of the union of B.

We take a "functional" approach. Let $[E]$ denote the indicator function for a predicate E. The measure problem is equivalent to computing the integral of $[(x_1, \ldots, x_d) \notin \bigcup B]$ over Γ. We consider a slight generalization of the problem:

Problem 4.1 Given a set B of boxes and univariate step functions $h_1, \ldots, h_d$ ("density" functions) of total complexity $O(n)$, compute the integral of $[(x_1, \ldots, x_d) \notin \bigcup B] \cdot h_1(x_1) \cdots h_d(x_d)$ over Γ.

This generalized problem can be reduced back to the original problem by readjusting the coordinate values: namely, between two consecutive x_i values, say, α^-, α^+, that appear among the vertices of B, we can reset the length of the x_i-interval $[\alpha^-, \alpha^+]$ to $\int_{\alpha^-}^{\alpha^+} h_i(\xi)\, d\xi$ (if h_i may take on negative values, we can separate into negative and positive components first). The elimination of B_i as described in Section 2 can be viewed alternatively as zeroing out some $h_i(x_i)$ values.

For each i, j, compute the union of B_{ij} (see Figure 2); this reduces to four instances of the 2D *maxima* problem [32] and can be done in linear time after sorting. We can express the complement of the union as $\{(x_1, \ldots, x_d) \mid (x_j \geq f^-(x_i)) \wedge (x_j \geq g^-(x_i)) \wedge (x_j \leq f^+(x_i)) \wedge (x_j \leq g^+(x_i))\}$ where f^-, f^+ are univariate monotone-decreasing step functions and g^-, g^+ are univariate monotone-increasing step functions. This motivates the following definitions:

Definition 4.2 A *basic predicate* $E(x_1, \ldots, x_d)$ is a conjunction of $O(d^2)$ conditions, each of which is the form $x_j \ ?\ f(x_i)$ where f is a monotone step function.

A *basic function* is a function of the form $F(x_1, \ldots, x_d) = [E(x_1, \ldots, x_d)] \cdot h_1(x_1) \cdots h_d(x_d)$ where E is a basic predicate and $h_1, \ldots, h_d$ are piecewise-polynomial functions (*density* functions). The *complexity* of F refers to the total complexity (number of pieces) of all the monotone step functions in E and of the density functions $h_1, \ldots, h_d$. The degree of F refers to the maximum degree of $h_1, \ldots, h_d$.

In the $\widehat{B} = \emptyset$ case, Problem 4.1 reduces to integrating a basic function that corresponds to B_*. More generally:

Observation 4.3 *Problem 4.1 is equivalent to computing the integral of a basic function $F(x_1, \ldots, x_d)$ over the complement of the union of $\widehat{B}$, where F has complexity $O(n)$ and degree 0.*

This reformulation allows us to avoid the complications of visualizing in higher dimensions, but instead approach the problem simply by mechanical manipulations of formulas. Our definition of basic functions is designed in such a way that they satisfy a closure property under integration—this will be the key lemma.

Lemma 4.4 *If F is a basic function of complexity n and degree s, then $\widehat{F}(x_1,\ldots,x_d) = \int_{-\infty}^{x_d} F(x_1,\ldots,x_{d-1},\xi)\,d\xi$ is a sum of $O(1)$ basic functions of complexity $O(n)$ and degree $s+1$, constructible in linear time.*

Proof: Certain elementary facts about univariate functions will be useful; for example, the inverse of a monotone step function and the composition of two monotone step functions are monotone step functions; the pointwise minimum of two monotone increasing step functions is a monotone increasing step function; the product of a piecewise polynomial function and a step function is a piecewise polynomial function; the integral $\widehat{h}(x) = \int_{-\infty}^{x} h(\xi)\,d\xi$ of a degree-s piecewise polynomial function h is a degree-$(s+1)$ piecewise polynomial function. These new functions have complexity linear in the complexities of the given functions, and can be generated in linear time.

To prove the lemma, we describe a series of straightforward algebraic manipulations to rewrite the function $F(x_1,\ldots,x_{d-1},\xi)\cdot[\xi \le x_d]$, to make integration over the variable ξ easier later:

- First rewrite each condition of the form $x_i \;?\; f(\xi)$ as $\xi \;?\; f^{-1}(x_i)$. (The direction of the second "?" may be reversed depending on whether f is monotone increasing or decreasing.)

- Next apply the following rule repeatedly: rewrite $[(\xi \le f(x_i)) \wedge (\xi \le g(x_j))]$ as $[(f(x_i) \le g(x_j)) \wedge (\xi \le f(x_i))] + [(f(x_i) > g(x_j)) \wedge (\xi \le g(x_j))]$. (This holds even when $j = d$ and g is the identity function.) Application of this rule increases the number of terms in the sum, but each term has one fewer occurrence of the variable ξ. A similar rule holds for rewriting $[(\xi \ge f(x_i)) \wedge (\xi \ge g(x_j))]$. At the end, each term will have only one condition of the form $\xi \ge f(x_i)$ and one of the form $\xi \le g(x_j)$.

- The preceding rule creates new conditions of the form $f(x_i) \;?\; g(x_j)$, not technically allowed in the definition of a basic predicate. To fix this, rewrite $f(x_i) \;?\; g(x_j)$ as $x_j \;?\; g^{-1}(f(x_i))$ if $i \ne j$. If $i = j$, then $[f(x_i) \;?\; g(x_i)]$ is just a step function in x_i, which can be multiplied with the density function for x_i.

- Finally, to reduce the number of conditions to $O(d^2)$ in each term, rewrite $(x_i \le f(x_j)) \wedge (x_i \le g(x_j))$ as $x_i \le \min\{f,g\}(x_j)$ for any two monotone increasing step functions. A similar rule holds for two monotone decreasing step functions, and for $\ge$ instead of $\le$.

The number of rewriting steps is $O(1)$. At the end, $F(x_1,\ldots,x_{d-1},\xi)\cdot[\xi \le x_d]$ becomes a sum of $O(1)$ terms, each of which is of the form $G(x_1,\ldots,x_d)\cdot[f(x_i) \le \xi \le g(x_j)]\cdot h(\xi)$ for some basic function G and some i,j. (It could happen that $j = d$ and g is the identity function.) The integral of such a term over ξ is $G(x_1,\ldots,x_d)\cdot[f(x_i) \le g(x_j)]\cdot(\widehat{h}(g(x_j)) - \widehat{h}(f(x_i)))$, where $\widehat{h}(x) = \int_{-\infty}^{x} h(\xi)\,d\xi$. This expression is a sum of basic functions (after recalling the rewriting rule for $f(x_i) \;?\; g(x_j)$). $\square$

Applying Lemma 4.4 d times, we can integrate a basic function $F(x_1,\ldots,x_d)$ in linear time (after sorting) and thus solve Problem 4.1 in the $\widehat{B} = \emptyset$ case.

More generally, when $\widehat{B} \ne \emptyset$, we invoke Observation 4.3. We describe a way to replace the basic function F with a new function $\widetilde{F}$ that has smaller complexity, such that the integrals of $F(x_1,\ldots,x_d)$ and $\widetilde{F}(x_1,\ldots,x_d)$ over the complement of the union of $\widehat{B}$ coincide.

Let X_i be the set of x_i-coordinate values that appears among the vertices of $\widehat{B}$. Draw an axis-aligned hyperplane $\{(x_1,\ldots,x_d) \mid x_i = \alpha\}$ for every $\alpha \in X_i$ and for every $i \in \{1,\ldots,d\}$; the result is a grid. Since the union of $\widehat{B}$ is a union of selected cells from the grid, it suffices to ensure that the integrals of F and $\widetilde{F}$ over every grid cell coincide.

To this end, let $\pi_i^-(x)$ be the largest value in X_i that is less than x, and $\pi_i^+(x)$ be the smallest value in X_i that is greater than x; these "successor" and "predecessor" functions are monotone step functions. Our construction of $\widetilde{F}$ is as follows:

$$\widetilde{F}(x_1,\ldots,x_d) = \tfrac{1}{(\pi^+(x_1)-\pi^-(x_1))\cdots(\pi^+(x_d)-\pi^-(x_d))} \int_{\pi_i^-(x_1)}^{\pi_1^+(x_1)} \cdots \int_{\pi_d^-(x_d)}^{\pi_d^+(x_d)} F(\xi_1,\ldots,\xi_d)\, d\xi_d \cdots d\xi_1.$$

To understand the "magic" behind this formula, first note that within any grid cell, $\pi^-(x_i)$ and $\pi^+(x_i)$ are constants for all i, and thus $\widetilde{F}$ is constant as well. The volume of the cell is $(\pi^+(x_1) - \pi^-(x_1))\cdots(\pi^+(x_d) - \pi^-(x_d))$. So the integral of $\widetilde{F}$ over the cell is equal to $\int_{\pi_i^-(x_1)}^{\pi_1^+(x_1)} \cdots \int_{\pi_d^-(x_d)}^{\pi_d^+(x_d)} F(\xi_1,\ldots,\xi_d)\, d\xi_d \cdots d\xi_1$, i.e., it is equal to the integral of F over the cell—exactly as desired!

We claim that $\widetilde{F}$ is a sum of $O(1)$ basic functions. To see this, first note that $\int_{\pi_d^-(x_d)}^{\pi_d^+(x_d)} F(x_1,\ldots,x_{d-1},\xi)\, d\xi = \int_{-\infty}^{\pi_d^+(x_d)} F(x_1,\ldots,x_{d-1},\xi)\, d\xi - \int_{-\infty}^{\pi_d^-(x_d)} F(x_1,\ldots,x_{d-1},\xi)\, d\xi$ is a sum of $O(1)$ basic functions by Lemma 4.4 and by composition with the monotone step functions π_d^- and π_d^+. Repeating d times and multiplying the monotone step function $\frac{1}{\pi^+(x_i)-\pi^-(x_i)}$ with the density function for x_i for each i yield the claim. Furthermore, since $\pi_i^-(x_i)$ and $\pi_i^+(x_i)$ take on only $O(|\widehat{B}|)$ different values, all the monotone step functions arising in the basic predicates for $\widetilde{F}$ have complexity $O(|\widehat{B}|)$ after composition with the π_i^-'s and π_i^+'s; the density functions also have complexity $O(|\widehat{B}|)$ and are step functions with degree 0 after composition with the π_i^-'s and π_i^+'s. Finally, we can invoke Observation 4.3 to get back $O(1)$ instances of the measure problem with $O(|\widehat{B}|)$ input size. To summarize, we have proved the following:

Lemma 4.5 *Let B_* be the subset of all boxes of B that are of the form $\{(x_1,\ldots,x_d) \mid x_i \ ?\ \llcorner\!\lrcorner\}$ or $\{(x_1,\ldots,x_d) \mid (x_i \ ?\ \llcorner\!\lrcorner) \wedge (x_j \ ?\ \llcorner\!\lrcorner)\}$ when restricted to Γ. Let $\widehat{B} = B - B_*$. In $O(n)$ time after sorting, we can reduce the measure problem for B to $O(1)$ instances of the measure problem where in each instance the boxes in B_* are replaced by $O(|\widehat{B}|)$ boxes of the same form.*

Remarks. Bringmann [7] described a similar procedure for computing the measure of B_*, but not for the simplification of B_* when there are additional boxes in $\widehat{B}$. Our functional approach is thus more powerful.

4.2 An $\widetilde{O}(n^{d/3})$-Time Algorithm for Arbitrary Orthants

We are now ready to present our algorithm for the case when the input boxes are arbitrary orthants. We follow the same basic approach from Section 2.

How to simplify. We apply the procedure from Lemma 4.5. Observe that the unsimplified boxes in $\widehat{B}$ must have at least one $(d-3)$-face intersecting Γ. Let $T(n, N_{d-3})$ be the running time needed for an input where n is an upper bound on the input size and N_{d-3} is an upper bound on the total weight of all $(d-3)$-faces of B intersecting Γ. Lemma 4.5 yields

$$T(n, N_{d-3}) \ \le\ O(1)\, T(O(N_{d-3}), N_{d-3}) + O(n).$$

How to cut. We use the cutting step from Section 2, but now with the weighted median of the $(d-3)$-faces. The weight of a $(d-3)$-face orthogonal to the i-th, j-th, and k-th axis is defined as $2^{(i+j+k)/d}$. By the same argument, N_{d-3} decreases by a factor of $2^{3/d}$ in each subcell, and so

$$T(n, N_{d-3}) \ \le\ 2\,T(n, N_{d-3}/2^{3/d}) + O(n).$$

Analysis. Suppose that we apply the simplification procedure only after roughly every $\log_2(r^d)$ levels of cutting. Then setting $T(N) = T(cN, N)$ for a suitable constant c, we obtain

$$T(N) \ \le\ O(r^d)\,T(N/r^3) + O(r^d N).$$

By choosing the parameter $r = N^\delta$ for a small constant $\delta > 0$, the recurrence solves to $T(N) = O(N^{d/3}\log^{O(1)} N)$ for $d \ge 4$.

Remarks. $n^{d/3}$ seems to be the best possible with this approach. To get $n^{d/4}$, we would need to simplify 3-sided orthants and work with monotone functions in two variables, but these functions may not have nice inverses.

4.3 An $\widetilde{O}(n^{(d+1)/3})$-Time Algorithm for Arbitrary Hypercubes

Finally, we present our algorithm for the case when the input boxes are arbitrary hypercubes.

How to simplify. We apply the procedure from Lemma 4.5. Some boxes in $\widehat{B}$ may be of the form $\{(x_1,\dots,x_d) \mid (\llcorner \le x_i \le \llcorner) \wedge (x_j \ ?\ \llcorner)\}$ or $\{(x_1,\dots,x_d) \mid (\llcorner \le x_i \le \llcorner) \wedge (\llcorner \le x_j \le \llcorner)\}$ when restricted to Γ; we say that these boxes are *bad*, and their faces are similarly bad. All other boxes in $\widehat{B}$ must have at least one $(d-3)$-face intersecting Γ.

Let $T(n, n_{d-2}, n'_{d-2}, n_{d-3})$ be the running time needed for an input where n is an upper bound on the input size, n_i is an upper bound on the number of i-faces in B intersecting Γ for $i \in \{d-2, d-3\}$, and n'_{d-2} is an upper bound on the number of bad $(d-2)$-faces in B intersecting Γ. Lemma 4.5 yields

$$T(n, n_{d-2}, n'_{d-2}, n_{d-3}) \ \le\ O(1)\,T\left(O(n'_{d-2} + n_{d-3}), O(n'_{d-2} + n_{d-3}), n'_{d-2}, n_{d-3}\right) + O(n). \qquad (5)$$

(One technicality: it is better to avoid adjusting coordinate values, but instead adjust density functions, to ensure that the input boxes remain hypercubes.)

How to cut. For the cutting step, instead of weighted medians, we switch back to Overmars and Yap's original approach [30]:

Lemma 4.6 *Given a set of n axis-parallel flats (of different dimensions) inside a d-dimensional space, and given a parameter r, we can divide space into $O(r^d)$ cells so that each cell intersects $O(n/r^{d-i})$ i-flats. The construction takes $O(r^d n)$ time.*

Proof: We adapt the presentation from the author's previous paper [11] of Overmars and Yap's approach: Vertically project the input onto the first $d-1$ dimensions and recursively construct the partition. Then lift each cell to get a vertical column γ along the d-th dimension. Partition γ with $O(r)$ cuts using hyperplanes orthogonal to the d-th axis, so that the number of i-flats orthogonal to the

d-th axis and intersecting each subcell is a factor of r less than that for γ, for each $i \in \{0, \ldots, d-1\}$. Clearly, the total number of cells in this construction is $O(r^d)$.

Let $n_i^{(d)}$ be the maximum number of i-flats intersecting each cell in this d-dimensional construction. If an i-flat f is not orthogonal to the d-th axis, the vertical projection of f is an $(i-1)$-flat. If f is orthogonal to the d-th axis, the vertical projection of f is an i-flat. We therefore have

$$n_i^{(d)} \le n_{i-1}^{(d-1)} + \frac{n_i^{(d-1)}}{r}.$$

With the trivial base cases, it follows by induction that $n_i^{(d)} = O(n/r^{d-i})$. $\square$

The lemma immediately implies

$$T(n, n_{d-2}, n'_{d-2}, n_{d-3}) \;\le\; O(r^d)\, T\left(n, O\left(\frac{n_{d-2}}{r^2}\right), O\left(\frac{n_{d-2}}{r^2}\right), O\left(\frac{n_{d-3}}{r^3}\right)\right) + O(r^d n). \tag{6}$$

How to cut: another option. The preceding cutting procedure is not specialized to reduce the number n'_{d-2} of bad $(d-2)$-faces. To this end, we suggest another cutting procedure. The idea is to cut along the longest dimension of Γ. More precisely, let $\ell_i(b)$ denote the x_i-length of a box b; for a hypercube b, we can drop the subscript. Let i^* maximize $\ell_{i^*}(\Gamma)$. Compute $r-1$ quantiles among the x_{i^*}-th coordinate values of the bad $(d-2)$-faces. Cut Γ into r subcells using a hyperplane $x_{i^*} = m$ at each quantile m.

Consider a bad box b that is equivalent to a box of the form $\{(x_1, \ldots, x_d) \mid (\text{␣} \le x_i \le \text{␣}) \wedge (x_j \;?\; \text{␣})\}$ or $\{(x_1, \ldots, x_d) \mid (\text{␣} \le x_i \le \text{␣}) \wedge (\text{␣} \le x_j \le \text{␣})\}$ when restricted to Γ, where the x_i-projection of b lies in the x_i-projection of Γ. Then $\ell(b) < \ell_i(\Gamma)$. On the other hand, $\ell(b) \ge \ell_k(\Gamma)$ for all $k \in \{1, \ldots, d\} - \{i, j\}$. It follows that i^* is either i or j. Thus, after cutting into r subcells along the i^*-th axis, the number of bad $(d-2)$-faces intersecting each subcell drops by a factor of r. Note that some boxes that have $(d-3)$-faces intersecting Γ could now become bad in a subcell. This alternative cutting procedure yields

$$T(n, n_{d-2}, n'_{d-2}, n_{d-3}) \;\le\; r\, T\left(n, n_{d-2}, \frac{n'_{d-2}}{r} + O(n_{d-3}), n_{d-3}\right) + O(rn). \tag{7}$$

Analysis. Setting $T(n) = T(n, n, n, n)$ and applying (6), (7), and (5) in that order, we obtain

$$T(n) \;\le\; O(r^{d+1})\, T(O(n/r^3)) + O(r^{d+1} n).$$

By choosing the parameter $r = n^\delta$ for a small constant $\delta > 0$, the recurrence solves to $T(n) = O(n^{(d+1)/3} \log^{O(1)} n)$ for $d \ge 3$.

Remarks. The reason we revert to Overmars and Yap's less elegant approach instead of our weighted-median cutting approach is that we need to control more than one parameter simultaneously (n_{d-2} and n_{d-3}). Overmars and Yap's approach has extra constant factors in the number of subproblems in (6), which causes extra polylogarithmic factors during recursion, but this is inconsequential, since there is already an extra constant factor in the number of subproblems in (5).

To summarize in a few words, the improvement over Bringmann's algorithm stems from the same basic idea from our algorithm in Section 2: namely, simplify the input before recursion.

References

[1] P. K. Agarwal. An improved algorithm for computing the volume of the union of cubes. In *Proc. 26th Sympos. Comput. Geom.*, pages 230–239, 2010.

[2] P. K. Agarwal, H. Kaplan, and M. Sharir. Computing the volume of the union of cubes. In *Proc. 23rd Sympos. Comput. Geom.*, pages 294–301, 2007.

[3] S. Albers and T. Hagerup. Improved parallel integer sorting without concurrent writing. *Inf. Comput*, 136:25–51, 1997.

[4] J. L. Bentley. Algorithms for Klee's rectangle problem. Unpublished manuscript, 1977.

[5] B. K. Bhattacharya and Q. Shi. Application of computational geometry to network p-center location problems. In *Proc. 20th Canad. Conf. Comput. Geom.*, pages 119–122, 2008.

[6] J.-D. Boissonnat, M. Sharir, B. Tagansky, and M. Yvinec. Voronoi diagrams in higher dimensions under certain polyhedral distance functions. *Discrete Comput. Geom.*, 19:473–484, 1998.

[7] K. Bringmann. An improved algorithm for Klee's measure problem on fat boxes. *Comput. Geom. Theory Appl.*, 45:225–233, 2012.

[8] K. Bringmann. Bringing order to special cases of Klee's measure problem. Technical report, 2013.

[9] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete Comput. Geom.*, 22:547–567, 1999.

[10] T. M. Chan. Semi-online maintenance of geometric optima and measures. *SIAM J. Comput.*, 32:700–716, 2003.

[11] T. M. Chan. A (slightly) faster algorithm for Klee's measure problem. *Comput. Geom. Theory Appl.*, 43:243–250, 2010.

[12] B. Chazelle and J. Matousek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21:579–597, 1996.

[13] L. P. Chew, D. Dor, A. Efrat, and K. Kedem. Geometric pattern matching in d-dimensional space. *Discrete Comput. Geom.*, 21:257–274, 1999.

[14] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17:830–847, 1988.

[15] D. Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11:467–471, 1982.

[16] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, Germany, 3rd edition, 2008.

[17] D. P. Dobkin, D. Eppstein, and D. P. Mitchell. Computing the discrepancy with applications to supersampling patterns. *ACM Trans. Graph.*, 15:354–376, 1996.

[18] A. Dumitrescu, J. S. B. Mitchell, and M. Sharir. Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles. *Discrete Comput. Geom.*, 31:207–227, 2004.

[19] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete Comput. Geom.*, 11:321–350, 1994.

[20] D. Eppstein and S. Muthukrishnan. Internet packet filter management and rectangle geometry. In *Proc. 12th ACM–SIAM Sympos. Discrete Algorithms*, pages 827–835, 2001.

[21] G. S. Frandsen, J. P. Hansen, and P. B. Miltersen. Lower bounds for dynamic algebraic problems. *Inf. Comput.*, 171:333–349, 2001.

[22] M. L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5:49–60, 1976.

[23] F. Le Gall. Faster algorithms for rectangular matrix multiplication. In *Proc. 53rd IEEE Sympos. Found. Comput. Sci.*, pages 514–523, 2012.

[24] J. Galtier and P. Penna. Complexity links between matrix multiplication, Klee's measure, and call access control for satellite constellations. Technical report, INRIA, 2001.

[25] S. Har-Peled, V. Koltun, D. Song, and K. Y. Goldberg. Efficient algorithms for shared camera control. In *Proc. 19th Sympos. Comput. Geom.*, pages 68–77, 2003.

[26] H. Kaplan, N. Rubin, M. Sharir, and E. Verbin. Efficient colored orthogonal range counting. *SIAM J. Comput.*, 38:982–1011, 2008.

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

[28] S. Meiser. Point location in arrangements of hyperplanes. *Inf. Comput.*, 106:286–303, 1993.

[29] F. Meyer auf der Heide. A polynomial linear search algorithm for the n-dimensional knapsack problem. *J. ACM*, 31:668–676, 1984.

[30] M. Overmars and C.-K. Yap. New upper bounds in Klee's measure problem. *SIAM J. Comput.*, 20:1034–1045, 1991.

[31] M. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. *J. Algorithms*, 13:99–113, 1992.

[32] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[33] M. Thorup. Randomized sorting in $O(n \log \log n)$ time and linear space using addition, shift, and bit-wise boolean operations. *J. Algorithms*, 42:205–230, 2002.

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

[35] V. Vassilevska. Efficient algorithms for clique problems. *Inform. Process. Lett.*, 109:254–257, 2009.

[36] H. Yildiz and S. Suri. On Klee's measure problem for grounded boxes. In *Proc. 28th Sympos. Comput. Geom.*, pages 111–120, 2012.