A (Slightly) Faster Algorithm for Klee's Measure Problem

Timothy M. Chan*

January 2, 2009

Abstract

Given n axis-parallel boxes in a fixed dimension $d \geq 3$, how efficiently can we compute the volume of the union? This standard problem in computational geometry, commonly referred to as *Klee's measure problem*, can be solved in time $O(n^{d/2} \log n)$ by an algorithm of Overmars and Yap (FOCS 1988). We give the first (albeit small) improvement: our new algorithm runs in time $n^{d/2} 2^{O(\log^* n)}$, where $\log^*$ denotes the iterated logarithm.

For the related problem of computing the *depth* in an arrangement of n boxes, we further improve the time bound to near $O(n^{d/2} / \log^{d/2-1} n)$, ignoring $\log \log n$ factors. Other applications and lower-bound possibilities are discussed. The ideas behind the improved algorithms are simple.

1 Introduction

In this paper, we revisit a basic geometric problem: how to compute the measure (the volume) of the union of n axis-parallel boxes in a constant dimension d.

The problem made its debut in a 2-page article by V. Klee [23], who posed the question of whether a $o(n \log n)$ algorithm exists for the $d = 1$ case of intervals (we now know the answer is negative in the standard algebraic decision tree model [20]). Of more eventual significance in the article is a brief statement of the extension of the measure problem to d-dimensional boxes. Bentley [4] gave the first optimal algorithm for $d = 2$ with $O(n \log n)$ running time; in higher dimensions, the running time is $O(n^{d-1} \log n)$. Notice that this is faster than explicitly constructing the union, which has worst-case combinatorial complexity $\Theta(n^d)$ for arbitrary axis-parallel boxes. Van Leeuwen and Wood [24] subsequently removed the logarithmic factor for $d \geq 3$ and obtained the time bound $O(n^{d-1})$. In FOCS'88, Overmars and Yap [29] obtained the fastest algorithm to date for $d \geq 3$, with running time $O(n^{d/2} \log n)$.

Better results are possible in some special cases. For instance, for unit hypercubes (or more generally "fat" boxes of roughly equal sizes), the union has combinatorial complexity $O(n^{\lfloor d/2 \rfloor})$ [5] and can be constructed explicitly in $O(n^{\lfloor d/2 \rfloor} \operatorname{polylog} n)$ expected time [22]. This beats $O(n^{d/2} \log n)$ for odd dimensions $d \geq 3$. The author [9] has also obtained faster algorithms for unit hypercubes in even dimensions: the time bound is $O(n^{3/2} \operatorname{polylog} n)$ for $d = 4$ and $O(n^{\lceil d/2 \rceil - 1 + \frac{1}{\lceil d/2 \rceil}} \operatorname{polylog} n)$ for $d \geq 6$ (the latter requires Kaplan *et al.*'s union decomposition subroutine [22]). At last year's SoCG, Agarwal, Kaplan, and Sharir [1] obtained an improved result for another special case: for cubes of

*School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (tmchan@uwaterloo.ca).
This work has been supported by NSERC. A preliminary version of this paper has appeared in *Proc. 24th ACM Sympos. Comput. Geom.*, 2008.

arbitrary size (or more generally fat boxes) in dimension $d = 3$, they described an $O(n^{4/3} \log n)$-time algorithm.

Unfortunately, there have been no improved results for the general problem for arbitrary axis-parallel boxes. The key test case is $d = 3$, where Overmars and Yap's algorithm runs in $O(n^{3/2} \log n)$ time. Trying to reduce the 3/2 exponent might turn out to be futile, if one considers the complete lack of progress over the last twenty years. Indeed, algorithms with $n^{3/2}$-like complexities (e.g., see [14, 17, 21]) have since been found for many related problems about axis-parallel rectangles and boxes, and $n^{3/2}$ factors could very well be inherent to these types of problems. (See Section 5 for a more rigorous discussion on lower bounds.)

Thus, time seems ripe to consider the more approachable question of whether the $\log n$ factor can be reduced—a question that was specifically raised at the end of Overmars and Yap's paper, and to our knowledge, has remained unanswered till now. We give a new algorithm that almost completely eliminates the $\log n$ factor. Our running time is $n^{3/2} 2^{O(\log^* n)}$ for $d = 3$, and more generally, $n^{d/2} 2^{O(\log^* n)}$ in higher dimensions. Here, $\log^*$ is the (very slow-growing) iterated logarithm function.

Our improved algorithm is described in Section 2 for $d = 3$, and Section 4 for higher d; it is quite simple conceptually. The occurrence of iterated logarithms is rather interesting, but not unprecedented in geometric algorithms and data structures (e.g., $2^{O(\log^* n)}$ factors can be seen in certain partition-tree constructions by Matoušek [25, 26], or his algorithm for another eponymic favorite, Hopcroft's problem [27]).

Techniques developed for such a basic problem naturally have consequences to other problems in computational geometry. One related problem, for example, is computing the *depth* in the arrangement of n axis-parallel boxes in $\mathbb{R}^d$: the depth of a point p is defined as the number of boxes containing p, and the depth of the arrangement is defined as the maximum depth over all points in $\mathbb{R}^d$. A special case of both the measure and the depth problem is the *coverage* problem: deciding whether the union of n axis-parallel boxes inside a fixed box γ_0 covers all of γ_0. (To reduce coverage to depth, replace each input box with its complement, which can be decomposed into $O(1)$ boxes; then test whether the depth is n.)

Adapting Overmars and Yap's algorithm yields the current best result for both the depth and the coverage problem in 3-d. Adapting our faster algorithm immediately leads to improved results. In fact, we are even able to break the $n^{3/2}$ barrier for depth (and thus coverage)—we describe an algorithm with running time $O((n^{3/2}/\sqrt{\log n}) \operatorname{polyloglog} n)$ for $d = 3$ in Section 3. In higher dimensions, the bound is $O((n^{d/2}/\log^{d/2-1} n) \operatorname{polyloglog} n)$.

Getting $o(n^{d/2})$ is certainly interesting, though similar examples of logarithmic factor speedups exist outside of computational geometry (e.g., all-pairs shortest paths [10] and 3SUM [2]). We stress that the assumed model of computation is the *standard* RAM with $\log n$ word size. (Indeed, for the depth or coverage problem, we can normalize the coordinates to integers in $\{1, \ldots, O(n)\}$ by an initial pre-sorting in $O(n \log n)$ time; so real vs. integer input is a non-issue.)

For further applications, we mention two examples:

- Given n points in $\mathbb{R}^3$ and a number k, we want to find a subset of size k with the minimum L_∞-diameter. Eppstein and Erickson [16] gave an $O(n^{3/2} \log^2 n)$-time algorithm, by observing that the corresponding decision problem reduces to finding the depth in an arrangement of n unit cubes and applying Overmars and Yap's algorithm. (Agarwal *et al.*'s result on unit cubes [1] does not extend to the depth problem.) By further combining with a k-sensitizing technique of Datta *et al.* [12] and a randomized optimization technique of the author [8], the best previous time bound became $O(n \log n + n\sqrt{k} \log k)$. The new bound is $O(n \log n + n\sqrt{k/\log k}\, \mathrm{polyloglog}\, k)$. Similar speedups can be obtained in higher dimensions.

- Given two n-point sets A and B in $\mathbb{R}^3$, we want to find a translation of A that minimizes its (directed or undirected) L_∞-Hausdorff distance to B. Chew *et al.* [11] gave an $O(n^3 \log^2 n)$-time algorithm, by observing that the corresponding decision problem can be reduced to finding the depth in an arrangement of $O(n^2)$ boxes and applying Overmars and Yap's algorithm. By the author's randomized optimization technique [8], the time bound was reduced to $O(n^3 \log n)$. The new bound is $O((n^3/\sqrt{\log n})\, \mathrm{polyloglog}\, n)$.

Finally, in Section 5, we point to the difficulty of the measure problem by noting some surprising but simple connections with matrix-multiplication-related problems. One reduction implies an $\Omega(\sqrt{n})$ lower bound for the dynamic 2-d measure problem in some general models of computation; all known approaches to solving Klee's measure problem in 3-d amount to designing a dynamic 2-d data structure. Another reduction suggests that it might not be possible to beat $n^{d/2}$ by more than polylogarithmic factors if one uses only "purely combinatorial" techniques.

2 The 3-d Algorithm

For clarity of exposition, we present our solution to the 3-d Klee's measure problem first and discuss extensions to higher dimensions afterwards. As is well known [3, 29], the static Klee's measure problem reduces to the offline dynamic problem in one dimension lower by a standard sweep algorithm. It suffices to present a data structure for the offline dynamic 2-d problem—maintaining the area of the union of a set R of (axis-aligned) rectangles in the plane, under a pre-given sequence of insertions and deletions of rectangles. Actually, we only need to assume that the vertices of the rectangles are pre-given, not the update sequence itself.

We slightly generalize the problem to maintaining the portion of the area within a fixed rectangle γ_0, under the assumption that we are pre-given the set V_0 of all points strictly inside γ_0 that are vertices of rectangles from the update sequence. Let $U(n, N)$ represent the (amortized) update time required to solve the problem where $|R| < N$ at all times and $|V_0| < n$, with $n = O(N)$. (To deal with degeneracies, vertices with the same coordinates but from different rectangles are treated as distinct.) The time required to solve the original static 3-d problem is $O(n)\, U(O(n), O(n))$.

2.1 Overmars and Yap's approach

We begin with a quick review/reinterpretation of Overmars and Yap's solution to the dynamic 2-d problem. The key is a simple partitioning result, stated in the observation below in a slightly generalized form (the original version has $b = 1$). An *orthogonal BSP tree* refers to a binary tree of rectangular cells where the cell at the root is the entire space and the cell at each node is partitioned into the two cells at the children via an axis-aligned hyperplane (horizontal/vertical line in the 2-d case).

Proposition 2.1 *Given n points in $\mathbb{R}^2$ and a parameter $1 \le b \le n$, we can construct an orthogonal BSP tree of size $O(n)$ and height $O(\log(n/b))$ in $O(n \log n)$ time, such that*

(i) *each leaf cell contains* $< b$ *points;*

(ii) *each vertical or horizontal line intersects* $O(\sqrt{n/b})$ *leaf cells.*

Proof: *Option 1* (Overmars and Yap): Partition the plane via vertical cuts into $O(\sqrt{n/b})$ slabs each containing $< \sqrt{bn}$ points. Then partition each slab via horizontal cuts into $O(\sqrt{n/b})$ cells each containing $< b$ points. Clearly, the resulting leaf cells satisfy (i) and (ii).

Option 2 (folklore): Just use the standard k-d tree. Nodes with fewer than b points are pruned. We can verify (ii) by noting that the recurrence $T(n) = 2T(n/4)$ if $n \geq b$ and $T(n) = 1$ if $n < b$ solves to $T(n) = O(\sqrt{n/b})$. □

We can solve the dynamic 2-d problem using the BSP tree as follows:

Lemma 2.2 $U(n, N) \ \leq \ O(\sqrt{n/b})\, U(b, N) \ + \ O(\sqrt{n/b} \log n)$.

Proof: We apply Proposition 2.1 to the point set V_0. The amortized construction cost is $O(\log n)$ per update, since the number of updates is $\Omega(n)$. W.l.o.g., we can make γ_0 the root cell (whose parent cell is the entire plane by default).

For each cell γ, let $R[\gamma]$ be the subset of all rectangles intersecting γ but not completely containing γ. We maintain $\text{AREA}[\gamma]$, the portion of the area of the union of $R[\gamma]$ within γ. For each child γ_i ($i \in \{1, 2\}$) of γ, we further maintain $\text{COUNT}[\gamma_i]$, the number of rectangles in $R[\gamma]$ that completely contain γ_i.

Since a rectangle in $R[\gamma]$ must have at least one of its 4 bounding lines intersecting γ, each rectangle participates in $R[\gamma]$ for at most $O(\sqrt{n/b})$ leaf cells γ by Proposition 2.1(ii), and thus for $O(\sqrt{n/b} \log(n/b))$ cells γ along $O(\sqrt{n/b})$ paths in the tree. When inserting/deleting a rectangle, all COUNT values can be updated in $O(\sqrt{n/b} \log(n/b))$ total time. All $\text{AREA}[\gamma]$ values for leaf cells γ can be updated in $O(\sqrt{n/b}) U(b, N)$ time, since each leaf cell has $< b$ vertices. Since $\text{AREA}[\gamma]$ can be deduced from $\text{AREA}[\gamma_1]$, $\text{AREA}[\gamma_2]$, $\text{COUNT}[\gamma_1]$, and $\text{COUNT}[\gamma_2]$ in constant time, all $\text{AREA}[\gamma]$ values for internal node cells γ can be updated in $O(\sqrt{n/b} \log(n/b))$ time. The final answer can be deduced from $\text{AREA}[\gamma_0]$ (and $\text{COUNT}[\gamma_0]$). □

For the base case, Overmars and Yap observed that $U(1, N) = O(\log N)$: when there are no vertices inside a cell γ_0, the union within γ_0 forms a "trellis" pattern (see Figure 1 (left)) and its area can be determined from lengths of two unions of intervals in 1-d, by projecting the rectangles to the two axes. These 1-d subproblems can be solved in logarithmic time by adapting standard search trees. Setting $b = 1$ in Lemma 2.2, we immediately get $U(n, N) = O(\sqrt{n} \log N)$.

It is possible to eliminate the logarithmic factor in the second term of Lemma 2.2 (if Option 2 is taken in the proof of Proposition 2.1). The real obstacle for improvement is actually in the 1-d search subproblems at the base cases. Sublogarithmic bounds for the dynamic 1-d measure problem are not known for real coordinates, even when coordinates have been pre-sorted. We could try to handle multiple 1-d subproblems for multiple adjacent leaf cells faster by exploiting coherence somehow. It turns out there is an even simpler approach to improvement: just use a larger (nonconstant) value of b, and recurse!

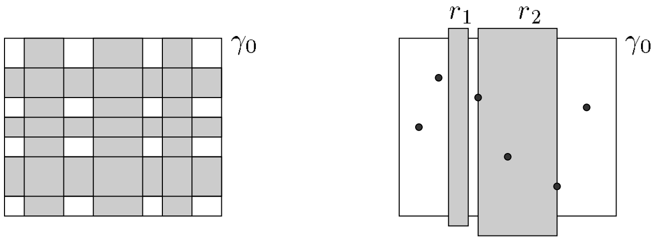

Figure 1: (Left) No vertices inside γ_0: the union forms a "trellis". (Right) $< b$ vertices inside γ_0, shown with a thin x-long rectangle r_1 and a normalized x-long rectangle r_2.

2.2 The new idea

In order to make recursion effective, we introduce one new ingredient: a way to reduce N (the number of rectangles) to a function of n (the number of vertices strictly inside γ_0).

Lemma 2.3 $U(n, N) \;\leq\; O(1)\, U(n, O(n^2)) \;+\; O(\log N)$.

Proof: Call a rectangle r *short* if some vertex is strictly inside γ_0. Call r *x-long* (resp. *y-long*) if only the vertical (resp. horizontal) edges can intersect γ_0. The number of short rectangles is $O(n)$. We will describe how to reduce the number of x-long rectangles; the y-long rectangles can be handled similarly.

Draw $O(n)$ vertical lines at the x-coordinates of V_0 to divide γ_0 into $O(n)$ *column* rectangles. Call a x-long rectangle r *normalized* if its left and right edges are on these vertical lines. Call r *thin* if it is contained inside one of these columns. See Figure 1 (right). We can decompose each x-long rectangle into at most two thin rectangles and one normalized rectangle. The number of normalized rectangles, after removal of duplicates, is at most $O(n^2)$. It suffices to reduce the number of thin rectangles.

Let R_i denote the subset of all thin rectangles inside column i. We replace all rectangles in R_i by a single *representative* thin rectangle r_i, which is x-long and placed inside column i. Specifically, the width of r_i is set to the length of the union of the x-projections of R_i (the exact placement of r_i will not matter). It is easy to see that the actual area of the union within γ_0 is preserved through this replacement (since there are no vertices strictly inside a column). The number of representative thin rectangles is $O(n)$, so the reduced set of rectangles, denoted $\widehat{R}$, has size at most $O(n^2)$.

To insert/delete an x-long rectangle, we may have to perform one insertion/deletion of a normalized rectangle in $\widehat{R}$. Specifically, we maintain a counter on the number of occurrences of each normalized rectangle and perform the insertion (resp. deletion) only when the counter is 0 (resp. 1). In addition, we have to change the widths of at most two representative thin rectangles. The values of the widths themselves can be updated through a data structure for the dynamic 1-d measure problem in logarithmic time. A representative rectangle can be updated by performing a deletion followed by an insertion in $\widehat{R}$. $\square$

By applying the two lemmas in tandem, we get

$$U(n, N) \;\leq\; O(\sqrt{n/b})\, U(b, O(b^2)) \;+\; O(\sqrt{n/b}\log N).$$

Choosing $b = \log^2 n$ for instance, and writing $U(n) := U(n, cn^2)$ for a sufficiently large constant c, we get the recurrence

$$U(n) \;\le\; O(\sqrt{n}/\log n)\, U(\log^2 n) \;+\; O(\sqrt{n}),$$

which solves to $U(n) \le \sqrt{n}2^{O(\log^* n)}$.

Theorem 2.4 *The 3-d measure problem can be solved in time $n^{3/2}2^{O(\log^* n)}$.*

3 Depth

We adapt the algorithm of the previous section to compute the depth of a set of boxes in 3-d. As before, the problem reduces to the corresponding offline dynamic 2-d problem.

To be precise, we want to maintain the maximum depth of a set R of 2-d rectangles within a rectangular cell γ_0, under the assumption that we are pre-given the set of all vertices strictly inside γ_0. We assume that short rectangles are distinct but allow for multiple copies of long rectangles. Let $U(n, N)$ represent the update time where the number of distinct rectangles in R (ignoring multiplicities) is $< N$ at all times and the number of vertices in V_0 is $< n$, with $n = O(N)$.

Both Lemmas 2.2 and 2.3 still hold, with straightforward modifications to the proofs. In the latter case, for instance, inserting/deleting a long rectangle causes the increment/decrement of the multiplicities of at most one normalized rectangle and two representative thin rectangles. The multiplicities of the thin rectangles can be updated through a data structure for the dynamic 1-d depth problem in logarithmic time. Note that the width of a representative thin rectangle r_i does not matter; in fact, we can expand r_i to fill its column and thereby making all long rectangles normalized rectangles. (Care has to be taken to avoid overlap at boundaries of rectangles; for example, we can make left edges closed and right edges open.)

Consequently, the $n^{3/2}2^{O(\log^* n)}$ result carries over... but we can do even better. The idea is this: when the subproblem size gets sufficiently small during the recursion, we can solve the subproblems directly using word RAM operations.

For our best improvement, we use a refinement of Lemma 2.3. Consider a variation of the 2-d dynamic problem, where we allow the following generalized updates for long rectangles: given a long normalized rectangle r and a number k, insert or delete k copies of r, i.e., increase or decrease its multiplicity by k. Here, we allow a rectangle to have negative multiplicity; the depth of a point p is defined as the sum of the multiplicities of the rectangles containing p. Let $U^*(n, N, \mu)$ denote the update time for this generalized problem, where the extra parameter μ bounds the maximum multiplicity of the rectangles (in absolute value). The lemma below improves Lemma 2.3 in the sense that the number of distinct rectangles is reduced to $O(n)$, while at the same time the maximum multiplicity is reduced to $O(n)$.

Lemma 3.1 $U(n, N) \;=\; O(1)\, U^*(n, O(n), O(n)) \;+\; O(\log N)$.

Proof: As noted above, by the proof of Lemma 2.3, we may assume that all long rectangles in R are normalized after spending logarithmic time per update. Recall that the number of short rectangles is $< n$ at any time. We will describe how to reduce the number of x-long rectangles; the y-long rectangles can be handled similarly.

After every n updates, we start a new *phase*. At the beginning of a phase, let k_i be the depth of column i w.r.t. the x-long rectangles in R (any point in the same column has the same depth because the x-long rectangles are all normalized). Let m be the column with the largest k_m. We replace all x-long rectangles from the previous phases with a set of column rectangles. Specifically, for each column i, we create a column rectangle with multiplicity $\max\{k_i - (k_m - 2n), 0\}$.

Let $\widehat{R}$ be the set of rectangles after replacement. Consider a point p in a column i with $k_i < k_m - 2n$. We claim that p cannot attain maximum depth in R or $\widehat{R}$: indeed, let p' be a point in column m with the same y-coordinate as p; the depth of p' exceeds the depth of p by $2n$ w.r.t. the x-long rectangles at the beginning of the phase, and thus by at least n at all times (since the phase has only n updates); the depths of p' and p are the same w.r.t. the y-long rectangles and differ by less than n w.r.t. the short rectangles; so p' has larger overall depth than p. It follows that points of maximum depth lie in columns i with $k_i \geq k_m - 2n$. So the maximum depth in $\widehat{R}$ is exactly the maximum depth in R minus $k_m - 2n$ at all times.

The set $\widehat{R}$ has size $O(n)$ and maximum multiplicity $O(n)$ at the beginning of the phase; this remains true at all times since a phase has only n updates.

In each phase, we have performed $O(n)$ insertions in $\widehat{R}$; by amortization, the number of insertions/deletions is $O(1)$ per update. We can compute the k_i's at the beginning of the phase as follows: We first compute the depth Δ_i of column i w.r.t. the x-long rectangles from the previous phase for all i; this is doable in $O(n)$ time by a left-to-right sweep over these $O(n)$ x-long rectangles. We then add Δ_i to the previous value of k_i for all i, in $O(n)$ time. By amortization, the cost is $O(1)$ per update. □

We now describe the improved base case. We assume a RAM model that supports two nonstandard word operations in $O(1)$ time. Let $w = \Omega(\log n)$ be the word size and let $\overline{w} = w/\log w$.

Lemma 3.2 $U^*(\overline{w}, O(\overline{w}), O(\overline{w})) = O(1)$.

Proof: We can renumber coordinates of V_0 to lie in $\{1, \ldots, \overline{w}\}$ without changing the depth problem. Any set R of $O(\overline{w})$ normalized rectangles with $O(\overline{w})$ maximum multiplicity can be encoded in $O(\overline{w}\log\overline{w}) = O(w)$ bits, i.e., $O(1)$ words. An update (involving a rectangle and a number) can be encoded in $O(\log\overline{w})$ bits, or $O(1)$ words. We simply create two operations: (i) given a set R and an update, return the new set R'; (ii) given a set R, return its maximum depth. The lemma follows. □

Lemmas 3.1 and 3.2 imply that $U(\overline{w}, O(\overline{w}^2)) = O(\log\overline{w})$. By applying Lemmas 2.2 and 2.3 twice with $b \geq \overline{w}$, we get:

$$\begin{aligned} U(n,n) &\leq O(\sqrt{n/b})\left[O(\sqrt{b/\overline{w}})\, U(\overline{w}, O(\overline{w}^2)) + O(\sqrt{b/\overline{w}}\log b)\right] + O(\sqrt{n/b}\log n) \\ &= O(\sqrt{n/\overline{w}}\log b + \sqrt{n/b}\log n). \end{aligned}$$

Setting $b = \overline{w}^4$ for instance yields $U(n,n) \leq O(\sqrt{n/\overline{w}}\log\overline{w}) = O(\sqrt{n/w}\log^{3/2} w)$.

We conclude that the 3-d depth problem can be solved in time $O((n^{3/2}/\sqrt{w})\log^{3/2} w)$. For $w = \Theta(\log n)$, we can work entirely in the standard RAM model by using table lookups to simulate the two nonstandard operations. Initially, we build a table storing all possible answers in time $2^{O(w)}$, which is sublinear if we set $w = \alpha\log n$ for a sufficiently small constant $\alpha > 0$.

Theorem 3.3 *The 3-d depth problem can be solved in time $O((n^{3/2}/\sqrt{\log n})\log^{3/2}\log n)$.*

Remarks: With more effort, the $\log \log n$ factors can probably be lowered somewhat. For instance, we can solve the dynamic 1-d depth subproblems in time $O(\log n / \log w)$ instead of $O(\log n)$ by adapting known RAM data structures of Dietz [13].

It is unclear whether further logarithmic factor speedup beyond $\sqrt{\log n}$ is possible. One idea is to handle multiple $(O(\overline{w}))$ updates simultaneously using word operations, but the details appear tricky.

A related question is whether $o(n^{3/2})$ time is possible for the 3-d measure problem on the transdichotomous word RAM, i.e., assuming that coordinates are w-bit integers. We don't even know of a faster $(o(\log n))$ solution to the dynamic 1-d measure problem for integer input.

4 Higher Dimensions

We describe the changes involved in extending the results of the previous sections to a constant dimension $d > 3$. The right analog for Proposition 2.1 is stated below. Overmars and Yap originally gave a proof for the original case $b = 1$, which we redescribe for the sake of completeness (see also [15]).

Proposition 4.1 *Given n axis-aligned $(d-2)$-flats in $\mathbb{R}^d$ and a parameter $1 \leq b \leq n$, we can construct an orthogonal BSP tree of size $O(n^{d/2})$ and height $O(\log(n/b))$ in $O(n^{d/2})$ time, such that*

(i) *each leaf cell intersects $< b$ $(d-2)$-flats;*

(ii) *each axis-aligned hyperplane intersects $O((n/b)^{(d-1)/2})$ leaf cells.*

Proof: Fix a parameter r. We describe more generally a BSP construction for a set consisting of both axis-aligned $(d-2)$-flats and $(d-1)$-flats (hyperplanes), of total size n: First project the flats along the d-th axis and recursively construct a BSP tree for the projected $(d-3)$-flats and $(d-2)$-flats in $\mathbb{R}^{d-1}$. Lift each cell to get a vertical prism γ in $\mathbb{R}^d$. Partition γ with $O(r)$ horizontal cuts so that the number of intersecting horizontal $(d-2)$-flats for each subcell is a factor of r less than that for γ, and the number of intersecting horizontal $(d-1)$-flats for each subcell is a factor of r less than that for γ.

Clearly, the total number of cells in this construction is $O(r^d)$, and the number of cells intersecting any axis-aligned hyperplane is $O(r^{d-1})$.

Let f_d be the maximum number of $(d-2)$-flats intersecting a cell and g_d be the maximum number of $(d-1)$-flats intersecting a cell in this d-dimensional construction. As vertical $(d-2)$-flats project to $(d-3)$-flats and horizontal $(d-2)$-flats project to $(d-2)$-flats, we have $f_d \leq f_{d-1} + g_{d-1}/r$. As vertical $(d-1)$-flats project to $(d-2)$-flats, we have $g_d \leq g_{d-1} + n/r$. With the trivial base cases, the recurrences imply $g_d = O(n/r)$ and $f_d = O(n/r^2)$. Setting $r = \Theta(\sqrt{n/b})$ yields the result. $\square$

For the dynamic d-dimensional Klee's measure problem, V_0 now becomes a set of $(d-2)$-faces. Lemma 2.2 changes to:

$$U(n, N) \;\leq\; O(n/b)^{(d-1)/2}\, U(b, N) \;+\; O((n/b)^{(d-1)/2} \log(n/b)).$$

Lemma 2.3 still holds with minor changes to the proof. For instance, a box is now called *short* if some $(d-2)$-face intersects γ_0; a box is *x_j-long* if only the $(d-1)$-faces orthogonal to the j-th axis can intersect γ_0. Consequently, we get $U(n) \leq n^{(d-1)/2} 2^{O(\log^* n)}$.

For the dynamic d-dimensional depth problem, Lemmas 3.1 and 3.2 also still hold with no major changes to their proofs. Consequently, we get $U(n,n) \leq O((n/\overline{w})^{(d-1)/2}\log\overline{w}) = O((n/w)^{(d-1)/2}\log^{(d+1)/2} w)$.

Theorem 4.2 *The $(d+1)$-dimensional measure problem can be solved in time $n^{(d+1)/2}2^{O(\log^* n)}$. The $(d+1)$-dimensional depth problem can be solved in time $O((n^{(d+1)/2}/\log^{(d-1)/2} n)\log^{(d+1)/2}\log n)$.*

Remark: Dumitrescu, Mitchell, and Sharir [15] have shown that the $O(n^{d/2})$ size bound in Proposition 4.1 is tight for $(d-2)$-flats. This fact increases the likelihood that $n^{d/2}$ is near the best possible, at least among algorithms based on the same partitioning approach. (BSP trees with $O(n^{d/(d-j)})$ cells are known for j-flats [15], but for $j \leq d-3$, the geometry inside a leaf cell would no longer be "trellis"-like.)

5 Lower Bounds?

We now provide further evidence to suggest that $n^{d/2}$ could be the right bound for Klee's measure problem for $d = 3$ or in higher dimensions (and thus our attention on improving logarithmic factors may be warranted after all).

In the *dynamic matrix-vector multiplication* problem, we want to maintain an $n \times n$ real matrix $A = \{a_{ij}\}_{i=1,\ldots,n,j=1,\ldots,n}$ and a real vector $x = \{x_j\}_{j=1,\ldots,n}$, where the allowed query operation is to compute a given entry of the product Ax, and the allowed update operation is to change a given entry of A or a given entry of x.

The trivial solution achieves $O(1)$ update time and $O(n)$ query time. Frandsen, Hansen, and Miltersen [19] (extending previous results by Reif and Tate [30]) have established an $\Omega(n)$ lower bound on the worst-case cost per operation for this problem under various computational models. For instance, the lower bound holds in the *history-dependent algebraic computation tree* model, the *generalized real-RAM* model, and (in the case of integer input) the *word RAM* model.

The following simple reduction then implies an $\Omega(\sqrt{n})$ lower bound for the dynamic 2-d measure problem in these models.

Proposition 5.1 *If the dynamic 2-d measure problem can be solved in $U(n)$ worst-case update time, then the dynamic matrix-vector multiplication problem can be solved in $O(U(O(n^2)))$ worst-case query and update time.*

Proof: By scaling, assume that all the matrix and vector entries are between 0 and 1. Form a grid with unit side length over an $n^2 \times n$ rectangle γ_0. For each $i, j \in \{1,\ldots,n\}$, create a y-long rectangle in grid row $in + j$ of height a_{ij}. For each $i \in \{1,\ldots,n\}$, create an x-long rectangle in grid column i of width x_i. Create rectangles to cover precisely all grid cells except those at row $it + j$ and column j' ($i, j, j' \in \{1,\ldots,n\}$) with $j = j'$; this can clearly be done with $O(n^2)$ rectangles. The total number of rectangles is $O(n^2)$.

To update an entry a_{ij}, simply update the corresponding y-long rectangle. To update an entry x_i, simply update the corresponding x-long rectangle. To evaluate the i-th entry of the product, insert two temporary rectangles to cover precisely all grid rows except rows $in+1,\ldots,in+n$; then the area of the union is precisely $n^3 - n + \sum_{j=1}^{n} a_{ij}x_j$. Delete the two temporary rectangles afterwards. □

In Section 2, we have solved a version of the dynamic 2-d measure problem with $\sqrt{n}2^{O(\log^* n)}$ update time, where the coordinates inside γ_0 are known in advance (this assumption holds in the above reduction). This upper bound thus almost matches the lower bound.

On the other hand, the above reduction does not quite prove hardness of the static 3-d measure problem, because the update lower bound is worst-case (not amortized) and the update sequence is on-line (not off-line). Furthermore, in solving the static 3-d problem, we are not "required" to output the measure within each of the n vertical 2-d slices.

For lower bounds on the static problem, another simpler reduction is more relevant:

Observation 5.2 *If the static d-dimensional measure (or coverage) problem can be solved in $T_d(n)$ time, then we can decide whether an arbitrary n-vertex graph $G = (V, E)$ contains a clique of size d in $O(T_d(O(n^2)))$ time.*

Proof: Number the vertices so that $V = \{0, \ldots, n-1\}$. For each $u, v \in V$ with $uv \notin E$ and for each $i, j \in \{1, \ldots, d\}$ $(i \neq j)$, create a rectangle $\{(x_1, \ldots, x_d) \in [0, n)^d \mid x_i \in [u, u+1),\ x_j \in [v, v+1)\}$. The total number of rectangles is $O(n^2)$.

Then $(x_1, \ldots, x_d) \in [0, n)^d$ is not in the union iff $\lfloor x_i \rfloor \lfloor x_j \rfloor \in E$ for every $i, j \in \{1, \ldots, d\}$ $(i \neq j)$. Thus, the union does not cover all of $[0, n)^d$ iff the graph contains K_d. $\square$

Currently, the best time bound for the k-clique problem in terms of n for a constant $k \geq 3$ is $O(n^{\omega \lfloor k/3 \rfloor + (k \bmod 3)})$ [28] using fast matrix multiplication, where $\omega < 2.376$. Without "algebraic" techniques (à la Strassen and others), the best time bound for a purely combinatorial algorithm is still the naive $O(n^k)$ bound, ignoring logarithmic-factor-type speedups. The above reduction thus implies that solving the measure or depth problem in time $o(n^{[\omega \lfloor d/3 \rfloor + (d \bmod 3)]/2})$, or simply $o(n^{(\omega/6)d})$, would require a breakthrough on a longstanding graph problem. Without algebraic techniques, solving the measure or depth problem in time $o(n^{d/2-\varepsilon})$ would similarly require a breakthrough.

We have not ruled out the possibility that the measure problem could be solved in time $o(n^{d/2-\varepsilon})$ using fast matrix multiplication. (If it were possible, this would be quite a pleasant surprise.) Perhaps a better reduction argument involving a different graph problem could be concocted.

In any case, since the clique problem is $W[1]$-complete, there can be no polynomial algorithm for Klee's measure problem that has an exponent independent of d, according to a commonly accepted conjecture on fixed-parameter tractability. (This dashes hope for a subquadratic algorithm in all dimensions as expressed by J. Erickson [18].) We note that at least two other recent papers [6, 7] have addressed fixed-parameter intractability of geometric problems with respect to the dimension.

Final Remark: The reductions above say nothing about the special case of hypercubes or unit hypercubes.

References

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

[2] I. Baran, E. D. Demaine, and M. Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50:584–596, 2008.

[3] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th ACM Sympos. Theory Comput.*, pages 80–86, 1983.

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

[5] 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.

[6] S. Cabello, P. Giannopoulos, and C. Knauer. On the parametrized complexity of d-dimensional point set pattern matching. *Inform. Process. Lett.*, 105:73–77, 2008.

[7] S. Cabello, P. Giannopoulos, C. Knauer, D. Marx, and G. Rote. Geometric clustering: fixed-parameter tractability and lower bounds with respect to the dimension. In *Proc. 19th ACM-SIAM Sympos. Discrete Algorithms*, pages 836–843, 2008.

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

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

[10] T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *Proc. 39th ACM Sympos. Theory Comput.*, pages 590–598, 2007.

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

[12] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid. Static and dynamic algorithms for k-point clustering problems. *J. Algorithms*, 19:474–503, 1995.

[13] P. F. Dietz. Optimal algorithms for list indexing and subset rank. In *Proc. 1st Workshop Algorithms Data Struct.*, Lect. Notes Comput. Sci., vol. 382, Springer-Verlag, pages 39–46, 1989.

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

[15] 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.

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

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

[18] J. Erickson. Klee's measure problem. http://compgeom.cs.uiuc.edu/~jeffe/open/klee.html, 1998.

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

[20] M. L. Fredman and B. Weide. The complexity of computing the measure of $\cup[a_i, b_i]$. *Commun. ACM*, 21:540–544, 1978.

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

[22] H. Kaplan, N. Rubin, M. Sharir, and E. Verbin. Counting colors in boxes. In *Proc. 18th ACM-SIAM Sympos. Discrete Algorithms*, pages 785–794, 2007.

[23] 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.

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

[25] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[26] J. Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.*, 2:169–186, 1992.

[27] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom.*, 10:157–182, 1993.

[28] J. Nesetril and S. Poljak. On the complexity of the subgraph problem. *Commen. Math. Univ. Carol.*, 26:415–419, 1985.

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

[30] J. H. Reif and S. R. Tate. On dynamic algorithms for algebraic problems. *J. Algorithms*, 22:347–371, 1997.