

The Union of Probabilistic Boxes: Maintaining the Volume

Hakan Yıldız¹, Luca Foschini¹, John Hershberger², and Subhash Suri¹

¹ University of California, Santa Barbara, USA

² Mentor Graphics Corporation, Wilsonville, Oregon, USA

Abstract. Suppose we have a set of n axis-aligned rectangular boxes in d -space, $\{B_1, B_2, \dots, B_n\}$, where each box B_i is *active* (or present) with an independent probability p_i . We wish to compute the *expected* volume occupied by the union of all the active boxes. Our main result is a data structure for maintaining the expected volume over a *dynamic* family of such probabilistic boxes at an amortized cost of $O(n^{(d-1)/2} \log n)$ time per insert or delete. The core problem turns out to be one-dimensional: we present a new data structure called an *anonymous segment tree*, which allows us to compute the expected length covered by a set of probabilistic segments in logarithmic time per update. Building on this foundation, we then generalize the problem to d dimensions by combining it with the ideas of Overmars and Yap [13]. Surprisingly, while the expected value of the volume can be efficiently maintained, we show that the tail bounds, or the probability distribution, of the volume are intractable—specifically, it is *NP-hard* to compute the probability that the volume of the union exceeds a given value V *even when the dimension is $d = 1$* .

1 Introduction

We consider the problem of estimating the volume covered by a set of overlapping boxes in d -space when the existence of each box is known only with partial certainty. Specifically, we are given a set of n axis-aligned boxes, in which the i th box is known to be present only with probability p_i . (The probabilities of different boxes are independent of each other.) This is a *probabilistic* version of the classical Klee’s Measure problem [11]. Besides being a fundamental problem in its own right, it is also a natural framework to model risk and uncertainty in geometric settings. In order to motivate the problem, consider the following scenario.

Suppose that a tract of land has a variety of health hazards, occurring in possibly overlapping regions. The virulence of each hazard is expressed as a survival rate, i.e., the probability that an entity survives after being exposed to the hazard. Assuming independence of the hazards, the probability of survival at a point is the product of the survival probabilities for the different hazards affecting the point. The *average survival rate* within the whole tract is the integral of the survival probabilities of all points in the tract divided by the area the tract. It is easy to see that the integral of concern equals the expected area of

covered by the hazardous regions, if we treat the survival rates as the probability of *absence* for each region.

Let us now introduce the problem more formally. A d -dimensional rectangular box B , which we call a d -box for convenience, is the Cartesian product of d one-dimensional ranges, namely, $B = \prod_{i=1}^d [a_i, b_i]$. The *volume* of a box B is defined as $\text{vol}(B) = \prod_{i=1}^d |b_i - a_i|$. Given a set \mathcal{B} of d -boxes $\{B_1, B_2, \dots, B_n\}$, its *union* $\bigcup B_i$ is the set of points contained in at least one box of \mathcal{B} . In our problem, each box B_i is assumed to be *present* (or active) with an independent probability p_i , and absent otherwise. We wish to compute the *expected value* of the total volume occupied by such a collection of boxes. More generally, we may wish to compute the probability distribution of the volume—for each value V , the probability that the volume of the union is V . In fact, we wish to maintain a collection of such probabilistic boxes so that their volume statistics (expectation or tail bounds) are easily updated as boxes (along with their activation probabilities) are inserted or deleted.

Our problem is a probabilistic and dynamic version of *Klee’s measure problem*, which has a long history in computational geometry [3, 7, 11–13]. The fastest algorithm currently known for Klee’s problem is due to Chan [7], with worst-case time $O(n^{d/2} 2^{O(\log^* n)})$. Despite a long and distinguished history, the computational complexity of the problem has remained largely unresolved for $d \geq 3$ since the breakthrough result of Overmars and Yap [13], with time complexity $O(n^{d/2} \log n)$. Most of the work in the past several years has focused on the conceptually easier case of the *union of cubes* [1, 2] or fat boxes [5].

Our main result is a data structure for maintaining the expected volume over a dynamic set of probabilistic boxes in amortized time $O(n^{(d-1)/2} \log n)$ per insert or delete. (Any major improvement in this update complexity will imply a breakthrough on Klee’s measure problem because the d -dimensional Klee’s problem can be solved by maintaining a $(d-1)$ -dimensional volume over n insertions and deletions [7, 13].) The core problem in computing the volume of probabilistic boxes arises already in one dimension, and leads us to a new data structure called an *anonymous segment tree*. This structure allows us to compute the expected length covered by a set of probabilistic segments in logarithmic time per update. Building on this foundation, we then generalize the problem to d dimensions by combining it with the ideas of Overmars and Yap [13]. (The issues underlying this extension are mostly technical, albeit somewhat non-trivial, since the Overmars-Yap scheme uses a space-sweep that requires *a priori* knowledge of the box coordinates, while we assume a fully dynamic setting with no prior knowledge of future boxes.)

Surprisingly, while the expected value of the volume can be efficiently maintained, we show that computing the tail bounds, or the probability distribution, of the volume is intractable—specifically, it is *NP*-hard to compute the *probability* that the volume of the union exceeds a given value V even when the dimension is $d = 1$.

Finally, in order to evaluate the practical usefulness of our, and Overmars-Yap’s, scheme in geospatial databases, we implemented the scheme. The results

confirm our theoretical bounds in practice and we show that our solution easily outperforms a naïve solution.

2 Probabilistic Volume: Expectation and Tail Bounds

If algorithmic efficiency were not the main concern, then the *expected* volume of the union of n boxes would be easy to compute in polynomial time. A set of n boxes in d -space is defined by $O(dn)$ facets, and the hyperplanes determined by those facets partition the space into $O(n^d)$ rectangular cells, with each cell fully contained in all the boxes that intersect it. For each cell, compute the probability that at least one of its covering boxes is active. By the linearity of expectation, the expected volume is simply the sum of the volumes of these cells weighted by their probability of being covered. This naïve algorithm runs in $O(n^{d+1})$ time, which is polynomial in n for fixed dimension. In Section 4, we present our main result: how to maintain the expected volume much more efficiently under dynamic updates.

On the other hand, we argue below that computing the probability distribution (i.e., the tail bounds) is intractable. In particular, even in one dimension, computing the probability that the union of n probabilistic line segments has volume (length) at least V is *NP*-hard. The following theorem shows that it is hard to compute the probability that the union has length precisely L for some integer L —since we use only integer-valued lengths, computing the tail bound is at least as hard (the probability of length exactly L can be determined from those of lengths at least L and at least $L + 1$.)

Theorem 1. *Given n disjoint line segments on the integer line, where the i th segment is active with probability p_i , it is *NP*-hard to compute the probability that the union of the active segments has length L .*

Proof. We show a reduction from the well-known *NP*-complete problem SUBSET-SUM [9]. The subset-sum problem takes as input a set of positive integers $A = \{a_1, \dots, a_n\}$, a *target* integer L , and asks if there is an index subset $I \subseteq \{1, \dots, n\}$ whose elements sum to the target value L , that is, $\sum_{i \in I} a_i = L$. Given an instance of the subset-sum problem (A, L) , we create a set of probabilistic segments $\{s_1, s_2, \dots, s_n\}$, as follows. The segment s_i begins at point $\sum_{j < i} a_j$ and has length a_i . Each of the n segments occurs with probability $p_i = 1/2$. We observe that because of the uniform probability, each of the 2^n subsets of $\{s_1, s_2, \dots, s_n\}$ is equally likely, each occurring with probability 2^{-n} .

Since the segments s_1, \dots, s_n are disjoint, for any index subset $I \subseteq \{1, \dots, n\}$, the union of the segments $\{s_i | i \in I\}$ has length precisely $\sum_{i \in I} a_i$. Thus, I is a solution to the subset sum problem (A, L) if and only if the union of the segments indexed by I has length L . However, the probability that the union of the active segments has length L is precisely equal to the number of index subsets that are valid solutions of the subset sum problem divided by 2^n . Thus, given the probability that the union of the active segments is L , we can deduce whether the subset sum problem has a solution. This completes the proof. \square

3 Maintaining the Expected Measure in 1D

We begin by describing our data structure in one dimension, and then show how to embed it in an appropriately generalized version of the Overmars-Yap structure for the d -dimensional problem. We describe the data structure, called the *anonymous segment tree*, first without the probabilities, focusing on its form and updates, and then present an *abstraction* that retains all the key elements and yet accommodates probabilistic segments.

3.1 Anonymous Segment Tree

Let \mathcal{S} be a dynamic set of n line segments on the number line that undergoes insertions and deletions. Our goal is to maintain the length covered by the union of the segments in \mathcal{S} . We simply call this length the *measure* of \mathcal{S} . The segments in \mathcal{S} split the number line into at most $(2n + 1)$ disjoint intervals, called *primitive intervals*. We maintain a balanced binary tree whose keys are the coordinates of the segment endpoints, and whose leaves correspond to the primitive intervals. Each internal node represents the union of all its leaf descendants' intervals. (See Figure 1(a).)

Consider a leaf v and its associated primitive interval $I = (x_1, x_2)$. Let $\mathcal{S}_I \subset \mathcal{S}$ be the subset of segments that cover I , and define the *coverage count* of v , denoted $cover(v)$, as $|\mathcal{S}_I|$. The *measure* of v , denoted $\mu(v)$, is clearly zero if $cover(v) = 0$ and $x_2 - x_1$ otherwise. The measure of \mathcal{S} is the sum of $\mu(v)$ over all leaves v . The coverage count is an inefficient mechanism for maintaining the measure when segments are inserted or deleted, so we use a secondary quantity, called $ccover(v)$. (The name $ccover$ derives from *complete coverage count*.) The $ccover$ values satisfy the two invariants described below.

SUM INVARIANT: *For any leaf v , $cover(v)$ is the sum of $ccover(a)$ over all ancestors a of v (including v itself).*

A trivial way to achieve the invariant is to set $ccover(v) = cover(v)$ for each leaf v and $ccover(u) = 0$ for each non-leaf node u . But, as we show below, $ccover()$ allows us to support maintenance of the measure through its flexibility. We use $ccover()$ values to maintain the measure as follows, where $L(v)$ is the length of v 's interval.

$$\mu(v) = \begin{cases} L(v) & \text{if } ccover(v) > 0 \\ 0 & \text{if } ccover(v) = 0 \wedge v \text{ is a leaf} \\ \mu(v_l) + \mu(v_r) & \text{if } ccover(v) = 0 \wedge v \text{ has children } v_l, v_r \end{cases} \quad (1)$$

The following lemma is easily established.

Lemma 1. *Let a node v be called *exposed* if $ccover(a) = 0$ for all ancestors a of v (excluding v). Then for any exposed node v , $\mu(v)$ is the measure of \mathcal{S} restricted to v 's interval. In particular, $\mu(\text{root})$ is the measure of \mathcal{S} .*

PUSHUP INVARIANT: For each non-leaf node v with children v_l and v_r , at least one of $c_{cover}(v_l)$ and $c_{cover}(v_r)$ is zero.

We achieve this invariant by applying the following *push-up* operation at each internal node: Let v_l and v_r be children of v . Decrement $c_{cover}(v_l)$ and $c_{cover}(v_r)$ by $\min(c_{cover}(v_l), c_{cover}(v_r))$ and increment $c_{cover}(v)$ by the same amount. (See Figure 1(b).) This operation propagates the values of $c_{cover}()$ up the tree as much as possible, which in turn allows us to update the $\mu()$ values efficiently.¹

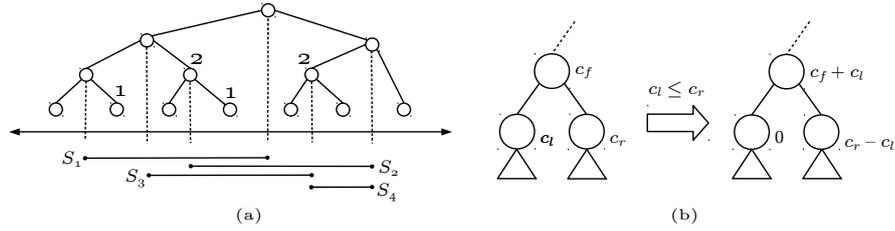


Fig. 1. (a) An anonymous segment tree, positive c_{cover} values are shown. (b) the pushup operation, c_i 's stand for c_{cover} values.

We can maintain the sum and the pushup invariants as segments are inserted or deleted by modifying $O(\log n)$ values in the tree. We briefly outline how this is achieved, omitting standard but technical details. A segment s corresponds to a set of $O(\log n)$ nodes in the tree called *canonical nodes* whose intervals are disjoint but their concatenation equals s . An insertion (deletion) is handled by incrementing (decrementing) the $c_{cover}()$ values of the canonical nodes, maintaining the sum invariant and, thus, the correct μ values in the tree. Due to the changes in primitive intervals, the tree may undergo rebalancing rotations, in which case we temporarily push the $c_{cover}()$ down below the rotating nodes to preserve the sum invariant. Afterwards, we apply the necessary push-ups to restore the pushup invariant. We note that pushing up the $c_{cover}()$ values is necessary otherwise deletions in the tree become inefficient. The push-up invariant guarantees that no $c_{cover}()$ value drops below zero after a deletion.

Lemma 2. We can maintain the measure of a dynamic set of n segments in $O(\log n)$ time for insert or delete operations, and $O(1)$ time for measure query.

We next show how to generalize the anonymous segment tree to deal with probabilistic segments. Towards that goal, we introduce an abstract framework that includes the measure of probabilistic segments as a special case.

¹ The ability to move $c_{cover}()$ values between nodes is the inspiration for the name *anonymous segment tree*: the coverage representation for a node is independent of the covering segments. Coverage of an interval by a single segment is indistinguishable from coverage by an arbitrary number of consecutive short segments.

3.2 An Abstract Anonymous Segment Tree

Let f be a function mapping the segments in \mathcal{S} to some range set G , and let \oplus be a commutative and associative binary operation on G . We consider the problem of maintaining the following sum for each primitive interval I of the set \mathcal{S} : $F(v) = \sum_{s \in \mathcal{S}_I} f(s)$, where v is the leaf associated with I and the summation uses the \oplus operation.²

We compute $F(v)$ indirectly by storing a quantity called $FF(v)$ at each node v of the tree, and maintain the invariant that the sum of $FF(a)$ over all ancestors a of v equals $F(v)$. We require that \oplus is invertible, and there is a total order \leq_G on G such that $A \leq_G B \iff A \oplus C \leq_G B \oplus C$. In other words, (G, \oplus, \leq_G) forms a *totally ordered abelian group*. Finally, we reduce the range of $f()$ from G to G^+ , defined as $G^+ = \{g \mid g \in G \wedge e \leq_G g\}$, where e is the identity element of \oplus .

The pushup invariant in this abstract setting is that, for each internal node v with children v_l and v_r , at least one of $FF(v_l)$ and $FF(v_r)$ is e and the other is in G^+ . Repeated pushup operations in the tree, starting from the leaves, establish this invariant. In particular, let v be an internal node with children v_l and v_r , and without loss of generality assume that $FF(v_l) \leq_G FF(v_r)$. The push-up operation sets $FF(v) = FF(v) \oplus FF(v_l)$, $FF(v_l) = e$ and $FF(v_r) = FF(v_r) \oplus FF(v_l)^{-1}$, where $^{-1}$ denotes the inverse with respect to \oplus . We can show that the values $FF()$ can be updated in $O(\log n)$ time as segments are inserted and deleted. (The details are technical, but have no bearing on what follows in the rest of the paper.) We now show below how to use this general framework for maintaining the measure of probabilistic segments.

3.3 Measure of Probabilistic Segments

For the sake of simplicity, we maintain the *complement* of the expected measure: the expected value of the length *not* covered by any active segment.³ In order to maintain the measure for probabilistic segments, we apply our abstract framework twice. First, for each leaf v , we maintain the number of segments that cover its interval *and* have probability 1. We denote this by $cover(v)$, and use the deterministic coverage count algorithm to maintain it. Second, we maintain the probability that the primitive interval of a leaf v is uncovered by the segments whose probability is strictly less than 1. (The segments with probability 1 are handled separately, and more easily.) We denote this quantity by $prob(v)$, and maintain it using our generalized scheme as follows. We define G as the set of positive reals, \oplus as multiplication, \leq_G as \geq , and set $f(s)$ to

$$f(s) = \begin{cases} (1 - p_s) & \text{if } p_s < 1 \\ 1 & \text{if } p_s = 1 \end{cases}$$

² Observe that if G is the set of integers, \oplus is integer addition, and $f(s) = 1$ for every s , then $F(v) = cover(v)$, as in the preceding section.

³ We assume that all segments are contained in a finite, bounded range, ensuring that the complement is bounded.

Observe that $F(v)$ represents $prob(v)$. For ease of reference we denote the $FF(v)$ values used to maintain $prob()$ by $pprob(v)$. We can define the *uncovered measure* of a node v , denoted $\nu(v)$, recursively as follows:

$$\nu(v) = \begin{cases} 0 & \text{if } ccover(v) > 0 \\ pprob(v) \cdot L(v) & \text{if } ccover(v) = 0 \wedge v \text{ is a leaf} \\ pprob(v) \cdot (\nu(v_l) + \nu(v_r)) & \text{if } ccover(v) = 0 \wedge v \text{ has children } v_l \text{ and } v_r \end{cases} \quad (2)$$

Lemma 3. *Let $\bar{\mu}(v)$ denote the complement of the expected measure of \mathcal{S} restricted to v 's interval, let $aprob(v)$ be the product of $pprob(a)$ over all ancestors a of v (excluding v), and let a node v be called *exposed* if $ccover(a) = 0$ at all strict ancestors a of v . Then, for any exposed node v , we have $\bar{\mu}(v) = aprob(v) \cdot \nu(v)$.*

Proof. If an exposed node v has $ccover(v) > 0$, then $\bar{\mu}(v) = 0$. By the first line of (2), $\bar{\mu}(v)$ equals $aprob(v) \cdot \nu(v)$. Now consider an exposed leaf v such that $ccover(v) = 0$. Then $cover(v) = 0$. We write

$$\bar{\mu}(v) = prob(v) \cdot L(v) = aprob(v) \cdot pprob(v) \cdot L(v)$$

By the second line of (2), this expression equals $aprob(v) \cdot \nu(v)$. Finally, consider an exposed internal node v such that $ccover(v) = 0$. Then v_l and v_r are exposed. By induction, $\bar{\mu}(v_l) = aprob(v_l) \cdot \nu(v_l)$ and $\bar{\mu}(v_r) = aprob(v_r) \cdot \nu(v_r)$. Then

$$\begin{aligned} \bar{\mu}(v) &= \bar{\mu}(v_l) + \bar{\mu}(v_r) = aprob(v_l) \cdot \nu(v_l) + aprob(v_r) \cdot \nu(v_r) \\ &= aprob(v) \cdot pprob(v) \cdot (\nu(v_l) + \nu(v_r)) \end{aligned}$$

By the third line of (2), the expression equals $aprob(v) \cdot \nu(v)$. □

Lemma 4. *$\nu(\text{root})$ equals the complement of the expected measure of \mathcal{S} .*

By Lemma 4, one can report the expected measure of \mathcal{S} simply by returning the complement of $\nu(\text{root})$. By maintaining $\nu()$ the same way we maintain $\mu()$ in Section 3.1, we end up with a structure that solves the stochastic measure problem:

Theorem 2. *The expected measure of a dynamic set of segments can be maintained in $O(1)$ query time, $O(\log n)$ insertion/deletion time and $O(n)$ space.*

4 Dynamic Probabilistic Volume in d Dimensions

We now show how to maintain the expected volume of the union of a dynamic set of probabilistic boxes in d -space. We adapt the framework of Overmars and Yap's solution to Klee's problem [13]. The first step is to apply Theorem 2 to a very special kind of d -dimensional box arrangement called a *trellis*. A trellis in d dimensions is a rectangular region R and a collection of boxes \mathcal{B} that such that each box in \mathcal{B} forms of an axis-parallel *strip* inside R . In other words, no $(d-2)$ -dimensional face (a corner in two dimensions) of a box in \mathcal{B} intersects the

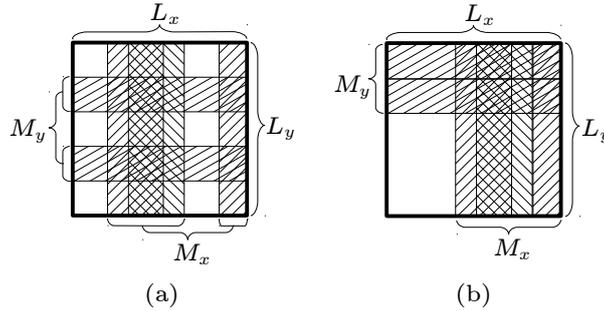


Fig. 2. (a) A two-dimensional trellis formed by 5 boxes. (b) The shape with the same area formed by moving strips.

interior of R . A two-dimensional example is shown in Figure 2(a), where each box is either a vertical or a horizontal strip.

The volume of a trellis is easy to compute efficiently. First consider the problem in two dimensions. Suppose the horizontal and vertical side lengths of R are L_x and L_y , respectively. Let M_x be the length of the portion of the x -interval of R covered by the vertical strips, and M_y the length of R 's y -interval covered by the horizontal strips. Then it is easy to see that the area covered in R is $L_x \times L_y - (L_x - M_x) \times (L_y - M_y)$. (A visual proof is offered in Figure 2(b).)

It follows that computing the area of a trellis reduces to maintaining M_x and M_y separately, i.e., to solving two *one-dimensional* volume problems. In $d > 2$ dimensions, the volume formula for a trellis generalizes easily to

$$\prod_i L_i - \prod_i (L_i - M_i),$$

where the product index ranges from 1 to d , L_i is the side length of R along the i th axis and M_i is the sublength of L_i that is covered by strips orthogonal to the i th axis [13].

To maintain the expected volume within a trellis for stochastic boxes, we use the same formula, except that all the variables in the formula are replaced by their expectations. Specifically, the formula for the d -dimensional case becomes

$$\prod_{1 \leq i \leq d} L_i - \prod_{1 \leq i \leq d} (L_i - E(M_i))$$

where $E(M_i)$ is the expected value of M_i . Note that M_i 's are independent. Then, by linearity and multiplicativity of expectation over independent variables, the formula correctly represents the expected volume.

It is clear that the expected volume in a trellis can be maintained in logarithmic time per update by using d instances of anonymous segment tree, each maintaining M_i for $1 \leq i \leq d$. While several efficient solutions are known for maintaining the one-dimensional measure of *non-probabilistic* segments [4, 8, 10],

they all seem quite specialized, focusing on a particular application. It is unclear whether they can be adapted to our probabilistic setting without fairly complicated modifications. The anonymous segment tree, on the other hand, offers a simple, and general, framework suitable for probabilistic measure maintenance.

The second step is to partition the space hierarchically such that the leaves of the partition contain trellis structures. The partition proceeds in d steps. Let us call a face of a box orthogonal to the i th axis an i -face and the hyperplane it sits on an i -bound. In the first step of the partition, we divide the space into regions called 1-slabs by cutting it with hyperplanes through every \sqrt{n} th 1-bound of the boxes along the first axis. Consequently, $O(\sqrt{n})$ 1-slabs are formed, each of which contains $O(\sqrt{n})$ 1-faces. In the second step, each 1-slab is split into 2-slabs by hyperplanes perpendicular to the second coordinate axis. These hyperplanes are introduced as follows: A hyperplane is drawn along every \sqrt{n} th 2-bound in \mathcal{B} . Additionally, for each box B that has a 1-face inside the 1-slab, two hyperplanes are drawn along both of its 2-bounds. Consequently, each 1-slab is partitioned into $O(\sqrt{n})$ 2-slabs, each of which intersects with $O(\sqrt{n})$ 1-faces and 2-faces of the boxes in \mathcal{B} . In the third step, each 2-slab is partitioned into $O(\sqrt{n})$ 3-slabs. This time, the splitting hyperplanes pass along every \sqrt{n} th 3-bound and the 3-bounds of each box that has a 1-face or a 2-face intersecting the inside of the 2-slab. This partitioning strategy is continued until the d th step, in which each $(d-1)$ -slab is divided into $O(\sqrt{n})$ cells. The following lemma, whose proof can be found in [13], summarizes the key properties of this orthogonal partition.

Lemma 5. *The orthogonal partition contains $O(n^{d/2})$ cells such that each box of \mathcal{B} partially covers $O(n^{(d-1)/2})$ cells, each cell partially intersects $O(\sqrt{n})$ boxes in \mathcal{B} , and the boxes partially overlapping a cell form a trellis.*

We can now maintain the uncovered expected volume as follows. For each cell C , we maintain uncovered volume of the boxes that partially intersects C restricted to C . By using the trellis structure we have mentioned, this is doable in logarithmic time per update and constant time per query. Since a box partially intersects $O(n^{(d-1)/2})$ cells, we can update all trellis structures in $O(n^{(d-1)/2} \log n)$ time during a box insertion/deletion.

The cells in a $(d-1)$ -slab form a linear sequence. We can track the boxes that completely overlap the cells of a $(d-1)$ -slab using a structure called a *slab tree*, which is just an anonymous segment tree with trellises at its leaves. Equation 2 applies at each node of a slab tree, except that if v is a leaf with $ccover(v) = 0$, then $\nu(v) = pprob(v) \cdot \nu_T(v)$, where $\nu_T(v)$ is the uncovered expected measure of the trellis stored at v . The uncovered measure of the whole arrangement is the sum of $\nu(r)$ over all the roots r of the $(d-1)$ -slab trees. Updating a slab tree with a box takes logarithmic time. Since there are $O(n^{(d-1)/2})$ $(d-1)$ -slabs, updating all slab trees takes $O(n^{(d-1)/2} \log n)$ during a box insertion/deletion. This yields to a total update time of $O(n^{(d-1)/2} \log n)$.

We have a final missing ingredient: a dynamic version of the hierarchical orthogonal partition. Recall that Overmars and Yap's partition is based on making slab cuts at the \sqrt{n} th coordinate in each dimension. We relax this constraint

by maintaining the sorted sequence of box coordinates in each dimension and cutting along a fairly stable—but not static—set of slab boundaries. The slab boundaries for each dimension partition the corresponding sorted coordinate sequence into *buckets*. We maintain the invariant that each bucket contains at most $2\sqrt{n}$ coordinates, and any two adjacent buckets contain at least \sqrt{n} . Whenever one of these invariants is violated, we split or merge the adjacent buckets as necessary by introducing or removing slab boundaries. This invalidates some trellises and slab trees; we restore them by rebuilding. By a potential argument, it can be shown that the amortized cost of all the rebuilding is also $O(n^{(d-1)/2} \log n)$ time per box insertion/deletion, matching the direct cost of data structure updates.

Theorem 3. *The expected volume of a dynamic set of n stochastic boxes in d -space can be maintained in $O(1)$ query time and $O(n^{(d-1)/2} \log n)$ amortized time per update, using an $O(n^{(d+1)/2})$ -space data structure.*

5 Discrete Volume

We can also solve the following *discrete* volume version of the problem. *Given a dynamic set \mathcal{P} of points, and a dynamic set \mathcal{B} of d -boxes where each point and box has a probability of being active, maintain the expected number of active points in \mathcal{P} that are contained in the union of the active boxes.* The solution idea, in brief, is to represent the points as tiny boxes of size ϵ . We use three structures to maintain the expected volume of the union of: (1) the boxes, (2) the points, and (3) both the boxes and the points. Then, the inclusion-exclusion principle can be used to compute the the expected volume of the intersection between the points and the boxes, which is equal to the expected discrete volume times ϵ . This structure achieves an update time of $O(n^{(d-1)/2} \log n)$, where n is the *total* number of points and boxes. Note that the algorithm has a few technical details. First, one needs to represent ϵ symbolically, so that the algorithm works correctly regardless of the box sizes. Second, the degenerate cases where the points lie on the box boundaries should be handled.

6 Experimental Evaluation

We implemented our algorithm in C++ (available at <http://www.cs.ucsb.edu/~foschini/dynOY/>). The main goal was to evaluate the memory usage and the update time behavior of the data structure. Since those parameters are not affected by the probabilities of the boxes, we performed all our simulations for the deterministic case, namely, $p_i = 1$.

We tested the algorithm using two-dimensional boxes with 64-bit integer coordinates. The experiments were performed on an Intel(R) Core(TM)2 Duo CPU @2.20GHz equipped with 4GB of RAM. To check the correctness of our implementation on large inputs we constructed the union of boxes explicitly using the CGAL [6] library primitives `Boolean_set_operations_2`. This construction is fast when the arrangement of boxes is small, but slows down dramatically when

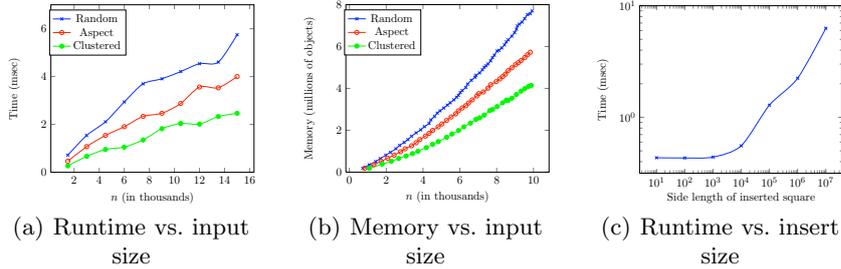


Fig. 3. Experimental Results

the arrangement size approaches the quadratic worst case. As an example of this, consider a trellis formed by 50 thin vertical and horizontal boxes. Our implementation takes 0.05 seconds to compute the measure of the union, by inserting rectangles one by one, while the CGAL program takes 8.3 seconds. This conforms with the worst case guarantees provided by our analysis.

Datasets. To test the sensitivity of our algorithm to input distributions, we used three different input configurations: (1) *Random*, a set of 15K rectangles randomly generated, with boundary coordinates uniformly distributed in $[-10^7, 10^7]$, (2) *Aspect*, a set of 15K rectangles with small (< 0.1) aspect ratio and random orientations, with boundary coordinates uniformly distributed in $[-10^7, 10^7]$, and (3) *Clustered*, a set of 15K rectangles with boundary coordinates clustered in ten groups of bounding box $[0, 2 \cdot 10^6]^2$ placed along the diagonal of the $[-10^7, 10^7]^2$ box.

Results. For each dataset, we measured the average time to insert a rectangle as a function of the number of objects n present in the data structure. (Deletion time is about 80% of insertion time.) Since the cost of a single update is predicted to be $O(\sqrt{n} \log n)$ only in an amortized sense, we report the average insertion time of the last 1500 rectangles before each tested n . Results are reported in Figures 3(a) and 3(b). The insertion time is roughly proportional to \sqrt{n} , highlighting the fact that the amortized worst case bound is a good indicator of the average case behavior. The memory required (plotted as the number of allocated objects) conforms with the predicted $O(n\sqrt{n})$ worst case bound.

In another experiment we test the sensitivity of the algorithm to the size of the box inserted. In Figure 3(c) we report the average time needed to insert a square into a set of 10K random boxes. the inserted square varying from 20 to 2×10^7 (the size of the full database). The time needed for the insertion increases by a factor of around ten over the range of insertion sizes. This is expected, since a larger square intersects more cells.

The main limitation of the data structure seems to be its memory use. The memory bound of $O(\sqrt{n})$ nodes per box limits the scalability of the algorithm. Especially, if we consider that each box has four edges each affecting $O(\sqrt{n})$ trellises and slab trees, it is easy to see how as few as 15K boxes require roughly 1.5GB of memory in our implementation.

7 Conclusions

In this paper, we considered the problem of maintaining the volume of the union of n boxes in d -space when each box is known to exist with an arbitrary, but independent, probability. We showed that, even in one dimension, computing the probability distribution, namely the probability that the volume exceeds a given value, is NP -hard. On the other hand, we showed that the expected volume of the union can be maintained, nearly as efficiently as in the static and deterministic case. Along the way we introduced a data structure called *anonymous segment tree* that may be of independent interest in dealing with dynamic segment problems with abstract measures. Finally, we also implemented our volume data structure, and showed experimentally that it performs as predicted by theory, and indeed significantly outperforms a naïve solution. Our simulation results also highlight the limitation of a Overmars-Yap type approach: the data structure is memory-intensive, which makes it unsuitable for large data sets. Thus, an interesting future research question is to explore better space-time tradeoffs that might yield scalable solutions to our dynamic and stochastic Klee’s problem.

References

1. P. K. Agarwal. An improved algorithm for computing the volume of the union of cubes. In *Proc. of 26th Symp. on Computational Geometry*, pages 230–239, 2010.
2. P. K. Agarwal, H. Kaplan, and M. Sharir. Computing the volume of the union of cubes. In *Proc. of 23rd Symp. on Computational Geometry*, pages 294–301, 2007.
3. J. L. Bentley. Solutions to Klee’s rectangle problems. *Unpublished manuscript, Dept. of Comp. Sci., CMU, Pittsburgh PA*, 1977.
4. G. van den Bergen, A. Kaldewaij, and V. J. Dielissen. Maintenance of the union of intervals on a line revisited. In *Computing Science Reports*. Eindhoven University of Technology, 1998.
5. K. Bringmann. Klee’s measure problem on fat boxes in time $O(n^{(d+2)/3})$. In *Proc. of 26th Symp. on Computational Geometry*, pages 222–229, 2010.
6. CGAL, Computational Geometry Algorithms Library. <http://www.cgal.org>.
7. T. M. Chan. A (slightly) faster algorithm for Klee’s measure problem. *Computational Geometry*, 43(3):243–250, 2010.
8. S. W. Cheng and R. Janardan. Efficient maintenance of the union intervals on a line, with applications. In *Proc. of ACM Symp. on Discrete Algorithms*, pages 74–83, 1990.
9. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. 1979.
10. G. H. Gonnet, J. I. Munro, and D. Wood. Direct dynamic structures for some line segment problems. *Computer Vision, Graphics, and Image Processing*, 23(2):178–186, 1983.
11. V. Klee. Can the measure of $\cup[a_i, b_i]$ be computed in less than $O(n \lg n)$ steps? *American Mathematical Monthly*, pages 284–285, 1977.
12. J. van Leeuwen and D. Wood. The measure problem for rectangular ranges in d -space. *Journal of Algorithms*, 2(3):282–300, 1981.
13. M. H. Overmars and C.-K. Yap. New upper bounds in Klee’s measure problem. *SIAM J. Comput.*, 20(6):1034–1045, 1991.