Dynamic and Streaming Algorithms for Union Volume Estimation

Sujoy Bhore
Indian Institute of Technology Bombay, Mumbai, India

Karl Bringmann
Saarland University and Max-Planck-Institute for Informatics, Saarbrücken, Germany

Timothy M. Chan
University of Illinois at Urbana-Champaign, IL, USA

Yanheng Wang
Saarland University and Max-Planck-Institute for Informatics, Saarbrücken, Germany

Abstract

The *union volume estimation* problem asks to $(1 \pm \varepsilon)$-approximate the volume of the union of n given objects $X_1, \ldots, X_n \subset \mathbb{R}^d$. In their seminal work in 1989, Karp, Luby, and Madras solved this problem in time $O(n/\varepsilon^2)$ in an oracle model where each object X_i can be accessed via three types of queries: obtain the volume of X_i, sample a random point from X_i, and test whether X_i contains a given point x. This running time was recently shown to be optimal [Bringmann, Larsen, Nusser, Rotenberg, and Wang, SoCG'25]. In another line of work, Meel, Vinodchandran, and Chakraborty [PODS'21] designed algorithms that read the objects in one pass using polylogarithmic time per object and polylogarithmic space; this can be phrased as a dynamic algorithm supporting insertions of objects for union volume estimation in the oracle model.

In this paper, we study algorithms for union volume estimation in the oracle model that support both *insertions* and *deletions* of objects. We obtain the following results:

1. an algorithm supporting insertions and deletions in polylogarithmic update and query time and linear space (this is the first such dynamic algorithm, even for 2D triangles);
2. an algorithm supporting insertions and suffix queries (which generalizes the sliding window setting) in polylogarithmic update and query time and space;
3. an algorithm supporting insertions and deletions of convex bodies of constant dimension in polylogarithmic update and query time and space.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Data structures design and analysis

Keywords and phrases union volume estimation, dynamic algorithms, streaming algorithms

Digital Object Identifier 10.4230/LIPIcs.SoCG.2026.12

Related Version *Full Version*: `https://arxiv.org/pdf/2602.16306` [5]

Funding *Karl Bringmann and Yanheng Wang:* This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

1 Introduction

In the *union volume estimation* problem, we are given *objects* $X_1, \ldots, X_n$, which are assumed to be measurable sets in $\mathbb{R}^d$, and the task is to $(1 \pm \varepsilon)$-approximate the volume of $\bigcup_{i=1}^n X_i$.[1] This is a fundamental problem in computer science, with connections to several areas such as

[1] A discrete variant of the problem, estimating the *cardinality* of the union of sets $X_1, \ldots, X_n \subset \mathbb{Z}^d$, can be easily reduced to the continuous version, by replacing each grid point with a unit grid cell.

network reliability [32, 34], DNF counting [34], general model counting [45, 14], probabilistic databases [35, 21, 46], and probabilistic query evaluation on databases [13, 12]. Perhaps the most popular special case in computational geometry is *Klee's measure problem* [36], where each object is a d-dimensional axis-aligned box. There has been extensive work on Klee's measure problem (either exact or approximate) in constant dimensions [44, 15, 16, 37, 27, 10].

To capture a wide variety of input objects with a unified framework, Karp and Luby [33] introduced an *oracle model*. An algorithm in this model can access each object X_i exclusively via three types of queries: (i) X_i.SIZE() gives the volume of X_i, (ii) X_i.SAMPLE() samples a point uniformly at random from the set X_i, and (iii) X_i.CONTAINS(x) tests whether a given point x is contained in X_i. The time complexity of the algorithm is measured by the number of queries plus the number of additional steps; that is, each query is assumed to take unit time. In a seminal result, Karp, Luby, and Madras [34] gave an algorithm in the oracle model that solves union volume estimation in time $O(n/\varepsilon^2)$ with constant success probability, where ε is the approximation precision. Black-box usage of this result also yields the best known algorithms for several applications, including DNF counting. (For Klee's measure problem, it implies an $O(nd/\varepsilon^2)$-time approximation algorithm [9]; in small dimensions, this was improved to $O((n + 1/\varepsilon^2)\log^{O(d)} n)$ time at last year's SoCG [10].) Only recently has a matching lower bound been established: $\Omega(n/\varepsilon^2)$ queries are necessary to solve union volume estimation in the oracle model [10].

With the static setting now fully understood, attention naturally shifts to the dynamic setting, where the input evolves over time through insertions and deletions of objects. The study of *dynamic union volume estimation* has a twofold motivation. On one hand, it captures modern data-driven applications where datasets are inherently mutable, and volume estimates must be maintained continuously rather than recomputed from scratch after each update. On the other hand, the dynamic setting presents a compelling theoretical challenge, as it is far from obvious whether the tools developed for the static setting can be dynamized efficiently. Indeed, dynamic geometric problems have repeatedly proven to be substantially more difficult than their static counterparts. An extensive body of work in recent years deals with dynamic geometric problems such as set cover [2, 17], independent set [28, 7, 6], planar point location [43], connectivity [19, 18], and nearest neighbor search [1], yet in this broader context dynamic union volume estimation remains largely unexplored. The following question has not been resolved even for very simple classes of objects such as triangles in $\mathbb{R}^2$:

Q1: Can we solve dynamic union volume estimation in the oracle model under insertions and deletions of objects in $\text{poly}\left(\log n, \frac{1}{\varepsilon}\right)$ *update and query time?*

An even more demanding paradigm is the (one-pass) streaming setting, where the algorithm has to not only handle updates, but use space that is typically required to be polylogarithmic in the input size. Earlier work has explored streaming algorithms under strong restrictions on the input objects. For example, Tirthapura and Woodruff [47] designed a dynamic streaming algorithm for Klee's measure problem on axis-aligned boxes. However, the problem remains unaddressed for more general classes of objects:

Q2: Can we solve dynamic union volume estimation in the oracle model under insertions and deletions of objects in $\text{poly}\left(\log n, \frac{1}{\varepsilon}\right)$ *space as well as update and query time?*

In the restricted *insertion-only* setting, Meel, Vinodchandran, and Chakraborty [40] gave a positive answer to this question, using $O\left(\frac{1}{\varepsilon^2}\log^2(n)(\log\log(n) + \log(\frac{1}{\varepsilon}))\right)$ time per update and query, and $O\left(\frac{1}{\varepsilon^2}\log(n)\right)$ space in the oracle model.[2] However, their algorithm cannot handle deletions.

[2] In this paper, we assume that points from the input objects and $O(\log n)$-bit integers fit into a machine

2 Our contributions

2.1 Dynamic algorithm with insertions and deletions

We answer our driving question (Q1) positively, not only for 2D triangles, 3D simplices, and other common types of objects encountered in computational geometry, but in the full generality of the oracle model. Our new dynamic algorithm can handle arbitrary sequences of both insertions and deletions.

More precisely, we consider an initially empty multiset of objects $\mathcal{X}$ under the updates INSERT(X), which adds object X to $\mathcal{X}$, and DELETE(X), which removes object X from $\mathcal{X}$. Upon query ESTIMATE(), the algorithm outputs a $(1 \pm \varepsilon)$-approximation to the volume $\mathrm{vol}\left(\bigcup_{X \in \mathcal{X}} X\right)$.

During insertion/deletion the algorithm is given oracle access to the inserted/deleted object. We also need the ability to *store* objects (or their oracles), in case of future lookup of the previously inserted objects. We make the natural assumption that (i) the argument of INSERT(X) and DELETE(X) is a *pointer* to the oracle of X, (ii) we can store a pointer in a machine word, and (iii) knowing the pointer allows us to access the oracle to which the pointer is pointing. In this interpretation, $\mathcal{X}$ is treated as a multiset of pointers: the same pointer can be inserted multiple times, and upon deletion, the multiplicity of the pointer is decreased by one. Naturally, the multiplicity of any pointer must be non-negative throughout the process (in particular, objects being deleted must have previously been inserted).

▶ **Theorem 1.** *There is a dynamic algorithm in the oracle model that maintains a multiset $\mathcal{X}$ of objects under updates* INSERT(X) *and* DELETE(X)*. Upon query* ESTIMATE()*, it outputs a $(1 \pm \varepsilon)$-approximation to* $\mathrm{vol}\left(\bigcup_{X \in \mathcal{X}} X\right)$ *with high probability. The amortized expected update time is $O(\log^5(n/\varepsilon)/\varepsilon^2)$, the query time is $O(\log^4(n)/\varepsilon^2)$, and the space complexity is $O(n + \log^4(n)/\varepsilon^2)$.*

Note that since the algorithm operates in the general oracle model, it works well even for *high-dimensional* objects (e.g., for simplices in $\mathbb{R}^d$, with polynomial dependence on d).

2.2 Streaming algorithm with insertions and suffix queries

In our remaining results, we turn to streaming algorithms and our driving question (Q2) and show that polylogarithmic time *and* space can be achieved in certain settings. We first consider the *sliding window* setting [22], which supports insertions of objects and querying over the last w objects for a fixed window size w; thus, objects are deleted in the same order as they are inserted. Numerous results and techniques have been developed in the sliding window setting (see [41, 3, 8] for an overview and [48, 30, 25] for recent work), but no such result was known for union volume estimation.

We obtain the first efficient sliding-window algorithm for union volume estimation. Again, it works in the general oracle model! Moreover, the result holds in an extension of the sliding-window setting, where at each time $t = 1, \ldots, n$ an object X_t is inserted (but cannot be deleted), and a query ESTIMATE(s) may return a $(1 \pm \varepsilon)$-approximation to the volume of the *suffix* $\mathrm{vol}(X_s \cup \cdots \cup X_t)$ with high probability for *any* number s the user specifies.

word, and space complexity counts the number of machine words. An arithmetic operation on machine words is assumed to take unit time. Both are natural assumptions for a continuous universe. Some previous work such as [40] focuses on sets from a finite universe and studies bit complexity; when stating previous work in the introduction, we have translated their results into our setting. In the finite universe Ω, different trade-offs between $\log(n)$ and $\log|\Omega|$ were obtained in [39, 42].

▶ **Theorem 2.** *There is an algorithm that implicitly maintains a sequence of objects $X_1, \ldots, X_t$ of aspect ratio M. The algorithm supports update* INSERT(X)*, which appends an object X (as X_{t+1}) to the sequence, and query* ESTIMATE(s)*, which outputs a $(1 \pm \varepsilon)$-approximation to $\mathrm{vol}(X_s \cup \cdots \cup X_t)$ with high probability. The update time is $O(\log^2(n/\varepsilon)/\varepsilon^2)$, the query time is $O(\log(n)/\varepsilon^2)$, and the space complexity is $O(\log(M)\log(n)/\varepsilon^2)$.*

Our algorithm not only generalizes previous insertion-only results [40], but is also remarkably simple. Note that the space bound depends on the aspect ratio M, defined as the ratio of the largest volume of any object ever inserted and the smallest volume of any object ever inserted. In many applications, M is a polynomial in n and thus $\log(M) = O(\log n)$. Furthermore, geometric algorithms in the sliding window setting [24, 20] usually have logarithmic dependencies on parameters such as the aspect ratio or spread.

2.3 Streaming algorithm for convex bodies with insertions and deletions

Lastly, we consider streaming algorithms that support *both* insertions and deletions. We manage to develop an efficient algorithm for a reasonably wide class of geometric objects – namely, convex bodies in any constant dimension:

▶ **Theorem 3.** *Given $R \geq r > 0$, there is an algorithm that implicitly maintains a multiset $\mathcal{X}$ of convex bodies in $[0, R]^d$, where each convex body contains a ball of radius r, under updates* INSERT(X) *and* DELETE(X)*. Upon query* ESTIMATE$()$*, it outputs a $(1 \pm \varepsilon)$-approximation to $\mathrm{vol}\left(\bigcup_{X \in \mathcal{X}} X\right)$ with high probability. The expected update and query time and space are $O(\mathrm{polylog}(\frac{nR}{\varepsilon r})/\varepsilon^2)$ for constant d.*

The bounds depend polylogarithmically on R/r, which is common in volume estimation algorithms for a single convex object [23] as well as geometric streaming algorithms [29, 26].

Previously, Tirthapura and Woodruff obtained an efficient dynamic streaming algorithm only for axis-aligned boxes [47] (with $\mathrm{poly}(\log n, \log \Delta, \frac{1}{\varepsilon}, d)$ update time and space in a discrete universe $\{1, \ldots, \Delta\}^d$). To emphasize the generality of our algorithm, note that many non-convex objects can be decomposed into a small number of convex objects, e.g., non-convex polyhedra of constant combinatorial complexity in constant dimensions can be decomposed into a constant number of simplices, to which our algorithm can be applied.

3 Technical overview

3.1 Dynamic algorithm with insertions and deletions

Karp, Luby and Madras' algorithm [34] and subsequent work are based on the observation that we can estimate the volume of a union U by maintaining a finite random subset $S^{(\ell)} \subset U$ that selects every point independently with probability density $1/2^\ell$. It turns out that the "right" choice of ℓ is one such that $|S^{(\ell)}| \approx \log(n)/\varepsilon^2$; in this case $2^\ell |S^{(\ell)}|$ provides a good estimate of the volume of U.

Meel, Vinodchandran, and Chakraborty [40] described a particularly simple way to maintain such a random set $S^{(\ell)}$ for insertion-only updates: when a new object X is inserted, we remove from $S^{(\ell)}$ all points that are contained in X, then add to $S^{(\ell)}$ freshly sampled points in X with density $1/2^\ell$. This step is inexpensive for the right choice of ℓ since $|S^{(\ell)}|$ is logarithmic. If $|S^{(\ell)}|$ becomes too big, we increment ℓ (thereby halving the sampling rate and also $|S^{(\ell)}|$ in expectation) and move on.

Difficulty arises in the presence of deletions: the volume is no longer monotonically increasing, and the right choice of ℓ may increase or decrease over time. Our first observation is that the *deletion-only* case can be handled by a variant of Meel, Vinodchandran, and

Chakraborty's insertion-only algorithm. Roughly, we can maintain $S^{(\ell)}$ for the right ℓ by keeping a counter on the number of objects containing each sample point (recall that $|S^{(\ell)}|$ is logarithmic for the right ℓ). When an object X is deleted, we decrement the counters of all points in $S^{(\ell)} \cap X$, and when a counter drops to 0, the point is removed from $S^{(\ell)}$. When the number of surviving sample points drops below logarithmic, we decrease ℓ, rebuild $S^{(\ell)}$ by iterating over all objects that has not been deleted (recall that we can store pointers to all objects), and reinitialize the counters.

A popular technique in dynamic computational geometry by Bentley and Saxe [4], called the *logarithmic method*, transforms deletion-only data structures into fully dynamic ones. It works by decomposing a fully dynamic set into logarithmically many deletion-only subsets, increasing the update and query time by typically just a logarithmic factor. The logarithmic method applies to so-called *decomposable* problems that satisfy the following property: if the input is decomposed into subsets, then the answer to the original problem can be quickly obtained from the answers to the individual subsets. Unfortunately, union volume estimation is *not* a decomposable problem: We cannot determine the volume of the union of $\mathcal{X}_0 \cup \cdots \cup \mathcal{X}_{\log n}$ just from the volume of the union of each $\mathcal{X}_i$.

Nevertheless, we show that a novel variant of the logarithmic method can still work. Recall that the deletion-only data structure already maintains a sample for each $\mathcal{X}_i$. For each sample point we keep track of more counters – specifically, how many sets from the other $\mathcal{X}_i$'s contain the point. These counters provide enough information to avoid double counting and help us estimate the union volume accurately. One issue though is that many counters need to be adjusted when one of the deletion-only data structures changes ℓ and recomputes $S^{(\ell)}$. Fortunately, we manage to bound the total number of such changes, so the amortized cost for maintaining all counters remains polylogarithmic. See Section 5 for details.

3.2 Streaming algorithm with insertions and suffix queries

The dynamic algorithm with arbitrary insertions and deletions needs linear space because the data structure rebuilds on the remaining objects when the "right" ℓ decreases. However, in the sliding window setting and more generally the setting of suffix queries, the remaining objects are contiguous in time and need not be stored explicitly. We obtain a streaming algorithm in this case by a simple yet non-trivial generalization of Meel, Vinodchandran, and Chakraborty's algorithm. Specifically, we keep the timestamp when each point in $S^{(\ell)}$ is sampled. For each ℓ we keep track of the largest timestamp $s^{(\ell)}$ such that discarding points older than $s^{(\ell)}$ from $S^{(\ell)}$ makes $|S^{(\ell)}|$ logarithmically bounded. This timestamp $s^{(\ell)}$ is monotonically increasing over time for each fixed ℓ. To estimate the volume of the union of $X_s \cup \cdots \cup X_t$ for a given s at time t, we identify the smallest ℓ for which $s^{(\ell)} \le s$, and output $2^\ell |S^{(\ell)} \cap (X_s \cup \ldots \cup X_t)|$. See Section 6 in the full version [5] for details.

3.3 Streaming algorithm for convex bodies with insertions and deletions

Our streaming algorithm for low-dimensional convex objects for arbitrary insertions and deletions is based on another simple sampling strategy. First, it is not difficult to discretize the space $\mathbb{R}^d$ to a grid $[\Delta]^d$ for some large Δ, so that the volume of the union U is close to the number of grid points in $U \cap [\Delta]^d$. The idea now is to select a random subset $R^{(\ell)}$ of the grid $[\Delta]^d$ in the beginning, and maintain the intersection $S^{(\ell)} := U \cap R^{(\ell)}$ during the updates. Using a pairwise independent family of hash functions, $R^{(\ell)}$ can be represented implicitly with logarithmically many bits.

When we insert or delete an object X, we just need to insert or delete $X \cap R^{(\ell)}$ to or from $S^{(\ell)}$ (viewed as a multiset of points). An issue is that the space usage may be large for a $S^{(\ell)}$ with a "wrong" sampling rate, and we do not know which ℓ is the right choice beforehand. Furthermore, a wrong choice may become right at a later time. This issue can be resolved by representing $S^{(\ell)}$ by data structures for sparse vectors [26, 31], which support adding numbers to a given coordinate, as well as recovering the vector when its norm is below a threshold. In this way, we essentially just pay for the space at the right sampling rate.

A question remains: how to enumerate the points of $X \cap R^{(\ell)}$ for a given object X and for our implicitly represented set $R^{(\ell)}$? For the case where the objects are rotated (non-axis-aligned) boxes in constant dimension, and for a particular pairwise independent family of hash functions, we observe that this subproblem directly reduces to integer linear programming (ILP) with a constant number of constraints and variables, and therefore can be solved efficiently by well-known algorithms [38]. For more general convex objects in constant dimension, we bound each object X by a small rotated box $B \supseteq X$, use an ILP to enumerate the points of $B \cap R^{(\ell)}$ and discard those not in X. By "small" we mean that the volume of B is comparable to that of X, so in particular the number of discarded points is small. See Section 7 in the full version [5] for details.

Our approach is related to Tirthapura and Woodruff's algorithm [47] for axis-aligned boxes. Their idea was to directly adapt a streaming algorithm for approximately counting distinct elements to estimate $|U \cap [\Delta]^d|$: to insert or delete an object X, we cannot afford to literally insert or delete all points in $|X \cap [\Delta]^d|$, but they showed how to efficiently simulate the effect for a specific distinct elements algorithm in the case of axis-aligned boxes. Our approach is more modular in that we apply sparse recovery only as a black box. Our main contribution however is the realization, via solving an ILP, that this type of approach works far more generally than for axis-aligned boxes, namely, for general convex objects.

4 Preliminaries

We assume that all relevant parameters (the precision ε, an upper bound n on the number of operations, etc.) are given at initialization. Knowing n is in line with the usual assumptions in streaming algorithms; it can easily be avoided but this complicates notation. The phrase "with high probability" means that the probability is at least $1 - n^{-\Omega(1)}$. We denote $[n] := \{1, 2 \ldots, n\}$. The notation $a \in b \pm c$ means that $b - c \le a \le b + c$. Similarly, $a \in (1 \pm \varepsilon)b$ means that $(1 - \varepsilon)b \le a \le (1 + \varepsilon)b$, and we say that a is a $(1 \pm \varepsilon)$-approximation to b.

As in all previous work, our algorithms exploit the close relation between volume estimation and sampling. The notion of ℓ-samples is central:

▶ **Definition 4** (ℓ-sample). *Let U be a measurable set and $\ell \in \mathbb{Z}$. A random finite subset $S \subset U$ is an* ℓ-sample *of U, denoted $S \sim \mathrm{Pois}(U, \ell)$, if $|S \cap V| \sim \mathrm{Pois}(\mathrm{vol}(V)/2^\ell)$ for every measurable subset $V \subseteq U$.*

It follows from this definition that: (i) if $S \sim \mathrm{Pois}(U, \ell)$, then $S \cap V \sim \mathrm{Pois}(V, \ell)$ for any measurable subset $V \subseteq U$; (ii) if $S \sim \mathrm{Pois}(U, \ell)$ and $T \sim \mathrm{Pois}(V, \ell)$ are independent and U, V are disjoint, then $S \cup T \sim \mathrm{Pois}(U \cup V, \ell)$.

▶ **Definition 5** (level). *For a measurable set U, we define* $\mathrm{level}(U)$ *as the unique integer such that* $\frac{32 \log n}{\varepsilon^2} \le \frac{\mathrm{vol}(U)}{2^{\mathrm{level}(U)}} < \frac{64 \log n}{\varepsilon^2}$.

We will see later that if $S \sim \mathrm{Pois}(U, \ell)$ where $\ell \approx \mathrm{level}(U)$, then $|S|$ concentrates around its expectation $\mathrm{vol}(U)/2^\ell$.

Let U denote the union we are interested in. Each of our algorithms morally maintains an ℓ-sample $S^{(\ell)}$ of U for every $\ell \in \mathbb{Z}$ throughout the updates; upon query, it identifies some $L \approx \text{level}(U)$ and outputs $2^L|S^{(L)}|$. However, there are two pitfalls. First, the identification of L actually requires inspecting the outcomes of the random sets $S^{(\ell)}$, so L is a random variable and this complicates the concentration analysis. Second, we cannot afford to maintain all sets $S^{(\ell)}$ at once. Our algorithms only explicitly store those $S^{(\ell)}$ with $\ell \approx \text{level}(U)$; for such ℓ we have $\mathbb{E}|S^{(\ell)}| \approx \text{vol}(U)/2^{\text{level}(U)} = \Theta(\log(n)/\varepsilon^2)$. For $\ell \ll \text{level}(U)$ the set $S^{(\ell)}$ is too large and must be compressed or discarded. Naturally, this further complicates the algorithms.

5 Dynamic algorithm with insertions and deletions

In this section, we prove Theorem 1 by presenting a fully dynamic (non-streaming) algorithm for union volume estimation in linear space and polylogarithmic update time. Throughout we assume that the volume of each object is in the range $[(3n/\varepsilon)^2, (3n/\varepsilon)^4]$. The general case can be easily reduced to this case; see Lemma 4.1 in the full version [5].

5.1 Preparations

We will use the Karp–Luby–Madras algorithm as a subroutine:

▶ **Theorem 6** (Karp, Luby, Madras [34]). *Let $n \in \mathbb{N}$ be a parameter. There is an algorithm $\text{KLM}(\mathcal{X})$ that computes a (1 ± 0.5)-approximation to $\text{vol}(\bigcup_{X\in\mathcal{X}} X)$ with probability at least $1 - n^{-3}$ and runs in time $O(|\mathcal{X}| \log n)$.*

We need a somewhat technical notion that is "close" to an ℓ-sample at the right level ℓ.

▶ **Definition 7.** *Let U be a measurable set. A family of pairs $\{(\ell_1, R_1), \dots, (\ell_N, R_N)\}$ is called an* ensemble *of U if $\ell_k \in \text{level}(U) \pm 1$ and $R_k \sim \text{Pois}(U, \ell_k)$ for all $k \in [N]$. A random pair (L, S) is called a* (δ, N)-digest *of U if there exists an ensemble $\mathcal{E}$ of U such that $|\mathcal{E}| \le N$ and $\mathbb{P}[(L, S) \notin \mathcal{E}] \le \delta$. The ensemble $\mathcal{E}$ is called a* support ensemble *of the digest (L, S).*

▶ **Lemma 8.** *Fix measurable sets $V \subseteq U$. If $\mathcal{E}$ is an ensemble of U, then with probability at least $1 - \frac{2|\mathcal{E}|}{n^4}$, we have $\left|2^\ell\,|R \cap V| - \text{vol}(V)\right| \le \varepsilon\,\text{vol}(U)$ for all $(\ell, R) \in \mathcal{E}$.*

Proof. Consider an arbitrary $(\ell, R) \in \mathcal{E}$. By definition, $\ell \le \text{level}(U) + 1$ and $R \sim \text{Pois}(U, \ell)$. In particular, $|R \cap V| \sim \text{Pois}(\lambda)$ where $\lambda := \text{vol}(V)/2^\ell$. Let us write $d := \varepsilon\,\text{vol}(U)/2^\ell \ge \varepsilon\lambda$. By concentration of Poisson variables [11],

$$\begin{aligned}
&\mathbb{P}\left(\left|2^\ell\,|R \cap V| - \text{vol}(V)\right| > \varepsilon\,\text{vol}(U)\right) \;=\; \mathbb{P}(||R \cap V| - \lambda| > d) \\
&\le 2\exp\left(-\frac{d^2}{2(\lambda + d)}\right) \le 2\exp\left(-\frac{\varepsilon d}{2(1+\varepsilon)}\right) = 2\exp\left(-\frac{\varepsilon^2\,\text{vol}(U)}{2(1+\varepsilon)\cdot 2^\ell}\right) \le 2\mathrm{e}^{-4\log n} \le \frac{2}{n^4}
\end{aligned}$$

where the second last step uses $\frac{\text{vol}(U)}{2^\ell} \ge \frac{\text{vol}(U)}{2^{\text{level}(U)+1}} \ge \frac{16\log n}{\varepsilon^2}$. Taking a union bound over all $(\ell, R) \in \mathcal{E}$ proves the lemma. ◀

▶ **Corollary 9.** *Fix measurable sets $V \subseteq U$. If S is a (δ, N)-digest of U, then*

$$\mathbb{P}\left(\left|2^L\,|S \cap V| - \text{vol}(V)\right| \le \varepsilon\,\text{vol}(U)\right) \ge 1 - \delta - \frac{2N}{n^4}.$$

Proof. Let $\mathcal{E}$ be a support ensemble of (L, S), so $|\mathcal{E}| \le N$ and $\mathbb{P}[(L, S) \notin \mathcal{E}] \le \delta$. The event in the statement is implied by $(L, S) \in \mathcal{E}$ and the event in Lemma 8. ◀

▶ **Lemma 10.** *Let $n \in \mathbb{N}$ and $\varepsilon \in (0, \frac{1}{4}]$ be parameters. There is an algorithm* ERASE-RESAMPLE$(\mathcal{X}, L)$ *that, given a multiset $\mathcal{X}$ of $m \le n$ objects and a random variable L, returns a set S of size $|S| = O(\log(n)/\varepsilon^2)$ such that (L, S) is a $(\delta, 3)$-digest of $U := \bigcup_{X \in \mathcal{X}} X$ for $\delta := \frac{2}{n^3} + \mathbb{P}[L \notin \text{level}(U) \pm 1]$. The algorithm runs in time $O(m \log(n)/\varepsilon^2)$.*

Algorithm 1 Digesting a decremental union.

```
function ERASE-RESAMPLE(𝒳, L)
    S := ∅
    foreach X ∈ 𝒳 do
        foreach x ∈ S do
            if X.CONTAINS(x) then
                remove x from S
        sample a random number N ~ Pois(X.SIZE/2^L)
        if |S| + N > 160 log(n)/ε^2 then
            return ∅
        for k = 1, ..., N do
            x := X.SAMPLE()
            add x to S
    return S
```

Proof. The algorithm is described as Algorithm 1. The bound on the size $|S|$ is clear. The running time analysis is also easy: there are m iterations, each running in time $O(\log(n)/\varepsilon^2)$ due to the bound on $|S|$ (and on $|S| + N$).

Next, we show the correctness. Assume first that the algorithm does not return from the if-clause. Let $X_1, \dots, X_m$ be the objects in $\mathcal{X}$ in the order enumerated by the foreach-loop, and write $U := \bigcup_{i=1}^m X_i$. We define the random set $X_i^{(L)}$ as the set of points sampled from X_i in the for-k-loop; observe that $X_i^{(L)} \sim \text{Pois}(X_i, L)$ for every $i \in [n]$. Let S_i be the set S at the end of the iteration for $X = X_i$. Note that with $S_0 := \emptyset$ the algorithm computes $S_i = (S_{i-1} \setminus X_i) \cup X_i^{(L)}$ in each iteration. It follows that $S_i = \bigcup_{j=1}^i X_j^{(L)} \setminus (X_{j+1} \cup \dots \cup X_i)$. Since the random sets $X_j^{(L)} \setminus (X_{j+1} \cup \dots \cup X_i) \sim \text{Pois}(X_j \setminus (X_{j+1} \cup \dots \cup X_i), L)$ are L-samples with disjoint supports and the union of supports is $\bigcup_{j=1}^i X_j$, we have $S_i \sim \text{Pois}(\bigcup_{j=1}^i X_j, L)$.

Now let us analyze the probability of the algorithm aborting. Observe that the algorithm aborts if and only if $|S_i| > 160 \log(n)/\varepsilon^2$ for some $i \in [n]$. By the monotonicity of the size of the support $\bigcup_{j=1}^i X_j$, we have $\mathbb{P}[|S_1| > 160 \log(n)/\varepsilon^2] \le \dots \le \mathbb{P}[|S_n| > 160 \log(n)/\varepsilon^2]$. Fix $L \in \text{level}(U) \pm 1$. Since $(1+\varepsilon)\frac{\text{vol}(U)}{2^L} \le (1+\varepsilon)\frac{\text{vol}(U)}{2^{\text{level}(U)-1}} \le (1+\varepsilon)\frac{128 \log n}{\varepsilon^2} \le \frac{160 \log n}{\varepsilon^2}$, we have

$$\mathbb{P}\left(|S_n| > \frac{160 \log n}{\varepsilon^2}\right) \le \mathbb{P}\left(|S_n| > (1+\varepsilon)\frac{\text{vol}(U)}{2^L}\right) \le \frac{2}{n^4},$$

where the second inequality follows from Lemma 8 by interpreting (L, S_n) as a size-1 ensemble of U and plugging in $V := U$. By a union bound over all events $|S_i| > 160 \log(n)/\varepsilon^2$, it follows that the algorithm aborts with probability at most $\frac{2}{n^3}$, if $L \in \text{level}(U) \pm 1$.

If the algorithm does not abort, then we have shown above that it returns a set $S = S_n \sim \text{Pois}(\bigcup_{j=1}^n X_j, L) = \text{Pois}(U, L)$. It follows that we can compare the distribution of (L, S) with the size-3 ensemble $\mathcal{E} := \{(\ell, \text{Pois}(U, \ell)) \colon \ell \in \text{level}(U) \pm 1\}$. We can bound the error probability $\mathbb{P}[(L, S) \notin \mathcal{E}]$ by the probability that $L \notin \text{level}(U) \pm 1$ plus the probability that the algorithm aborts assuming $L \in \text{level}(U) \pm 1$, and we have bounded the latter probability by $\frac{2}{n^3}$. This proves the claim. ◀

5.2 Digesting decremental union

Next we study the decremental setting. That is, we receive a set of objects $\mathcal{X}$ and want to build a data structure that allows efficient deletions. At any time, it needs to quickly report a digest of the union of the surviving objects (i.e., objects that have not been deleted).

Algorithm 2 Digest data structure for decremental union.

parameters: n, ε
public variables: $\mathcal{X}, S, L$
function INITIALIZE($\mathcal{Y}$)
  let $\mathcal{X} := \mathcal{Y}$ and $L := \infty$
  REFRESH()
function REFRESH
  $v := \mathrm{KLM}(\mathcal{X})$
  let L' be the unique integer such that $\frac{32 \log n}{\varepsilon^2} \le v/2^{L'} < \frac{64 \log n}{\varepsilon^2}$
  $L := \min\{L - 1, L'\}$
  $S :=$ ERASE-RESAMPLE($\mathcal{X}, L$)
  foreach $x \in S$ **do**
    compute and store $m_x := \#\{X \in \mathcal{X} : X.\text{CONTAINS}(x)\}$
function DELETE(X)
  $\mathcal{X} := \mathcal{X} \setminus X$
  foreach $x \in S$ **do**
    if $X.\text{CONTAINS}(x)$ **then**
      $m_x := m_x - 1$
    if $m_x = 0$ **then**
      $S := S \setminus \{x\}$ and forget m_x
  if $|S| < \frac{24 \log n}{\varepsilon^2}$ **then**
    REFRESH()
    return true
  else
    return false

▶ **Lemma 11.** *Let $n \in \mathbb{N}$ and $\varepsilon \in (0, \frac{1}{16}]$ be parameters. Algorithm 2 supports* INITIALIZE($\mathcal{X}$)*, which initializes with a multiset $\mathcal{X}$ of $m \le n$ objects whose volumes are in $[(3n/\varepsilon)^2, (3n/\varepsilon)^4]$, and* DELETE($X$)*, which removes object X from $\mathcal{X}$. The algorithm maintains a pair (L, S) which is a $(\frac{18}{n^2}, 6n)$-digest of $\bigcup_{X \in \mathcal{X}} X$ at any point in time, and $|S| = O(\log(n)/\varepsilon^2)$. In total,* INITIALIZE *and any sequence of* DELETE *operations run in expected time $O(m \log^2(n/\varepsilon)/\varepsilon^2)$. The space complexity is $O(m + \log(n)/\varepsilon^2)$.*

Moreover, the set S only rarely gains new points: DELETE(X) *returns true if the set S after the deletion is not a subset of the set S before the deletion (and false otherwise); and over any sequence of deletions only $O(\log(n/\varepsilon))$ deletions return true, with probability $1 - O(n^{-2})$.*

Proof. Fix the initial objects and an arbitrary sequence of m deletions. After $t = 0, 1, \ldots, m$ deletions, let U_t be the union of the surviving objects, and let L_t, S_t be the variables L, S in the algorithm. We claim that (L_t, S_t) is a $(\frac{9(t+1)}{n^3}, 3(t+1))$-digest of U_t.

Let us prove the claim by induction. First consider initialization ($t = 0$). By Theorem 6, with probability at least $1 - n^{-3}$ KLM computes $v \in (1 \pm 0.5)\,\mathrm{vol}(U_0)$, and in this case the algorithm computes $L \in \mathrm{level}(U_0) \pm 1$. Thus, ERASE-RESAMPLE returns a $(\frac{3}{n^3}, 3)$-digest (L_0, S_0) of U_0 by Lemma 10.

Now we proceed to $t \geq 1$. By the induction hypothesis, (L_{t-1}, S_{t-1}) is a $(\frac{9t}{n^3}, 3t)$-digest of U_{t-1}. Let $\mathcal{E}$ be a support ensemble of (L_{t-1}, S_{t-1}). Define

$$\mathcal{E}' := \{(\ell, R \cap U_t) : (\ell, R) \in \mathcal{E},\ \ell \leq \text{level}(U_t) + 1\}.$$

Observe that $|\mathcal{E}'| \leq |\mathcal{E}| \leq 3t$ and $\mathcal{E}'$ is an ensemble of U_t since $\text{level}(U_t) \leq \text{level}(U_{t+1})$.

Let E be the event that $(L_{t-1}, S_{t-1}) \in \mathcal{E}$. Let F be the event that $\left|2^\ell |R \cap U_t| - \text{vol}(U_t)\right| \leq \varepsilon\,\text{vol}(U_{t-1})$ for all $(\ell, R) \in \mathcal{E}$. Note that $\mathbb{P}(\overline{E}) \leq \frac{9t}{n^3}$ by the induction hypothesis, and $\mathbb{P}(\overline{F}) \leq \frac{2 \cdot 3t}{n^4} < \frac{6}{n^3}$ by Lemma 8.

Assuming $E \cap F$, we have $L_{t-1} \in \text{level}(U_{t-1}) \pm 1$ and $2^{L_{t-1}} |S_{t-1} \cap U_t| \in \text{vol}(U_t) \pm \varepsilon\,\text{vol}(U_{t-1})$. Let us distinguish three cases and derive further consequences:

(1) If $L_{t-1} \leq \text{level}(U_t)$, then

$$|S_{t-1} \cap U_t| \geq \frac{\text{vol}(U_t)}{2^{\text{level}(U_t)}} - \varepsilon \frac{\text{vol}(U_{t-1})}{2^{\text{level}(U_{t-1})-1}} \geq \frac{32 \log n}{\varepsilon^2} - \varepsilon \frac{128 \log n}{\varepsilon^2} \geq \frac{24 \log n}{\varepsilon^2}.$$

So the algorithm does not refresh. As a result, $L_t = L_{t-1} \leq \text{level}(U_t)$ and $S_t = S_{t-1} \cap U_t$, thus $(L_t, S_t) \in \mathcal{E}'$.

(2) If $L_{t-1} \geq \text{level}(U_t) + 2$, then

$$|S_{t-1} \cap U_t| \leq \frac{\text{vol}(U_t)}{2^{\text{level}(U_t)+2}} + \varepsilon \frac{\text{vol}(U_{t-1})}{2^{\text{level}(U_{t-1})-1}} < \frac{16 \log n}{\varepsilon^2} + \varepsilon \frac{128 \log n}{\varepsilon^2} \leq \frac{24 \log n}{\varepsilon^2}.$$

So the algorithm refreshes. During refresh, the algorithm computes $L' \in \text{level}(U_t) \pm 1$ with probability at least $1 - \frac{1}{n^3}$ by Theorem 6, and then we have $L_t = \min\{L_{t-1} - 1, L'\} = L' \in \text{level}(U_t) \pm 1$. Hence, it returns a $(\frac{3}{n^3}, 3)$-digest (L_t, S_t) of U_t by Lemma 10.

(3) Otherwise $L_{t-1} = \text{level}(U_t)+1$, and the algorithm may or may not refresh. If it does, then (L_t, S_t) is a $(\frac{3}{n^3}, 3)$-digest of U_t as in case (2). If it does not, then $L_t = L_{t-1} = \text{level}(U_t)+1$ and $S_t = S_{t-1} \cap U_t$, thus $(L_t, S_t) \in \mathcal{E}'$.

Taking all cases into account, recalling $\mathbb{P}(\overline{E \cap F}) \leq \mathbb{P}(\overline{E}) + \mathbb{P}(\overline{F}) \leq \frac{9t+6}{n^3}$, and adding the error probability $\frac{3}{n^3}$ of the new digest after refreshing, we conclude that (L_t, S_t) is a $(\frac{9(t+1)}{n^3}, 3(t+1))$-digest of U_t. (In particular, it is a $(\frac{18}{n^2}, 6n)$-digest of the union of the remaining objects.)

As a consequence, with probability at least $1 - O(n^{-2})$ we have $L_0 \in \text{level}(U_0) \pm 1$ and $L_{m-1} \in \text{level}(U_{m-1}) \pm 1$. On the other hand, $\text{level}(U_0) \leq O(\log(n/\varepsilon))$ and $\text{level}(U_{m-1}) \geq 0$ (because each object has volume in $[(3n/\varepsilon)^2, (3n/\varepsilon)^4]$), and every refresh causes L to decrease by at least one. So the number of calls to REFRESH is bounded by $O(\log(n/\varepsilon))$. Since we have $S_t \subseteq S_{t-1}$ except when we do a refresh, it follows that the set S can only gain points during $O(\log(n/\varepsilon))$ many deletions, with probability $1 - O(n^{-2})$.

The size bound $|S| = O(\log(n)/\varepsilon^2)$ follows from Lemma 10.

It remains to bound the running time and space usage. For INITIALIZE and REFRESH, the subroutine KLM runs in $O(m \log n)$ time by Theorem 6; the subroutine ERASE-RESAMPLE runs in $O(m \log(n)/\varepsilon^2)$ time by Lemma 10; and the loop runs in $O(m \log(n)/\varepsilon^2)$ time since $|S| = O(\log(n)/\varepsilon^2)$ and each point in S is tested against at most m objects.

For DELETE, we spend $O(\log(n)/\varepsilon^2)$ time in the loop. An extra $O(m \log(n)/\varepsilon^2)$ time may be incurred by a refresh. As we argued before, there are at most $O(\log(n/\varepsilon))$ refreshes throughout the deletion sequence with probability $1 - O(n^{-2})$. There can be at most m refreshes with the remaining probability $O(n^{-2})$. So the expected time for the whole deletion sequence is $O(m \log(n)/\varepsilon^2 \cdot (\log(n/\varepsilon) + m/n^2)) \leq O(m \log^2(n/\varepsilon)/\varepsilon^2)$.

Finally, let us analyze the space complexity. Storing $\mathcal{X}$ in a binary search tree needs $O(m)$ space. Storing S and the counters m_x in a table needs $O(\log(n)/\varepsilon^2)$ space. Storing L needs constant space. So the algorithm uses $O(m + \log(n)/\varepsilon^2)$ space in total. ◀

5.3 Handling insertions

In this section, we apply the decremental data structure from the previous section to derive a dynamic algorithm for union volume estimation, using a novel variant of the logarithmic method [4]. Throughout the algorithm, the remaining objects are partitioned into bins $\mathcal{X}_0, \dots, \mathcal{X}_{\log n}$. Each bin $\mathcal{X}_j$ holds at most 2^j objects and is implemented by the decremental data structure. Every now and then, we merge some (small) bins into a larger bin by re-initializing the target bin with the objects in the source bins. Between any two merges that involve a bin $\mathcal{X}_j$, it undergoes deletions only. At any time, we can access a digest $(\mathcal{X}_j.L, \mathcal{X}_j.S)$ of the union $\bigcup_{X \in \mathcal{X}_j} X$. Additionally, for every point $x \in \mathcal{X}_j.S$, we keep track of the number $n_x := \#\{X \in \mathcal{X}_0 \cup \dots \cup \mathcal{X}_{j-1} : x \in X\}$.

▶ **Theorem 12.** *Let $n \in \mathbb{N}$ and $\hat{\varepsilon} \in (0, \frac{1}{16}]$ be parameters. Algorithm 3 maintains an initially empty multiset $\mathcal{X}$ of objects, where each object has volume in $[(3n/\hat{\varepsilon})^2, (3n/\hat{\varepsilon})^4]$, under updates* INSERT$(X)$ *and* DELETE(X) *and query* ESTIMATE$()$*, which $(1 \pm \hat{\varepsilon})$-approximates the volume of the union of $\mathcal{X}$ and is correct with high probability. The amortized expected update time is $O(\log^5(n/\hat{\varepsilon})/\hat{\varepsilon}^2)$, the query time is $O(\log^4(n)/\hat{\varepsilon}^2)$, and the space complexity is $O(n + \log^4(n)/\hat{\varepsilon}^2)$.*

Algorithm 3 Dynamic union volume estimation with arbitrary insertions and deletions.

parameters: $n, \hat{\varepsilon}$
function INITIALIZE
  for $j = 0, \dots, \log n$ **do**
    $\mathcal{X}_j$.INITIALIZE$(\emptyset)$ with parameters n and $\varepsilon := \hat{\varepsilon}/(1 + \log n)$
function INSERT(X)
  let h be the minimal integer such that $1 + \sum_{j=0}^{h} |\mathcal{X}_j| \le 2^h$
  $\mathcal{X}_h$.INITIALIZE$(\{X\} \cup \mathcal{X}_0 \cup \dots \cup \mathcal{X}_h)$ `/* merge */`
  $\mathcal{X}_j$.INITIALIZE$(\emptyset)$ for $j = 0, \dots h-1$
  foreach $x \in \mathcal{X}_h.S$ **do**
    $n_x := 0$
  foreach $x \in \bigcup_{j=h+1}^{\log n} \mathcal{X}_j.S$ **do**
    if X.CONTAINS(x) **then**
      $n_x := n_x + 1$
function DELETE(X)
  find the bin $\mathcal{X}_j$ containing X
  refreshed := $\mathcal{X}_j$.DELETE(X)
  if refreshed **then**
    foreach $x \in \mathcal{X}_j.S$ **do**
      $n_x := \#\{X \in \mathcal{X}_0 \cup \dots \cup \mathcal{X}_{j-1} : X$.CONTAINS$(x)\}$
  foreach $x \in \bigcup_{k=j}^{\log n} \mathcal{X}_k.S$ **do**
    if X.CONTAINS(x) **then**
      $n_x := n_x - 1$
function ESTIMATE$()$
  denote $S_j := \mathcal{X}_j.S$ and $L_j := \mathcal{X}_j.L$ for $j = 0, 1, \dots, \log n$
  compute $R_j := \{x \in S_j : n_x = 0\}$ for $j = 0, 1, \dots, \log n$
  return $\sum_{j=0}^{\log n} 2^{L_j} |R_j|$

Proof. Fix a sequence of at most n insertions and deletions. Then the partition of surviving objects into bins $\mathcal{X}_1, \dots, \mathcal{X}_{\log n}$ is determined; in particular $U_j := \bigcup_{X \in \mathcal{X}_j} X$ is determined, and we write $V_j := U_j \setminus (U_0 \cup \cdots \cup U_{j-1})$. Note that the union of the surviving objects $U := \bigcup_{j=0}^{\log n} U_j = \bigcup_{j=0}^{\log n} V_j$ is a disjoint union of V_j's, thus $\operatorname{vol}(U) = \sum_{j=0}^{\log n} \operatorname{vol}(V_j)$.

Now let us consider a call to ESTIMATE. By Lemma 11, each S_j is a $(\frac{18}{n^2}, 6n)$-digest of U_j. Note that the algorithm computes $R_j := V_j \cap S_j$, so by Corollary 9, we have $2^{L_j}|R_j| \in \operatorname{vol}(V_j) \pm \varepsilon \operatorname{vol}(U_j)$ with probability at least $1 - \frac{18}{n^2} - \frac{12n}{n^4} = 1 - O(n^{-2})$. By a union bound, this holds for all j simultaneously with probability $1 - O(n^{-1})$. Under this event,

$$\sum_{j=0}^{\log n} 2^{L_j}|R_j| \in \operatorname{vol}(U) \pm \varepsilon \sum_{j=0}^{\log n} \operatorname{vol}(U_j).$$

We can crudely bound the additive error by $\varepsilon(1 + \log n)\operatorname{vol}(U) = \hat{\varepsilon}\operatorname{vol}(U)$, just as desired.

Next we analyze the running time. We will charge the cost of DELETE to insertions. Note that in DELETE, it takes time $O(\log^2(n))$ to compute the bin j containing X, if for each j we store (pointers to) all objects in $\mathcal{X}_j$ in a binary search tree. The last for-loop takes time $O(\log^2(n)/\varepsilon^2)$. To bound the running time of the "if refreshed" block, we change our viewpoint and focus on the sequence of deletions occurring between two merges of a particular bin $\mathcal{X}_j$.

- The total cost of $\mathcal{X}_j$.DELETE is $O(2^j \log^2(n/\varepsilon)/\varepsilon^2)$ in expectation by Lemma 11.
- With probability $1 - O(n^{-2})$, there are $O(\log(n/\varepsilon))$ refreshes between two merges of bin $\mathcal{X}_j$ by Lemma 11, so the if-block is executed $O(\log(n/\varepsilon))$ times. Each time, the algorithm recomputes n_x for every $x \in S_j$ by testing against at most $\sum_{k=0}^{j-1} 2^k \le 2^j$ objects, which takes time $O(2^j \log(n)/\varepsilon^2)$. Therefore, the total contribution of the if-block is $O(2^j \log^2(n/\varepsilon)/\varepsilon^2)$ in expectation.

Therefore, over the sequence of deletions between two merges of $\mathcal{X}_j$, deletions from $\mathcal{X}_j$ take expected total time $O(2^j \log^2(n/\varepsilon)/\varepsilon^2)$. Note that at least 2^j insertions occur between two merges of bin $\mathcal{X}_j$. We can thus charge the cost of deletions to insertions. Each insertion is charged $O(\log n)$ times (once for every bin), so we charge each insertion with a cost of $O(\log^3(n/\varepsilon)/\varepsilon^2)$ in expectation.

Now we analyze the running time of INSERT. It depends on the value h. By Lemma 11, the merge step costs $O\left(\sum_{j=0}^{h} 2^j \log^2(n/\varepsilon)/\varepsilon^2\right) = O\left(2^h \log^2(n/\varepsilon)/\varepsilon^2\right)$, which can be expensive in the worst case. However, bin $\mathcal{X}_h$ is merged at most once every 2^h updates, so it contributes amortized cost $O(\log^2(n/\varepsilon)/\varepsilon^2)$ to each insertion. Summing over all h, the amortized cost of merges is $O(\log^3(n/\varepsilon)/\varepsilon^2)$ per insertion. Next we consider the foreach-loops. Since each digest contains $O(\log(n)/\varepsilon^2)$ points by Lemma 11, and the algorithm spends constant time on each point, these loops take time $O(\log^2(n)/\varepsilon^2)$. To summarize, INSERT runs in amortized expected time $O(\log^3(n/\varepsilon)/\varepsilon^2) = O(\log^5(n/\varepsilon)/\hat{\varepsilon}^2)$ (and DELETE runs in amortized time 0).

The running time of ESTIMATE is clearly $O(\log^2(n)/\varepsilon^2) = O(\log^4(n)/\hat{\varepsilon}^2)$.

The space of bin $\mathcal{X}_j$ is $O(2^j + \log(n)/\varepsilon^2)$ by Lemma 11, and we can charge the space of counters n_x to it. In total the space complexity is $O\left(\sum_{j=0}^{\log n}(2^j + \log(n)/\varepsilon^2)\right) = O(n + \log^4(n)/\hat{\varepsilon}^2)$. ◀

Theorem 1 now follows from Theorem 12 together with Lemma 4.1 in [5].

References

1 Pankaj K. Agarwal, Lars Arge, and Frank Staals. Improved dynamic geodesic nearest neighbor searching in a simple polygon. In *Proc. 34th International Symposium on Computational Geometry (SoCG)*, volume 99 of *LIPIcs*, pages 4:1–4:14, 2018. doi:10.4230/LIPIcs.SOCG.2018.4.

2 Pankaj K. Agarwal, Hsien-Chih Chang, Subhash Suri, Allen Xiao, and Jie Xue. Dynamic geometric set cover and hitting set. *ACM Transactions on Algorithms*, 18(4):40:1–40:37, 2022. doi:10.1145/3551639.

3 Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proc. 21st ACM Symposium on Principles of Database Systems (PODS)*, pages 1–16, 2002. doi:10.1145/543613.543615.

4 Jon Louis Bentley and James B Saxe. Decomposable searching problems I. Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980. doi:10.1016/0196-6774(80)90015-2.

5 Sujoy Bhore, Karl Bringmann, Timothy M. Chan, and Yanheng Wang. Dynamic and streaming algorithms for union volume estimation, 2026. arXiv:2602.16306.

6 Sujoy Bhore and Timothy M. Chan. Fast static and dynamic approximation algorithms for geometric optimization problems: Piercing, independent set, vertex cover, and matching. In *Proc. 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2357–2386, 2025. doi:10.1137/1.9781611978322.79.

7 Sujoy Bhore, Martin Nöllenburg, Csaba D. Tóth, and Jules Wulms. Fully dynamic maximum independent sets of disks in polylogarithmic update time. In *40th International Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 19:1–19:16, 2024. doi:10.4230/LIPIcs.SOCG.2024.19.

8 Vladimir Braverman. Sliding window algorithms. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 2006–2011. Springer New York, New York, NY, 2016. doi:10.1007/978-1-4939-2864-4_797.

9 Karl Bringmann and Tobias Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Computational Geometry*, 43(6-7):601–610, 2010. doi:10.1016/J.COMGEO.2010.03.004.

10 Karl Bringmann, Kasper Green Larsen, André Nusser, Eva Rotenberg, and Yanheng Wang. Approximating Klee's measure problem and a lower bound for union volume estimation. In *Proc. 41st International Symposium on Computational Geometry (SoCG)*, pages 25:1–25:16, 2025. doi:10.4230/LIPIcs.SOCG.2025.25.

11 Clément Canonne. A short note on Poisson tail bounds, 2019. URL: https://github.com/ccanonne/probabilitydistributiontoolbox/blob/master/poissonconcentration.pdf.

12 Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. *ACM Transactions on Database Systems*, 48(1):1–45, 2023. doi:10.1145/3578517.

13 Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. In *Proc. 39th ACM Symposium on Principles of Database Systems (PODS)*, pages 393–409, 2020. doi:10.1145/3375395.3387662.

14 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable approximate model counter. In *Proc. 19th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 200–216, 2013. doi:10.1007/978-3-642-40627-0_18.

15 Timothy M. Chan. A (slightly) faster algorithm for Klee's measure problem. *Computational Geometry*, 43(3):243–250, 2010. doi:10.1016/J.COMGEO.2009.01.007.

16 Timothy M. Chan. Klee's measure problem made easy. In *Proc. 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 410–419, 2013. doi:10.1109/FOCS.2013.51.

17 Timothy M. Chan, Qizheng He, Subhash Suri, and Jie Xue. Dynamic geometric set cover, revisited. *SIAM Journal on Computing*, 54(3):664–701, 2025. doi:10.1137/23M1596582.

18 Timothy M. Chan and Zhengcheng Huang. Dynamic geometric connectivity in the plane with constant query time. In *40th International Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 36:1–36:13, 2024. doi:10.4230/LIPIcs.SOCG.2024.36.

19 Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM Journal on Computing*, 40(2):333–349, 2011. doi:10.1137/090751670.

20 Timothy M. Chan and Bashir S. Sadjad. Geometric optimization problems over sliding windows. *Int. J. Comput. Geom. Appl.*, 16(2-3):145–158, 2006. doi:10.1142/S0218195906001975.

21 Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4):523–544, 2007. doi:10.1007/S00778-006-0004-3.

22 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002. doi:10.1137/S0097539701398363.

23 Martin E. Dyer, Alan M. Frieze, and Ravi Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *Journal of the ACM*, 38(1):1–17, 1991. doi:10.1145/102782.102783.

24 Joan Feigenbaum, Sampath Kannan, and Jian Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2005. doi:10.1007/S00453-004-1105-2.

25 Shiyuan Feng, William Swartworth, and David P. Woodruff. Tight bounds for heavy-hitters and moment estimation in the sliding window model. In *Proc. 52nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 75:1–75:19, 2025. doi:10.4230/LIPIcs.ICALP.2025.75.

26 Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. In *Proc. 21st ACM Symposium on Computational Geometry (SoCG)*, pages 142–149, 2005. doi:10.1145/1064092.1064116.

27 Egor Gorbachev and Marvin Künnemann. Combinatorial designs meet hypercliques: Higher lower bounds for Klee's measure problem and related problems in dimensions $d \geq 4$. In *Proc. 39th International Symposium on Computational Geometry (SoCG)*, pages 36:1–36:14, 2023. doi:10.4230/LIPIcs.SOCG.2023.36.

28 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In *36th International Symposium on Computational Geometry (SoCG)*, volume 164 of *LIPIcs*, pages 51:1–51:14, 2020. doi:10.4230/LIPIcs.SOCG.2020.51.

29 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proc. 36th annual ACM Symposium on Theory of Computing*, pages 373–380, 2004. doi:10.1145/1007352.1007413.

30 Rajesh Jayaram, David P. Woodruff, and Samson Zhou. Truly perfect samplers for data streams and sliding windows. In *Proc. 41st ACM Symposium on Principles of Database Systems (PODS)*, pages 29–40, 2022. doi:10.1145/3517804.3524139.

31 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for L_p samplers, finding duplicates in streams, and related problems. In *Proc. 30th ACM Symposium on Principles of Database Systems (PODS)*, pages 49–58, 2011. doi:10.1145/1989284.1989289.

32 David R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Review*, 43(3):499–522, 2001. doi:10.1137/S0036144501387141.

33 Richard M. Karp and Michael Luby. Monte-Carlo algorithms for the planar multiterminal network reliability problem. *Journal of Complexity*, 1(1):45–64, 1985. doi:10.1016/0885-064X(85)90021-4.

34 Richard M. Karp, Michael Luby, and Neal Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989. doi:10.1016/0196-6774(89)90038-2.

35 Benny Kimelfeld, Yuri Kosharovsky, and Yehoshua Sagiv. Query efficiency in probabilistic XML models. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 701–714, 2008. doi:10.1145/1376616.1376687.

36 Victor Klee. Can the measure of $\bigcup_1^n [a_i, b_i]$ be computed in less than $O(n \log n)$ steps? *The American Mathematical Monthly*, 84(4):284–285, 1977. doi:10.1080/00029890.1977.11994336.

37 Marvin Künnemann. A tight (non-combinatorial) conditional lower bound for Klee's measure problem in 3D. In *Proc. 63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 555–566, 2022. doi:10.1109/FOCS54457.2022.00059.

38 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

39 Kuldeep S. Meel, Sourav Chakraborty, and N. V. Vinodchandran. Estimation of the size of union of Delphic sets: Achieving independence from stream size. In *Proc. 41st ACM Symposium on Principles of Database Systems (PODS)*, pages 41–52, 2022. doi:10.1145/3517804.3526222.

40 Kuldeep S. Meel, N. V. Vinodchandran, and Sourav Chakraborty. Estimating the size of union of sets in streaming models. In *Proc. 40th ACM Symposium on Principles of Database Systems (PODS)*, pages 126–137, 2021. doi:10.1145/3452021.3458333.

41 Shanmugavelayutham Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005. doi:10.1561/0400000002.

42 Mridul Nandi, N. V. Vinodchandran, Arijit Ghosh, Kuldeep S. Meel, Soumit Pal, and Sourav Chakraborty. Improved streaming algorithm for the Klee's measure problem and generalizations. In *Proc. 27th International Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX/RANDOM)*, pages 26:1–26:21, 2024. doi:10.4230/LIPIcs.APPROX/RANDOM.2024.26.

43 Yakov Nekrich. Dynamic planar point location in optimal time. In *53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1003–1014, 2021. doi:10.1145/3406325.3451100.

44 Mark H. Overmars and Chee-Keng Yap. New upper bounds in Klee's measure problem. *SIAM Journal on Computing*, 20(6):1034–1045, 1991. doi:10.1137/0220065.

45 A. Pavan, N. Variyam Vinodchandran, Arnab Bhattacharyya, and Kuldeep S. Meel. Model counting meets F_0 estimation. *ACM Transactions on Database Systems*, 48(3):1–28, 2023. doi:10.1145/3603496.

46 Christopher Ré, Nilesh Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. 23rd IEEE International Conference on Data Engineering*, pages 886–895, 2006. doi:10.1109/ICDE.2007.367934.

47 Srikanta Tirthapura and David Woodruff. Rectangle-efficient aggregation in spatial data streams. In *Proc. 31st ACM Symposium on Principles of Database Systems (PODS)*, pages 283–294, 2012. doi:10.1145/2213556.2213595.

48 David P. Woodruff and Samson Zhou. Tight bounds for adversarially robust streams and sliding windows via difference estimators. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1183–1196, 2021. doi:10.1109/FOCS52979.2021.00116.