Near-Optimal Dynamic Data Structures for Maximum Depth and Klee's Measure of Boxes

Sujoy Bhore
Indian Institute of Technology Bombay, Mumbai, India

Subhash Suri
University of California, Santa Barbara, CA, USA

Jie Xue
New York University Shanghai, China

Xiongxin Yang
University of California, Santa Barbara, CA, USA

Jiumu Zhu
New York University Shanghai, China

Abstract

We study two fundamental geometric problems on a dynamic set of n axis-parallel boxes in d-dimensional space. The *maximum depth* problem asks for the largest number of boxes that contain a common point, whereas *Klee's measure* problem asks for the volume of the union of the boxes. We present fully dynamic *exact* data structures for both problems achieving $\widetilde{O}(n^{(d-1)/2})$ amortized update time. This update time is optimal for an exact dynamic algorithm, up to logarithmic factors, assuming the Combinatorial k-Clique Hypothesis. Previously, matching bounds were established only for $d = 1$ [Imai and Asano, J. Algo.'83], and for $d = 2$ [Suri, Xue, Yang, and Zhu, SoCG'25].

Our approach integrates a classic grid-based partition framework with a novel charging analysis that controls the cost of structure-sensitive offline routines within each cell. This argument allows us to perform a global aggregation of the update time, by circumventing the worst-case costs associated with individual cell updates. We believe this technique may be of independent interest for other dynamic geometric problems.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases dynamic algorithms, maximum depth

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.34

Category Track A: Algorithms, Complexity and Games

1 Introduction

For a set $\mathcal{O}$ of geometric objects, the *maximum depth* of $\mathcal{O}$ is the largest number of objects in $\mathcal{O}$ that have a nonempty common intersection. Maximum depth is an important parameter capturing the local density of a family of geometric objects, and it has appeared in many geometric and combinatorial settings. The maximum depth is closely related to the clique number of the intersection graph of the objects. For example, for families with *Helly number* two – including axis-parallel rectangles and boxes – pairwise intersection implies a nonempty common intersection, and thus the maximum depth is exactly the size of a maximum clique in the corresponding intersection graph [19, 26]. For fat objects in fixed dimension, the maximum depth still provides a constant-factor approximation to the maximum clique size of the intersection graph.

53rd International Colloquium on Automata, Languages, and Programming (ICALP 2026).
Editors: Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis;
Article No. 34; pp. 34:1–34:21


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Computing the maximum depth of an input family of objects in $\mathbb{R}^d$ is a fundamental problem in computational geometry and has been studied since the early days of the field [24, 18, 4, 1, 3, 13, 30]. For (axis-parallel) boxes in $\mathbb{R}^d$, the maximum depth problem can be solved in $\widetilde{O}(n^{d/2})$ time [29, 12, 13]. This bound is believed to be tight up to logarithmic factors, since any improvement would break the Combinatorial k-Clique Hypothesis [12].

For boxes, a closely related problem is *Klee's measure*: given a set $\mathcal{B}$ of boxes, compute the volume of their union (the *Klee's measure* of $\mathcal{B}$). The problem was first posed in 1977 by Klee [25], and it is among the earliest questions that influenced the development of computational geometry. Since then, it has been studied extensively [6, 31, 29, 12, 13, 5]. Moreover, algorithms for Klee's measure can typically be adapted to solve the maximum depth problem with essentially the same running time [5]. In 1991, Overmars and Yap [29] gave the first $O(n^{d/2} \log n)$-time algorithm for Klee's measure in $\mathbb{R}^d$. About two decades later, Chan [12, 13] simplified and refined the approach to obtain an $O(n^{d/2})$-time algorithm. Again, this bound is conditional tight up to logarithmic factors [12].

In this paper, we study the *dynamic* versions of the maximum depth problem and Klee's measure problem for boxes in $\mathbb{R}^d$. In this fully dynamic setting, boxes may be inserted and deleted over time, and the goal is to maintain the maximum depth (respectively, Klee's measure) of the current set efficiently. Since the seminal work of Overmars and van Leeuwen [28], dynamic geometric data structures have played a central role in computational geometry. Many classic geometric problems have been studied extensively in the dynamic model, leading to a rich collection of efficient data structures, including convex hulls [23, 10], closest pairs [20, 21, 14], range searching [27, 17], geometric width [11], geometric set cover and hitting set [15, 2, 16], geometric independent set [22, 9, 7], and geometric vertex cover and matching [8], among others. Due to the importance of maximum depth and Klee's measure, it is natural to ask how efficiently one can maintain these quantities for a dynamic set of boxes in $\mathbb{R}^d$.

The near-optimal $\widetilde{O}(n^{d/2})$ running time for the static problems yields a lower bound for the dynamic setting via a standard sweep-line reduction: Given a dynamic data structure for maximum depth (or Klee's measure) for boxes in $\mathbb{R}^d$ with update time $T(n)$, one can obtain a static algorithm for boxes in $\mathbb{R}^{d+1}$ with running time $O(n \cdot T(n))$ by applying the sweep-line framework of Imai and Asano [24]. Consequently, a dynamic data structure in $\mathbb{R}^d$ with update time $O(n^{(d-1)/2-\varepsilon})$ would yield a static algorithm in $\mathbb{R}^{d+1}$ with running time $O(n^{(d+1)/2-\varepsilon})$. In light of this reduction, the most compelling goal is to design dynamic data structures whose update time matches the $n^{(d-1)/2}$ barrier up to logarithmic factors. We formulate this as the following question, which is the main focus of this paper.

Do there exist dynamic data structures for maximum depth and Klee's measure of axis-parallel boxes in $\mathbb{R}^d$ with update time $\widetilde{O}(n^{(d-1)/2})$?

For $d = 1$, the classic work of Imai and Asano [24] achieves $O(\log n)$ update time and thus answers the above question affirmatively. Beyond this base case, however, the question had remained poorly understood. Only very recently, Suri, Xue, Yang, and Zhu [30] proposed (among other results) a dynamic maximum-depth data structure for rectangles in the plane with $O(n^{1/2} \log n)$ amortized update time. With additional work, their ideas can be adapted to maintain Klee's measure for planar rectangles as well, yielding $\widetilde{O}(n^{1/2})$ update time. This resolves the above question for $d = 2$. Unfortunately, while the 2D framework [30] extends to higher dimensions, it does not achieve the target bound of $\widetilde{O}(n^{(d-1)/2})$, already for $d = 3$ (see discussion in Section 2). Prior to this work, the above question was open for any value of $d \geq 3$.

The main contribution of this paper is to answer the question in the affirmative for every fixed $d \in \mathbb{N}$. Specifically, we design fully dynamic data structures for both maximum depth and Klee's measure of axis-parallel boxes in $\mathbb{R}^d$ with $\widetilde{O}(n^{(d-1)/2})$ update time.

▶ **Theorem 1.** *There exists a fully dynamic maximum depth data structure for axis-parallel boxes in $\mathbb{R}^d$ with $\widetilde{O}(n^{(d-1)/2})$ amortized update time.*

Our techniques extend beyond depth queries and also yield an efficient dynamic structure for maintaining the union volume for boxes.

▶ **Theorem 2.** *There exists a fully dynamic Klee's measure data structure for axis-parallel boxes in $\mathbb{R}^d$ with $\widetilde{O}(n^{(d-1)/2})$ amortized update time.*

2 Technical Overview

We provide an overview of our fully dynamic maximum-depth data structure. We focus on the high-level ideas and the role of the key charging quantities. The dynamic Klee's measure structure follows the similar framework with a different interval primitive.

2.1 Overview of the 2D data structure

We start by recalling the main idea behind the 2D data structure of [30]. It partitions the plane into r vertical strips so that each strip contains $O(n/r)$ rectangle corners, and maintains the maximum depth inside each strip separately. Although $\Omega(n)$ rectangles may intersect a fixed strip, all but $O(n/r)$ of them have no corner in it; we call these rectangles *good* (for the strip), and the remaining $O(n/r)$ rectangles *bad*. Inside the strip, every good rectangle spans the strip in the x-direction and is therefore equivalent to an interval on the y-axis. This reduces the subproblem inside the strip to maintaining the maximum depth of a dynamic set of (weighted) intervals: updates caused by good rectangles take only $\widetilde{O}(1)$ time, while updates caused by bad rectangles take $\widetilde{O}(n/r)$ time, since there are only $O(n/r)$ bad rectangles whose contribution must be recomputed. Moreover, each rectangle is bad for only $O(1)$ strips (namely those containing its left or right vertical side), and is good for all other strips. Overall, the update time is $\widetilde{O}(r + n/r)$, which becomes $\widetilde{O}(\sqrt{n})$ by setting $r = \sqrt{n}$.

2.2 Challenges in higher dimensions

A natural goal in $\mathbb{R}^d$ is to replicate the same high-level principle: partition space so that most boxes behave "one-dimensionally" inside each region (and can be maintained cheaply), while only a small set of boxes interacts with the region boundaries. Two straightforward generalizations break down for essentially the same reason: the contribution of the boundary-interacting (*"bad"*) boxes can have prohibitively large combinatorial complexity. There are multiple natural generalizations of this approach to higher dimensions; however, none of them guarantees the desired bounds.

- **Partition into r slabs along one axis:** Partition $\mathbb{R}^d$ into r slab regions $S_1, \ldots, S_r$ perpendicular to the x_1-axis, each containing $O(n/r)$ box corners. Inside a slab S_i, good boxes are those with no corner in S_i; they span the slab in the x_1-direction and thus reduce to $(d-1)$-dimensional boxes via projection. The difficulty is with the bad boxes: their contribution inside S_i can be described by a piecewise-constant depth function over $\mathbb{R}^{d-1}$ whose rectilinear decomposition may require $\Omega((n/r)^{d-1})$ regions in the worst case (because arrangements of t boxes in $\mathbb{R}^{d-1}$ can have complexity $\Omega(t^{d-1})$). Thus, even assuming optimal lower-dimensional dynamic structures, the induced subproblem sizes are too large to support the target update time.

- **Partition into r^{d-1} cells in the first $d-1$ dimensions:** Choose $r-1$ hyperplanes perpendicular to each of the first $d-1$ coordinate axes, so that these hyperplanes form a grid partition of $\mathbb{R}^d$ into r^{d-1} cells, each of the form $\square \times \mathbb{R}$, where $\square \subseteq \mathbb{R}^{d-1}$ is a $(d-1)$-dimensional axis-parallel box. Inside a fixed cell $\square \times \mathbb{R}$, a box is *good* if it spans the cell in the first $d-1$ coordinates, i.e., it completely covers $\square$. Restricted to the cell, such a box contributes exactly one interval on the last axis and can be maintained with a dynamic maximum-depth data structure for intervals. The bottleneck is the remaining *bad* boxes, namely those that intersect $\square$ but do not cover it, so some side facet of the box crosses the base $\square$. A direct adaptation would handle these bad boxes in each affected cell by sweeping along the last axis and repeatedly invoking a $(d-1)$-dimensional dynamic maximum-depth subroutine. Even assuming optimal lower-dimensional update bounds, this becomes too expensive once summed over the $\Theta(r^{d-2})$ cells whose bases can be intersected by the boundary of a single updated box.

2.3 Overview of Our Techniques

At a high-level, we also follow the grid partition into cells of the form $\square \times \mathbb{R}$, and we retain the same "good boxes $\Rightarrow$ interval updates" principle. The key new ingredient is a structure-sensitive way to process the bad boxes *within a fixed cell*.

Fix a cell $\square \subseteq \mathbb{R}^{d-1}$, and consider the set of bad boxes whose projections intersect $\square$ but do not cover it. Our data structure represents their contribution to the depth inside $\square \times \mathbb{R}$ by a piecewise-constant function on the last coordinate (equivalently, by a set of weighted (*type-2*) intervals, one per piece). Thus, for a cell $\square \times \mathbb{R}$, the total depth reduces to the maximum depth of a set of weighted intervals on the last axis: *type-1* intervals coming from good boxes (weight 1) plus the *type-2* intervals capturing the bad-box contribution.

The core task in an update is therefore the following: when the bad-box set for a cell changes, we must recompute the new inside-$\square$ depth function (and hence the new set of type-2 intervals) efficiently. We accomplish this via an offline subroutine that further refines $\square$ by inserting additional axis-parallel hyperplanes so that, inside every resulting subcell, each bad box becomes equivalent to a *slab* in $\mathbb{R}^{d-1}$ (i.e., it is bounded in exactly one of the first $d-1$ directions and is either fully present or fully absent in the other directions throughout the subcell). Once this holds, the contribution of the bad boxes inside each subcell reduces to a one-dimensional interval problem on the last axis.

The key idea to keep the refinement small is to avoid cutting at every bad-box endpoint in every partially overlapping direction. Instead, we choose an ordering (permutation) σ of the first $d-1$ coordinate axes. For a given bad box, we treat the earliest axis (in σ-order) along which it only partially overlaps $\square$ as its *active* direction (the slab direction), and we cut at the box endpoints only in the *later* partially overlapping directions. Equivalently, the box *charges* every partially overlapping direction except its first one under σ. In the formal development, this choice is captured by the indicators $\phi_{\sigma,\square}(\cdot, i)$ and the counts $\Phi_{\sigma,\square}(\cdot, i) = 1 + \sum \phi_{\sigma,\square}(\cdot, i)$. For a fixed σ, the number of inserted hyperplanes perpendicular to axis i is proportional to $\Phi_{\sigma,\square}(\cdot, i) - 1$, and the number of resulting subcells is bounded by $O(\prod_i \Phi_{\sigma,\square}(\cdot, i))$, which also governs the running time of the offline subroutine.

In the dynamic data structure, we maintain the values $\Phi_{\sigma,\square}(\cdot, i)$ for every cell $\square$, every axis $i \in [d-1]$, and every permutation σ. When an update changes the bad-box set in a cell, we pick the permutation σ that minimizes the product bound $\prod_i \Phi_{\sigma,\square}(\cdot, i)$ and invoke the offline subroutine under this choice to compute the updated inside-$\square$ depth function (and hence the new type-2 intervals) for that cell. Meanwhile, updates from good boxes remain fast interval insertions/deletions (type-1 updates).

The remaining challenge is to argue that the total cost of these offline computations over all cells affected by a single box update is bounded by $\widetilde{O}(n^{(d-1)/2})$. This is where the main charging analysis enters. The affected cells lie on $O(d)$ "slices" of the grid determined by the facets of the projected box: fixing a coordinate $t \in [d-1]$ and a slab index $a \in [r]$ specifies a family of cells $V_t(a)$. A global charging argument relates each local quantity $\Phi_{\sigma,\square_v}(\cdot, i)$ to the distribution of box corners among these slices and shows that, for any slice $V_t(a)$, choosing an order σ whose first element is t yields a strong bound on $\sum_{\square \in V_t(a)} \prod_i \Phi_{\sigma,\square}(\cdot, i)$ (Lemma 11). Combining this with the fact that each affected cell contains only $O(n/r)$ bad boxes yields the desired amortized update time after setting $r = \sqrt{n}$.

3 Preliminaries

Basic Notations. We use $\mathbb{N}$ to denote the set of positive integers and define $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. For an integer $n \in \mathbb{N}$, we write $[n] = \{1, \ldots, n\}$. Throughout the paper, all rectangles and boxes are axis-parallel.

Projection. Let $B = \prod_{i=1}^{d} [x_i^-, x_i^+]$ be a box in $\mathbb{R}^d$. For $i \in [d]$, let $\mathbf{proj}_i(B) := [x_i^-, x_i^+]$ denote the interval obtained by projecting B onto dimension i. More generally, for $I = \{i_1, \ldots, i_k\} \subseteq [d]$ where $i_1 < \cdots < i_k$, let $\mathbf{proj}_I(B) := \prod_{j=1}^{k} [x_{i_j}^-, x_{i_j}^+]$ denote the $|I|$-dimensional box obtained by projecting B onto the dimensions in I. For a set $\mathcal{B}$ of boxes in $\mathbb{R}^d$, we write $\mathbf{proj}_i(\mathcal{B}) = \{\mathbf{proj}_i(B) : B \in \mathcal{B}\}$ for $i \in [d]$ and $\mathbf{proj}_I(\mathcal{B}) = \{\mathbf{proj}_I(B) : B \in \mathcal{B}\}$ for $I \subseteq [d]$.

Maximum depth. Let $\mathcal{O}$ be a set of geometric objects in $\mathbb{R}^d$. For a point $p \in \mathbb{R}^d$, we define $\mathsf{dep}_p(\mathcal{O}) = |\{O \in \mathcal{O} : p \in O\}|$, which is called the *depth* of $\mathcal{O}$ at p. For a region $R \subseteq \mathbb{R}^d$, we define $\mathsf{dep}_R(\mathcal{O}) = \sup_{p \in R} \mathsf{dep}_p(\mathcal{O})$, which is called the *maximum depth* of $\mathcal{O}$ inside R. We write $\mathsf{dep}(\mathcal{O}) = \mathsf{dep}_{\mathbb{R}^d}(\mathcal{O})$ for the maximum depth of $\mathcal{O}$.

Klee's measure. For a geometric object O in $\mathbb{R}^d$, we denote by $V(O)$ the *volume* of O. Let $\mathcal{O}$ be a set of geometric objects in $\mathbb{R}^d$. For a region $R \subseteq \mathbb{R}^d$, we define $\mathsf{Klee}_R(\mathcal{O}) = V\big((\bigcup_{O \in \mathcal{O}} O) \cap R\big)$, which is called the *Klee's measure* of $\mathcal{O}$ inside R. We write $\mathsf{Klee}(\mathcal{O}) = \mathsf{Klee}_{\mathbb{R}^d}(\mathcal{O})$ for the Klee's measure of $\mathcal{O}$. Note that $\mathsf{Klee}(\mathcal{O})$ can be ∞ if $\mathcal{O}$ contains unbounded objects.

Piecewise constant functions. We say that a function $f : \mathbb{R} \to \mathbb{R}$ is *piecewise constant* if $\mathbb{R}$ can be partitioned into finitely many intervals $I_1, \ldots, I_r$ such that, for every $i \in [r]$, the restriction $f_{|I_i}$ is constant. Such a partition need not be unique. If we require r to be minimum possible, then the resulting partition into maximal intervals on which f is constant is uniquely determined; we call these intervals the *pieces* of f and denote them by $I_1^*, \ldots, I_r^*$. For each piece I_i^*, we write $f(I_i^*) = f(x)$ for an arbitrary $x \in I_i^*$; this value is well-defined since $f_{|I_i^*}$ is constant.

4 Offline dynamic maximum depth

In this section, we consider an offline variant of the dynamic maximum depth problem and propose an algorithm whose running time depends not only on the number of update operations but also on the structure of the boxes involved. We will use this offline algorithm as a subroutine in our fully dynamic data structure.

Offline Inside-R Box Maximum Depth in $\mathbb{R}^c$
Input: An initially empty set $\mathcal{B}$ of boxes in $\mathbb{R}^c$, a fixed box region $R \subseteq \mathbb{R}^c$, and a sequence of n updates to $\mathcal{B}$, each of which is of one of the following two types:
- **Insertion:** Insert a new box into $\mathcal{B}$.
- **Deletion:** Delete an existing box from $\mathcal{B}$

Output: A sequence $(\delta_1, \ldots, \delta_n)$, where $\delta_i \in \mathbb{N}_0$ is the maximum depth of $\mathcal{B}$ inside the region R after the i-th update.

4.1 Dynamic maximum depth for slabs

In designing our algorithm, we require an efficient dynamic maximum-depth data structure for *slabs*, i.e., boxes that are bounded in exactly one dimension. Formally, a *slab* L in $\mathbb{R}^c$ is a box such that there exists exactly one index $i \in [c]$ satisfying $\mathbf{proj}_i(L) \neq (-\infty, \infty)$.

If we have a dynamic set $\mathcal{L}$ of slabs in $\mathbb{R}^c$ that are all bounded in the same dimension, then maintaining the maximum depth of $\mathcal{L}$ reduces to maintaining the maximum depth of a dynamic set of intervals and can be done with logarithmic update time. Moreover, the following observation implies that even if the slabs in $\mathcal{L}$ are bounded in different dimensions, we can still maintain the maximum depth of $\mathcal{L}$ with logarithmic update time by handling each dimension separately.

▶ **Fact 3.** *Let $\mathcal{L}$ be a set of slabs in $\mathbb{R}^c$ and $\square$ be a fixed box in $\mathbb{R}^c$. Then we have $\mathsf{dep}_{\square}(\mathcal{L}) = \sum_{i=1}^{c} \mathsf{dep}_{\square}(\mathcal{L}_i)$, where $\mathcal{L}_i \subseteq \mathcal{L}$ consists of the slabs bounded in dimension i.*

Proof. Since $\mathcal{L} = \bigcup_{i=1}^{c} \mathcal{L}_i$, we have $\mathsf{dep}_{\square}(\mathcal{L}) \leq \sum_{i=1}^{c} \mathsf{dep}_{\square}(\mathcal{L}_i)$. It remains to show the reverse inequality. For each $i \in [c]$, let $p_i \in \square$ be a point satisfying $\mathsf{dep}_{p_i}(\mathcal{L}_i) = \mathsf{dep}_{\square}(\mathcal{L}_i)$. Since all slabs in $\mathcal{L}_i$ are perpendicular to the x_i-axis, we have $\mathsf{dep}_{p}(\mathcal{L}_i) = \mathsf{dep}_{p_i}(\mathcal{L}_i)$ for any point $p \in \mathbb{R}^c$ whose i-th coordinate is equal to that of p_i. Let p^* denote the point whose i-th coordinate equals the i-th coordinate of p_i for every $i \in [c]$. Then $\mathsf{dep}_{p^*}(\mathcal{L}_i) = \mathsf{dep}_{p_i}(\mathcal{L}_i)$ for all $i \in [c]$, and thus

$$\mathsf{dep}_{p^*}(\mathcal{L}) = \sum_{i=1}^{c} \mathsf{dep}_{p^*}(\mathcal{L}_i) = \sum_{i=1}^{c} \mathsf{dep}_{p_i}(\mathcal{L}_i) = \sum_{i=1}^{c} \mathsf{dep}_{\square}(\mathcal{L}_i).$$

Moreover, $p^* \in \square$ since $p_1, \ldots, p_c \in \square$. Therefore, $\mathsf{dep}_{\square}(\mathcal{L}) \geq \mathsf{dep}_{p^*}(\mathcal{L}) = \sum_{i=1}^{c} \mathsf{dep}_{\square}(\mathcal{L}_i)$. ◀

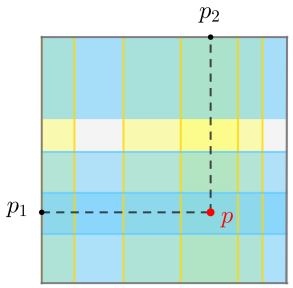

Figure 1 Illustration of Fact 3 in 2D. Blue (resp. yellow) rectangles represent horizontal (resp. vertical) slabs, with p_1 (resp. p_2) achieving maximum depth. The point p achieves maximum depth with respect to all slabs.

▶ **Lemma 4.** *Let $\square$ be a fixed box in $\mathbb{R}^c$. There exists a dynamic inside-$\square$ maximum depth data structure for slabs in $\mathbb{R}^c$ with $O(\log n)$ update time.*

Proof. Let $\mathcal{L}$ be a dynamic set of slabs in $\mathbb{R}^c$, and for each $i \in [c]$ let $\mathcal{L}_i \subseteq \mathcal{L}$ denote the slabs perpendicular to the x_i-axis. For each $i \in [c]$, we maintain a data structure $\mathbf{D}_i$ that supports updates to $\mathcal{L}_i$ and maintains $\mathsf{dep}_{\square}(\mathcal{L}_i)$. Maintaining $\mathsf{dep}_{\square}(\mathcal{L}_i)$ reduces to the one-dimensional problem of maintaining the maximum depth of the dynamic interval set $\mathbf{proj}_i(\mathcal{L}_i)$ inside the interval $\mathbf{proj}_i(\square)$, which admits $O(\log n)$ update time [24]. An insertion or deletion of a slab affects exactly one index i, and thus triggers a single update to the corresponding $\mathbf{D}_i$. Therefore, by Fact 3, $\mathbf{D}_1, \ldots, \mathbf{D}_c$ together maintain $\mathsf{dep}_{\square}(\mathcal{L})$. The update time is $O(\log n)$. ◀

4.2 Solving the offline problem

Let $\square$ be a box in $\mathbb{R}^c$. We say a box B in $\mathbb{R}^c$ is *nontrivial* in dimension $i \in [c]$ with respect to $\square$ if $\mathbf{proj}_i(\square) \cap \mathbf{proj}_i(B) \neq \emptyset$ and $\mathbf{proj}_i(\square) \not\subseteq \mathbf{proj}_i(B)$. For a permutation σ of $[c]$, an index $i \in [c]$, and a box B in $\mathbb{R}^c$, we define $\phi_{\sigma,\square}(B, i) = 1$ if both of the following conditions hold:

- B is nontrivial in dimension i with respect to $\square$,
- there exists $j <_\sigma i$ such that B is nontrivial in dimension j with respect to $\square$.

Otherwise, $\phi_{\sigma,\square}(B, i) = 0$. For a set $\mathcal{B}$ of boxes in $\mathbb{R}^c$, define $\Phi_{\sigma,\square}(\mathcal{B}, i) = 1 + \sum_{B \in \mathcal{B}} \phi_{\sigma,\square}(B, i)$.

▶ **Lemma 5.** *Let σ be a permutation of $[c]$ and $\square$ be a fixed box in $\mathbb{R}^c$. Then* Offline Inside-$\square$ Box Maximum Depth *in $\mathbb{R}^c$ can be solved in $O(n \log n \cdot \prod_{i=1}^{c} \Phi_{\sigma,\square}(\mathcal{B}^*, i))$ time, where $\mathcal{B}^*$ denotes the set of all boxes that are inserted into $\mathcal{B}$ in the sequence of updates.*

Proof. For simplicity, we present the proof for the case $\square = \mathbb{R}^c$ and drop the subscript $\square$ in the notation. the argument extends verbatim to an arbitrary fixed box $\square$ by restricting attention to $\square$. Without loss of generality, we assume the boxes in $\mathcal{B}^*$ are in general position in the sense that, for every $i \in [c]$, the intervals in $\mathbf{proj}_i(\mathcal{B}^*)$ have distinct endpoints.

For each $i \in [c]$, we define a set $\mathcal{H}_i$ of hyperplanes perpendicular to the x_i-axis as follows. Let $\mathcal{B}_i^* = \{B \in \mathcal{B}^* : \phi_\sigma(B, i) = 1\}$. For each endpoint $a \in \mathbb{R}$ of each interval in $\mathbf{proj}_i(\mathcal{B}_i^*)$, we include in $\mathcal{H}_i$ the hyperplane $x_i = a$. Note that $|\mathcal{H}_i| = 2|\mathcal{B}_i^*| = 2 \cdot \sum_{B \in \mathcal{B}^*} \phi_\sigma(B, i)$. The hyperplanes in $\mathcal{H}_1, \ldots, \mathcal{H}_c$ partition the space $\mathbb{R}^c$ into $\prod_{i=1}^{c}(|\mathcal{H}_i| + 1) = O(\prod_{i=1}^{c} \Phi_\sigma(\mathcal{B}^*, i))$ cells; let Γ denote the resulting set of cells. For each cell $\gamma \in \Gamma$, we maintain a dynamic data structure $\mathbf{D}_\gamma$ that maintains the maximum depth of the current set $\mathcal{B}$ inside the interior of γ; denote this value by y_γ. Under the general-position assumption, the maximum depth of $\mathcal{B}$ is attained in the interior of some cell, and thus equals $\max_{\gamma \in \Gamma} y_\gamma$. Therefore, replaying the n updates for every $\gamma \in \Gamma$ yields a total running time $O(|\Gamma| \cdot n \cdot t)$, where t is the maximum update time of each individual $\mathbf{D}_\gamma$.

The remaining task is to achieve $t = O(\log n)$. The key observation is that, inside the interior of any fixed cell $\gamma \in \Gamma$, every box $B \in \mathcal{B}^*$ behaves like a slab. To see this, fix $B \in \mathcal{B}^*$ and suppose B is nontrivial in dimensions $i_1, \ldots, i_p \in [c]$ with respect to $\mathbb{R}^c$, where $i_1 <_\sigma \cdots <_\sigma i_p$. By the definition of ϕ, we have $\phi_\sigma(B, i_2) = \cdots = \phi_\sigma(B, i_p) = 1$. Hence, for each $j \in \{2, \ldots, p\}$, we added to $\mathcal{H}_{i_j}$ the two hyperplanes corresponding to the endpoints of $\mathbf{proj}_{i_j}(B)$. It follows that, for each such j, the interior of $\mathbf{proj}_{i_j}(\gamma)$ is either contained in $\mathbf{proj}_{i_j}(B)$ or disjoint from it. Therefore, inside the interior of γ, the box B has the same intersection as the slab $\prod_{i=1}^{i_1-1}(-\infty, \infty) \times \mathbf{proj}_{i_1}(B) \times \prod_{i=i_1+1}^{c}(-\infty, \infty)$. As a consequence, inside the interior of γ, maintaining the maximum depth of $\mathcal{B}$ reduces to maintaining the maximum depth of a dynamic set of slabs. We can therefore instantiate $\mathbf{D}_\gamma$ using Lemma 4, which supports updates in $O(\log n)$ time. This gives $t = O(\log |\mathcal{B}^*|) = O(\log n)$ and yields a total running time as claimed. ◀

Let R be a region in $\mathbb{R}^{d-1}$ and let $\mathcal{B}$ be a set of boxes in $\mathbb{R}^d$. We define the *inside-R depth function* of $\mathcal{B}$ as the function $\delta_R^{\mathcal{B}} : \mathbb{R} \to \mathbb{N}_0$ given by $\delta_R^{\mathcal{B}}(p) := \mathsf{dep}_{R\times\{p\}}(\mathcal{B})$ for all $p \in \mathbb{R}$. Note that $\delta_R^{\mathcal{B}}$ is a piecewise constant function, for any choice of R. Indeed, as p moves from $-\infty$ to ∞, the set of boxes intersecting the slice $R \times \{p\}$ can change only when p reaches an endpoint of an interval in $\mathbf{proj}_d(\mathcal{B})$. Our offline algorithm above directly implies an algorithm for computing the inside-$\square$ depth function for a fixed box $\square$, which will be used in our dynamic data structure.

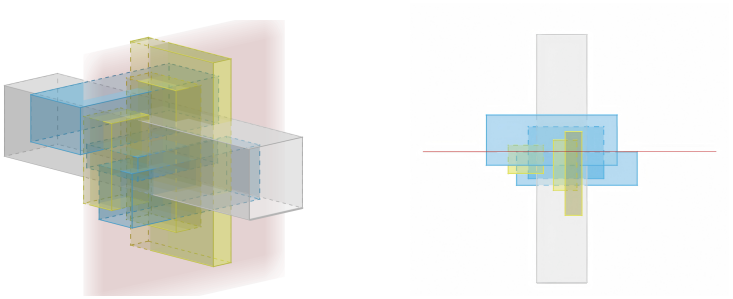

Figure 2 A cell $\gamma \in \Gamma$ (gray) that unbounded along one axis and its top-view. By the auxiliary partitions, all boxes intersecting γ are slabs. Colors indicate the bounded direction (yellow vs. blue). The cross-section induced by the red hyperplane yields a 2D instance of horizontal and vertical slabs as in Figure 1.

▶ **Corollary 6.** *Given a permutation σ of $[d-1]$, a box $\square$ in $\mathbb{R}^{d-1}$, and a set $\mathcal{B}$ of n boxes in $\mathbb{R}^d$, one can compute the inside-$\square$ depth function $\delta_{\square}^{\mathcal{B}} : \mathbb{R} \to \mathbb{N}_0$ in $O(n \log n \cdot \prod_{i=1}^{d-1} \Phi_{\sigma,\square}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i))$ time.*

Proof. Let P be the set of endpoints of the intervals in $\mathbf{proj}_d(\mathcal{B})$. Without loss of generality, assume that the boxes in $\mathcal{B}$ are in general position, so P contains exactly $2n$ distinct points $p_1, \ldots, p_{2n}$ with $p_1 < \cdots < p_{2n}$. Consider a point $q \in \mathbb{R}$ moving from $-\infty$ to ∞ on x_d-axis, and maintain a dynamic set $\mathcal{B}_0$ of boxes in $\mathbb{R}^{d-1}$, initially empty. Whenever q reaches a point p_i, if p_i is the left (resp., right) endpoint of $\mathbf{proj}_d(B)$ for some $B \in \mathcal{B}$, then we insert into $\mathcal{B}_0$ (resp., delete from $\mathcal{B}_0$) the box $\mathbf{proj}_{[d-1]}(B)$. Clearly, for any $a \in \mathbb{R}$, $\delta_{\square}^{\mathcal{B}}(a)$ equals the maximum depth of $\mathcal{B}_0$ inside $\square$ at the time $q = a$. Thus, to compute $\delta_{\square}^{\mathcal{B}}$, it suffices to maintain the maximum depth of $\mathcal{B}_0$ inside $\square$ under this sequence of insertions and deletions. This is an instance of Offline Inside-$\square$ Box Maximum Depth in $\mathbb{R}^{d-1}$, as the update sequence is fully known in advance. Moreover, the set of boxes that are ever inserted into $\mathcal{B}_0$ is exactly $\mathbf{proj}_{[d-1]}(\mathcal{B})$. Applying Lemma 5 completes the proof and yields the stated running time $O(n \log n \cdot \prod_{i=1}^{d-1} \Phi_{\sigma,\square}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i))$. ◀

5 Dynamic data structure for maximum depth

We now describe our dynamic data structure for maintaining the maximum depth of a dynamic set of boxes. Let $\mathcal{B}$ be a dynamic set of boxes in $\mathbb{R}^d$, which contains n boxes initially. Let r be a parameter to be specified later.

5.1 Construction of the data structure

In the preprocessing, for each dimension $i \in [d-1]$, we construct a set $\mathcal{H}_i$ of $r-1$ hyperplanes in $\mathbb{R}^d$ perpendicular to the x_i-axis such that, after sorting the hyperplanes along the x_i-axis, the number of corners of boxes in $\mathcal{B}$ lying between any two consecutive hyperplanes is $O(n/r)$.

Let $\mathcal{H} := \bigcup_{i=1}^{d-1} \mathcal{H}_i$. The hyperplanes in $\mathcal{H}$ partition $\mathbb{R}^d$ into r^{d-1} regions, which we call *cells*. We index cells by vectors $\vec{v} = (a_1, \ldots, a_{d-1}) \in [r]^{d-1}$, where a_i indicates the position of the cell along dimension i (i.e., the cell lies between the $(a_i - 1)$-st and a_i-th hyperplanes in $\mathcal{H}_i$, with the two outermost slabs treated as unbounded). Since we do not introduce hyperplanes perpendicular to the x_d-axis, each cell is of the form $\square \times \mathbb{R}$ for a box $\square \subseteq \mathbb{R}^{d-1}$. For each $\vec{v} \in [r]^{d-1}$, let $\square_{\vec{v}}$ denote the unique box in $\mathbb{R}^{d-1}$ such that the corresponding cell is $\square_{\vec{v}} \times \mathbb{R}$.

By the standard periodic reconstruction trick, we may assume that $|\mathcal{B}| = \Theta(n)$ throughout the reconstruction period. For simplicity, we also assume that, throughout the period, for each $i \in [d-1]$, the number of corners of boxes in $\mathcal{B}$ between any two consecutive hyperplanes in $\mathcal{H}_i$ remains $O(n/r)$. This assumption can be removed by applying the same splitting procedure as in [30]; we defer these implementation details to Appendix A.

For each $\vec{v} \in [r]^{d-1}$, our data structure maintains the following information:

- $\Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i)$ for all permutations σ of $[d-1]$ and all $i \in [d-1]$.
- The subset $\mathcal{B}_{\vec{v}} := \{B \in \mathcal{B} : \square_{\vec{v}} \cap \mathbf{proj}_{[d-1]}(B) \neq \emptyset \text{ and } \square_{\vec{v}} \not\subseteq \mathbf{proj}_{[d-1]}(B)\} \subseteq \mathcal{B}$.
- A dynamic maximum-depth data structure $\mathbf{D}_{\vec{v}}$ for weighted intervals, storing a multiset $\mathcal{I}_{\vec{v}}$ that consists of:
  - the interval $\mathbf{proj}_d(B)$ with weight 1, for every $B \in \mathcal{B}$ with $\square_{\vec{v}} \subseteq \mathbf{proj}_{[d-1]}(B)$; these are called *type-1* intervals,
  - the interval P with weight $\delta^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}(P)$, for every piece P of the function $\delta^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$; these are called *type-2* intervals.

The basic correspondence between $\mathcal{I}_{\vec{v}}$ and the inside-$\square_{\vec{v}}$ depth function is easy to establish.

▶ **Observation 7.** $\mathsf{dep}_p(\mathcal{I}_{\vec{v}}) = \delta^{\mathcal{B}}_{\square_{\vec{v}}}(p)$ *for all* $p \in \mathbb{R}$.

Proof. Fix $p \in \mathbb{R}$. By construction, $\mathsf{dep}_p(\mathcal{I}_{\vec{v}})$ equals $\delta^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}(p)$ plus the number of boxes $B \in \mathcal{B}$ satisfying $\square_{\vec{v}} \subseteq \mathbf{proj}_{[d-1]}(B)$ and $p \in \mathbf{proj}_d(B)$. The latter quantity is exactly $\delta^{\mathcal{B}\setminus\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}(p)$. Therefore, $\mathsf{dep}_p(\mathcal{I}_{\vec{v}}) = \delta^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}(p) + \delta^{\mathcal{B}\setminus\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}(p) = \delta^{\mathcal{B}}_{\square_{\vec{v}}}(p)$. ◀

The intervals in $\mathcal{I}_{\vec{v}}$ are stored in the data structure $\mathbf{D}_{\vec{v}}$ for weighted intervals. Observation 7 implies that $\mathsf{dep}(\mathcal{I}_{\vec{v}}) = \mathsf{dep}_{\square_{\vec{v}} \times \mathbb{R}}(\mathcal{B})$, and hence $\max_{\vec{v} \in [r]^{d-1}} \mathsf{dep}(\mathcal{I}_{\vec{v}}) = \mathsf{dep}(\mathcal{B})$. Accordingly, maintaining the values $\mathsf{dep}(\mathcal{I}_{\vec{v}})$ and taking their maximum over all $\vec{v} \in [r]^{d-1}$ suffices to obtain $\mathsf{dep}(\mathcal{B})$.

Consider an update on a box B that either inserts B into $\mathcal{B}$ or deletes an existing box $B \in \mathcal{B}$. We process the update for every $\vec{v} \in [r]^{d-1}$. First, we update the values $\Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i)$ for all permutations σ of $[d-1]$ and all $i \in [d-1]$. This is straightforward: for each such pair (σ, i), we compute $\phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(B), i)$ and either add it to or subtract it from the current value of $\Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i)$, depending on whether the update is an insertion or a deletion. We then update the set $\mathcal{B}_{\vec{v}}$, which is also immediate from its definition.

Next, we update the data structure $\mathbf{D}_{\vec{v}}$, since the interval multiset $\mathcal{I}_{\vec{v}}$ may change. There are three cases. If $\square_{\vec{v}} \cap \mathbf{proj}_{[d-1]}(B) = \emptyset$, then $\mathcal{I}_{\vec{v}}$ does not change. If $\square_{\vec{v}} \subseteq \mathbf{proj}_{[d-1]}(B)$, then $\mathcal{I}_{\vec{v}}$ changes only by a single type-1 interval: we insert $\mathbf{proj}_d(B)$ (with weight 1) if B is inserted, or delete the corresponding type-1 interval if B is deleted. Thus, it suffices to perform one insertion/deletion operation in $\mathbf{D}_{\vec{v}}$.

In the remaining case, we have $\square_{\vec{v}} \cap \mathbf{proj}_{[d-1]}(B) \neq \emptyset$ and $\square_{\vec{v}} \not\subseteq \mathbf{proj}_{[d-1]}(B)$, i.e., $B \in \mathcal{B}_{\vec{v}}$. In this case, the type-2 intervals in $\mathcal{I}_{\vec{v}}$ change, and we proceed as follows.

1. Pick a permutation σ of $[d-1]$ that minimizes $\prod_{i=1}^{d-1} \Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i)$.

2. Apply the algorithm of Corollary 6 with this permutation σ to compute the updated inside-$\square_{\vec{v}}$ depth function $\delta^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$, which in turn specifies the updated set of type-2 intervals in $\mathcal{I}_{\vec{v}}$.
3. Delete the old type-2 intervals from $\mathbf{D}_{\vec{v}}$ and insert the new type-2 intervals into $\mathbf{D}_{\vec{v}}$.

We perform the above update for all $\vec{v} \in [r]^{d-1}$. Correctness follows from Observation 7 and the fact that $\mathbf{D}_{\vec{v}}$ always stores exactly the current interval multiset $\mathcal{I}_{\vec{v}}$. The main challenge is to prove that this procedure achieves the desired amortized update time $\widetilde{O}(n^{(d-1)/2})$.

5.2 Update time analysis

We show that each update of our data structure can be processed in $\widetilde{O}((n/r)^{d-1} + r^{d-1})$ time. Setting $r = \sqrt{n}$ then yields the desired bound $\widetilde{O}(n^{(d-1)/2})$.

Consider an update that inserts a new box B^* or deletes an existing box B^*. First observe that, for every $\vec{v} \in [r]^{d-1}$ such that $\mathcal{B}_{\vec{v}}$ does not change, the corresponding local update costs only $\widetilde{O}(1)$ time (it consists of maintaining counters and possibly inserting/deleting a single type-1 interval). In total, this contributes $\widetilde{O}(r^{d-1})$ time. It therefore suffices to focus on those $\vec{v} \in [r]^{d-1}$ for which $\mathcal{B}_{\vec{v}}$ changes; let $V^* \subseteq [r]^{d-1}$ denote this set. Note that V^* consists exactly of the vectors $\vec{v}$ for which $\square_{\vec{v}}$ intersects the boundary of $\mathbf{proj}_{[d-1]}(B^*)$. For $\vec{v} \in V^*$, all steps of the update can still be performed in $\widetilde{O}(1)$ time except: (i) recomputing $\delta^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$ via Corollary 6, and (ii) deleting the old type-2 intervals from $\mathbf{D}_{\vec{v}}$ and inserting the new ones. Since $\delta^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$ has $O(|\mathcal{B}_{\vec{v}}|)$ pieces, step (ii) costs $\widetilde{O}(|\mathcal{B}_{\vec{v}}|)$ time, which is dominated (up to logarithmic factors) by the cost of step (i). Thus, it suffices to bound the total cost of computing $\delta^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$ for all $\vec{v} \in V^*$.

Let $\mathcal{B}$ denote the set of boxes *after* the insertion/deletion of B^*. For $i \in [d-1]$ and $a \in [r]$, define

$$V_i(a) := \{(a_1, \ldots, a_{d-1}) \in [r]^{d-1} : a_i = a\}.$$

Intuitively, the cells $\square_{\vec{v}} \times \mathbb{R}$ with $\vec{v} \in V_i(a)$ are precisely those whose i-th coordinate is a. Define $\gamma_i(a) := \sum_{\vec{v} \in V_i(a)} \big|\mathsf{cor}(\mathcal{B}) \cap (\square_{\vec{v}} \times \mathbb{R})\big|$, i.e., the number of corners of boxes in $\mathcal{B}$ lying in these cells. For $i, j \in [d-1]$ and $a, b \in [r]$, define $V_{i,j}(a,b)$ and $\gamma_{i,j}(a,b)$ similarly. The following basic properties are immediate from the construction.

▶ **Fact 8.** *The following three properties hold.*

(i) $\gamma_i(a) = O(n/r)$ *for all* $i \in [d-1]$ *and* $a \in [r]$.

(ii) $\sum_{a=1}^{r} \gamma_{i,j}(a,b) = \gamma_j(b)$ *for all* $i, j \in [d-1]$ *and* $b \in [r]$.

(iii) *For* $\vec{v} = (a_1, \ldots, a_{d-1}) \in [r]^{d-1}$, $|\mathcal{B}_{\vec{v}}| \le \sum_{i=1}^{d-1} \gamma_i(a_i)$.

Combining Item 1 and Item 3 yields $|\mathcal{B}_{\vec{v}}| = O(n/r)$ for all $\vec{v} \in [r]^{d-1}$, and in particular for all $\vec{v} \in V^*$. By Corollary 6, computing $\delta^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$ takes time

$$O\left(|\mathcal{B}_{\vec{v}}| \log |\mathcal{B}_{\vec{v}}| \cdot \min_{\sigma} \prod_{i=1}^{d-1} \Phi_{\sigma, \square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}_{\vec{v}}), i)\right),$$

where the minimum is over permutations σ of $[d-1]$. Thus, the remaining difficulty is to bound the multiplicative factor $\min_\sigma \prod_{i=1}^{d-1} \Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}_{\vec{v}}), i)$ in aggregate over $\vec{v} \in V^*$. A per-cell bound is too weak; instead we carry out a global analysis. The key is Lemma 11, which is based on the following relation between the Φ-values and the γ-values.

▶ **Lemma 9.** *Let $\vec{v} = (a_1, \ldots, a_{d-1}) \in [r]^{d-1}$ and σ be a permutation of $[d-1]$. Then for any box B in $\mathbb{R}^d$ and any $i \in [d-1]$, if $\phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(B), i) = 1$, then there exist $j \in [d-1]$ with $j <_\sigma i$ and a vector $\vec{u} \in V_{i,j}(a_i, a_j)$ satisfying that $\mathsf{cor}(B) \cap (\square_{\vec{u}} \times \mathbb{R}) \neq \emptyset$.*

Proof. Assume $\phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(B), i) = 1$. By definition of ϕ, $\mathbf{proj}_{[d-1]}(B)$ is nontrivial in dimension i with respect to $\square_{\vec{v}}$, and there exists $j <_\sigma i$ such that $\mathbf{proj}_{[d-1]}(B)$ is also nontrivial in dimension j with respect to $\square_{\vec{v}}$. Therefore, there is a facet F_i (resp., F_j) of $\mathbf{proj}_{[d-1]}(B)$ perpendicular to the x_i-axis (resp., x_j-axis) that intersects $\square_{\vec{v}}$. Let F_i' and F_j' be the corresponding facets of B. Then both F_i' and F_j' intersect the cell $\square_{\vec{v}} \times \mathbb{R}$. Since $i \neq j$, there is a corner $z \in \mathsf{cor}(B)$ lying on both F_i' and F_j'. Suppose $z \in \square_{\vec{u}} \times \mathbb{R}$ for $\vec{u} = (b_1, \ldots, b_{d-1}) \in [r]^{d-1}$. Then $\mathsf{cor}(B) \cap (\square_{\vec{u}} \times \mathbb{R}) \neq \emptyset$. Moreover, because $z \in F_i'$ and F_i' is perpendicular to the x_i-axis while intersecting $\square_{\vec{v}} \times \mathbb{R}$, the i-th coordinate slab of z equals that of $\square_{\vec{v}} \times \mathbb{R}$, i.e., $b_i = a_i$. Similarly, $b_j = a_j$. Hence $\vec{u} \in V_{i,j}(a_i, a_j)$. ◀

▶ **Corollary 10.** *Let $\vec{v} = (a_1, \ldots, a_{d-1}) \in [r]^{d-1}$ and σ be a permutation of $[d-1]$. Then for any $i \in [d-1]$, we have the inequality*

$$\Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i) - 1 \;\leq \sum_{j \in [d-1], j <_\sigma i} \gamma_{i,j}(a_i, a_j).$$

Proof. By definition, $\Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i) - 1 = \sum_{B \in \mathcal{B}} \phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(B), i)$. For a $B \in \mathcal{B}$ such that $\phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(B), i) = 1$, Lemma 9 implies that there exists $j <_\sigma i$ and some $\vec{u} \in V_{i,j}(a_i, a_j)$ such that $\mathsf{cor}(B) \cap (\square_{\vec{u}} \times \mathbb{R}) \neq \emptyset$. Consequently,

$$\phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(B), i) \leq \sum_{j \in [d-1], j <_\sigma i} \; \sum_{\vec{u} \in V_{i,j}(a_i, a_j)} \big|\mathsf{cor}(B) \cap (\square_{\vec{u}} \times \mathbb{R})\big|.$$

Summing this inequality over all $B \in \mathcal{B}$ yields

$$\begin{aligned}
\Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i) - 1 &\leq \sum_{B \in \mathcal{B}} \sum_{j \in [d-1], j <_\sigma i} \; \sum_{\vec{u} \in V_{i,j}(a_i, a_j)} \big|\mathsf{cor}(B) \cap (\square_{\vec{u}} \times \mathbb{R})\big| \\
&= \sum_{j \in [d-1], j <_\sigma i} \; \sum_{\vec{u} \in V_{i,j}(a_i, a_j)} \big|\mathsf{cor}(\mathcal{B}) \cap (\square_{\vec{u}} \times \mathbb{R})\big| \\
&= \sum_{j \in [d-1], j <_\sigma i} \gamma_{i,j}(a_i, a_j).
\end{aligned}$$ ◀

We are now ready to prove the key lemma. Roughly speaking, it bounds the total "Φ-product complexity" over all cells in a fixed tube $V_t(a)$, provided the permutation σ starts with t.

▶ **Lemma 11.** *Let $t \in [d-1]$ and $a \in [r]$. If σ is a permutation of $[d-1]$ whose first element is t, then we have $\sum_{\vec{v} \in V_t(a)} \prod_{i=1}^{d-1} \Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}_{\vec{v}}), i) = O((n/r + r)^{d-2})$.*

Proof. It suffices to prove $\sum_{\vec{v} \in V_t(a)} \prod_{i=1}^{d-1} \Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i) = O((n/r+r)^{d-2})$. Without loss of generality, assume $t = 1$ and $\sigma = (1, \ldots, d-1)$. Then $V_t(a)$ consists of the vectors $(a_1, \ldots, a_{d-1}) \in [r]^{d-1}$ with $a_1 = a$ and $a_2, \ldots, a_{d-1} \in [r]$. By Corollary 10,

$$\begin{aligned}
\sum_{\vec{v} \in V_t(a)} \prod_{i=1}^{d-1} \Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i) &\leq \sum_{a_1 = a}^{a} \sum_{a_2 = 1}^{r} \cdots \sum_{a_{d-1} = 1}^{r} \prod_{i=1}^{d-1} \left(1 + \sum_{j=1}^{i-1} \gamma_{i,j}(a_i, a_j)\right) \\
&\leq \sum_{a_1 = a}^{a} \sum_{a_2 = 1}^{r} \cdots \sum_{a_{d-1} = 1}^{r} \prod_{i=2}^{d-1} \left(\sum_{j=1}^{i-1} (\gamma_{i,j}(a_i, a_j) + 1)\right).
\end{aligned}$$

Define $Q := \{(j_2, \ldots, j_{d-1}) \in [d-1]^{d-2} : j_i < i \text{ for all } i \in \{2, \ldots, d-1\}\}$. Then in the above inequality, $\prod_{i=2}^{d-1} \left(\sum_{j=1}^{i-1} (\gamma_{i,j}(a_i, a_j) + 1)\right) = \sum_{(j_2,\ldots,j_{d-1})\in Q} \prod_{i=2}^{d-1} (\gamma_{i,j_i}(a_i, a_{j_i}) + 1)$, and therefore

$$\sum_{\vec{v}\in V_t(a)} \prod_{i=1}^{d-1} \Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i) \leq \sum_{(j_2,\ldots,j_{d-1})\in Q} \sum_{a_1=a}^{a} \sum_{a_2=1}^{r} \cdots \sum_{a_{d-1}=1}^{r} \prod_{i=2}^{d-1} (\gamma_{i,j_i}(a_i, a_{j_i}) + 1).$$

Since $|Q| = O(1)$ (as d is constant), it suffices to show that for every fixed $(j_2, \ldots, j_{d-1}) \in Q$, the left sum-of-products is bounded by $O((n/r + r)^{d-2})$. For each $k \in [d-1]$, define

$$S_k := \sum_{a_1=a}^{a} \sum_{a_2=1}^{r} \cdots \sum_{a_k=1}^{r} \prod_{i=2}^{k} (\gamma_{i,j_i}(a_i, a_{j_i}) + 1).$$

This is well-defined because $j_i < i$ for all i, and thus all indices appearing in the product are at most $k-1$. We prove by induction that $S_k = O((n/r + r)^{k-1})$ for all $k \in [d-1]$, which immediately gives the desired bound for $S_{d-1} = O((n/r + r)^{d-2})$.

For the base case $k = 1$, we have $S_1 = \sum_{a_1=a}^{a} \prod_{i=2}^{1} \gamma'_{i,j_i}(a_i, a_{j_i}) = O(1)$. Assume $S_{k-1} = O((n/r + r)^{k-2})$ for some $k \geq 2$. In the product $\prod_{i=2}^{k} (\gamma_{i,j_i}(a_i, a_{j_i}) + 1)$, all factors with $i \leq k-1$ are independent of a_k, since $j_i < i \leq k-1$. Hence,

$$\begin{aligned} S_k &= \sum_{a_1=a}^{a} \sum_{a_2=1}^{r} \cdots \sum_{a_{k-1}=1}^{r} \left(\prod_{i=2}^{k-1} (\gamma_{i,j_i}(a_i, a_{j_i}) + 1) \cdot \sum_{a_k=1}^{r} (\gamma_{k,j_k}(a_k, a_{j_k}) + 1) \right) \\ &= \sum_{a_1=a}^{a} \sum_{a_2=1}^{r} \cdots \sum_{a_{k-1}=1}^{r} \left(\prod_{i=2}^{k-1} (\gamma_{i,j_i}(a_i, a_{j_i}) + 1) \cdot \left(r + \sum_{a_k=1}^{r} \gamma_{k,j_k}(a_k, a_{j_k}) \right) \right). \end{aligned}$$

By Item 2 of Fact 8, $\sum_{a_k=1}^{r} \gamma_{k,j_k}(a_k, a_{j_k}) = \gamma_{j_k}(a_{j_k})$, and by Item 1 of Fact 8 this is $O(n/r)$ for every value of a_{j_k}. Therefore,

$$S_k = O(n/r + r) \cdot \sum_{a_1=a}^{a} \sum_{a_2=1}^{r} \cdots \sum_{a_{k-1}=1}^{r} \prod_{i=2}^{k-1} (\gamma_{i,j_i}(a_i, a_{j_i}) + 1) = O((n/r + r) \cdot S_{k-1}),$$

which yields $S_k = O((n/r + r)^{k-1})$ and completes the proof. ◀

▶ **Corollary 12.** *$\sum_{\vec{v}\in V^*} T_{\vec{v}} = \widetilde{O}((n/r)^{d-1} + nr^{d-3})$, where $T_{\vec{v}}$ denotes the time cost of computing $\delta_{\square_{\vec{v}}}^{\mathcal{B}_{\vec{v}}}$.*

Proof. Recall that V^* consists of all $\vec{v} \in [r]^{d-1}$ such that $\square_{\vec{v}}$ intersects the boundary of $\mathbf{proj}_{[d-1]}(B^*)$. Equivalently, $\square_{\vec{v}}$ intersects one of the $2(d-1)$ facets of $\mathbf{proj}_{[d-1]}(B^*)$. Let $F_1, \ldots, F_{d-1}, F'_1, \ldots, F'_{d-1}$ be these facets, where F_i, F'_i are perpendicular to the x_i-axis. For each $i \in [d-1]$, there exist indices $a_i, a'_i \in [r]$ such that $F_i \subseteq \bigcup_{\vec{v}\in V_i(a_i)} \square_{\vec{v}}$ and $F'_i \subseteq \bigcup_{\vec{v}\in V_i(a'_i)} \square_{\vec{v}}$. Hence, $V^* \subseteq \bigcup_{i=1}^{d-1} \big(V_i(a_i) \cup V_i(a'_i)\big)$ and

$$\sum_{\vec{v}\in V^*} T_{\vec{v}} \leq \sum_{i=1}^{d-1} \left(\sum_{\vec{v}\in V_i(a_i)} T_{\vec{v}} + \sum_{\vec{v}\in V_i(a'_i)} T_{\vec{v}} \right).$$

Combining Corollary 6 and Fact 8, we have for every $\vec{v} \in V^*$ and every permutation σ of $[d-1]$,

$$T_{\vec{v}} = \widetilde{O}\left(\frac{n}{r} \cdot \prod_{i=1}^{d-1} \Phi_{\sigma,\square_{\vec{v}}}(\mathbf{proj}_{[d-1]}(\mathcal{B}_{\vec{v}}), i) \right).$$

Fix $i \in [d-1]$ and choose a permutation σ whose first element is i. Applying Lemma 11 to the tube $V_i(a_i)$ (and similarly to $V_i(a_i')$) yields

$$\sum_{\vec{v} \in V_i(a_i)} T_{\vec{v}} = \widetilde{O}\left(\frac{n}{r} \cdot (n/r + r)^{d-2}\right), \qquad \sum_{\vec{v} \in V_i(a_i')} T_{\vec{v}} = \widetilde{O}\left(\frac{n}{r} \cdot (n/r + r)^{d-2}\right).$$

Since d is a constant, $n/r \cdot (n/r + r)^{d-2} = O((n/r)^{d-1} + nr^{d-3})$. Summing over all $i \in [d-1]$ (a constant number of terms) yields the desired bound. ◀

Combining Corollary 12 with the $\widetilde{O}(r^{d-1})$ cost for the cells outside V^* shows that the update time is $\widetilde{O}\big((n/r)^{d-1} + nr^{d-3} + r^{d-1}\big)$. Moreover, the middle term is always dominated by the other two terms: if $r \le \sqrt{n}$ then $nr^{d-3} \le (n/r)^{d-1}$, while if $r \ge \sqrt{n}$ then $nr^{d-3} \le r^{d-1}$. Therefore, the update time is $\widetilde{O}((n/r)^{d-1} + r^{d-1})$. Setting $r = \sqrt{n}$ yields $\widetilde{O}(n^{(d-1)/2})$, proving Theorem 1.

6 Adaptation to dynamic Klee's measure

In this section, we explain how to modify the data structure from the previous sections to maintain *Klee's measure* for axis-parallel boxes in the fully dynamic setting. The proofs of this section are defer to Appendix B.

6.1 Offline subroutine

We first modify the offline algorithm from Section 4. The only additional ingredient is the following analogue of Fact 3.

▶ **Fact 13.** *Let $\mathcal{L}$ be a set of slabs in $\mathbb{R}^c$ and $\square$ be a fixed box in $\mathbb{R}^c$. Then we have $1 - \mathsf{Klee}_{\square}(\mathcal{L})/V(\square) = \prod_{i=1}^{c}(1 - \mathsf{Klee}_{\square}(\mathcal{L}_i)/V(\square))$, where $\mathcal{L}_i \subseteq \mathcal{L}$ consists of the slabs bounded in dimension i and $V(\square)$ denotes the volume of $\square$.*

Fact 13 implies an analogue of Lemma 4, namely a dynamic inside-$\square$ Klee's measure data structure for slabs with $O(\log n)$ update time. Using this slab data structure in place of Lemma 4, we obtain an analogue of Lemma 5 for Offline Inside-$\square$ Box Klee's Measure with the same running-time bound.

To formulate the analogue of Corollary 6, we define the *inside-R measure function* of $\mathcal{B}$ as $\kappa_R^{\mathcal{B}} : \mathbb{R} \to \mathbb{R}$, where

$$\kappa_R^{\mathcal{B}}(p) := \mathsf{Klee}_{R \times \{p\}}(\mathcal{B}) \qquad \text{for all } p \in \mathbb{R}.$$

As in the maximum-depth setting, $\kappa_R^{\mathcal{B}}$ is a piecewise constant function whose breakpoints lie among the endpoints of the intervals in $\mathbf{proj}_d(\mathcal{B})$, and hence it has $O(|\mathcal{B}|)$ pieces. The following corollary follows by the same sweep-line reduction as in the proof of Corollary 6.

▶ **Corollary 14.** *Let σ be a permutation of $[d-1]$ and $\square$ be a fixed box in $\mathbb{R}^{d-1}$. Given a set $\mathcal{B}$ of n boxes in $\mathbb{R}^d$, one can use $O(n \log n \cdot \prod_{i=1}^{d-1} \Phi_{\sigma,\square}(\mathbf{proj}_{[d-1]}(\mathcal{B}), i))$ time to compute the inside-$\square$ measure function $\kappa_{\square}^{\mathcal{B}}$.*

6.2 Dynamic data structure

We now modify the dynamic data structure from Section 5. We use the same partition of $\mathbb{R}^d$ into r^{d-1} cells indexed by $\vec{v} \in [r]^{d-1}$, and we maintain $\mathsf{Klee}_{\square_{\vec{v}} \times \mathbb{R}}(\mathcal{B})$ for each cell individually. We follow the notation from Section 5.

Recall that for maximum depth we maintained, for each $\vec{v}$, a weighted interval multiset $\mathcal{I}_{\vec{v}}$ that encodes the inside-$\square_{\vec{v}}$ depth function. For Klee's measure, we define $\mathcal{I}_{\vec{v}}$ analogously, replacing the pieces of $\delta^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$ by the pieces of $\kappa^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$. However, the relation between $\mathsf{Klee}_{\square_{\vec{v}} \times \mathbb{R}}(\mathcal{B})$ and $\mathcal{I}_{\vec{v}}$ is slightly more involved. We therefore separate $\mathcal{I}_{\vec{v}}$ into two parts: let $\mathcal{I}'_{\vec{v}} \subseteq \mathcal{I}_{\vec{v}}$ be the set of (weight-1) intervals corresponding to boxes $B \in \mathcal{B}$ with $\square_{\vec{v}} \subseteq \mathbf{proj}_{[d-1]}(B)$, and let $\mathcal{I}''_{\vec{v}} \subseteq \mathcal{I}_{\vec{v}}$ be the set of intervals corresponding to the pieces of the function $\kappa^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$. For each $I \in \mathcal{I}''_{\vec{v}}$, we denote its weight by $w(I) := \kappa^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}(I)$. For a set $\mathcal{I}$ of intervals, we write $\bigcup \mathcal{I}$ for the union of the intervals in $\mathcal{I}$.

▶ **Fact 15.** $\mathsf{Klee}_{\square_{\vec{v}} \times \mathbb{R}}(\mathcal{B}) = \mathsf{Klee}(\mathcal{I}'_{\vec{v}}) \cdot V(\square_{\vec{v}}) + \sum_{I \in \mathcal{I}''_{\vec{v}}} V(I \setminus \bigcup \mathcal{I}'_{\vec{v}}) \cdot w(I)$.

To evaluate the right-hand side of Fact 15, we store $\mathcal{I}'_{\vec{v}}$ and $\mathcal{I}''_{\vec{v}}$ in two data structures $\mathbf{D}'_{\vec{v}}$ and $\mathbf{D}''_{\vec{v}}$. The data structure $\mathbf{D}'_{\vec{v}}$ maintains $\mathsf{Klee}(\mathcal{I}'_{\vec{v}})$ and also supports range queries for Klee's measure; this additional capability is needed when constructing and maintaining $\mathbf{D}''_{\vec{v}}$. The data structure $\mathbf{D}''_{\vec{v}}$ maintains the quantity $\sum_{I \in \mathcal{I}''_{\vec{v}}} V(I \setminus \bigcup \mathcal{I}'_{\vec{v}}) \cdot w(I)$.

Before defining $\mathbf{D}'_{\vec{v}}$, we introduce an auxiliary structure $\mathbf{A}$ on $\mathcal{I}'_{\vec{v}}$. Given a pair (L, R) of query intervals, $\mathbf{A}$ checks whether there exists an interval in $\mathcal{I}'_{\vec{v}}$ whose left endpoint lies in L and right endpoint lies in R. Such a structure can be obtained by mapping each interval in $\mathcal{I}'_{\vec{v}}$ to a point in $\mathbb{R}^2$ (left endpoint as the x-coordinate and right endpoint as the y-coordinate), reducing the query to dynamic 2D orthogonal range emptiness. Standard dynamic range-emptiness structures (e.g., dynamic 2D range trees) support both updates and queries in $\widetilde{O}(1)$ time, and we use $\mathbf{A}$ as such a black box.

Next, we define $\mathbf{D}'_{\vec{v}}$, a *dynamic range Klee's measure data structure* built on $\mathcal{I}'_{\vec{v}}$.

▶ **Definition 16.** *A* dynamic range Klee's measure data structure *stores a set $\mathcal{I}$ of intervals and supports the following three operations:*

- ***Insertion:*** *Insert a new interval into $\mathcal{I}$.*
- ***Deletion:*** *Delete an existing interval from $\mathcal{I}$.*
- ***Query:*** *For a query interval Q, return $\mathsf{Klee}_Q(\mathcal{I}_Q)$ where $\mathcal{I}_Q = \{I \in \mathcal{I} : Q \not\subseteq I\}$.*

▶ **Lemma 17.** *There exists a dynamic range Klee's measure data structure with $\widetilde{O}(1)$ amortized update time and $\widetilde{O}(1)$ query time.*

Let $\mathbf{D}'_{\vec{v}}$ be the data structure from Lemma 17 instantiated on $\mathcal{I}'_{\vec{v}}$. Querying $\mathbf{D}'_{\vec{v}}$ with $Q = (-\infty, +\infty)$ returns $\mathsf{Klee}(\mathcal{I}'_{\vec{v}})$.

We now define $\mathbf{D}''_{\vec{v}}$. It is an interval tree built on $\mathcal{I}''_{\vec{v}}$, whose stored values depend on both $\mathcal{I}''_{\vec{v}}$ and $\mathcal{I}'_{\vec{v}}$. The intervals in $\mathcal{I}''_{\vec{v}}$ form a partition of $\mathbb{R}$. The leaves of $\mathbf{D}''_{\vec{v}}$ correspond one-to-one to the intervals in $\mathcal{I}''_{\vec{v}}$, in left-to-right order. For a node t of $\mathbf{D}''_{\vec{v}}$, let $\mathcal{I}''_{\vec{v}}(t) \subseteq \mathcal{I}''_{\vec{v}}$ denote the set of leaf intervals in the subtree rooted at t, and define

$$\mathcal{I}'_{\vec{v}}(t) := \{I \in \mathcal{I}'_{\vec{v}} : \bigcup \mathcal{I}''_{\vec{v}}(t) \not\subseteq I\}.$$

At node t we store the value

$$\eta(t) := \sum_{I \in \mathcal{I}''_{\vec{v}}(t)} V(I \setminus \bigcup \mathcal{I}'_{\vec{v}}(t)) \cdot w(I).$$

If t has children s_1 and s_2, then $\mathcal{I}'_{\vec{v}}(t) \subseteq \mathcal{I}'_{\vec{v}}(s_1) \cap \mathcal{I}'_{\vec{v}}(s_2)$, and $\eta(t)$ can be computed from $\eta(s_1)$ and $\eta(s_2)$ by testing, for each child $s \in \{s_1, s_2\}$, whether $\mathcal{I}'_{\vec{v}}(s) = \mathcal{I}'_{\vec{v}}(t)$. We set $\eta'(s) = \eta(s)$ if $\mathcal{I}'_{\vec{v}}(s) = \mathcal{I}'_{\vec{v}}(t)$ and $\eta'(s) = 0$ otherwise, and then $\eta(t) = \eta'(s_1) + \eta'(s_2)$. Using $\mathbf{A}$, this equality test can be performed in $\widetilde{O}(1)$ time, and hence $\eta(t)$ can be computed in $\widetilde{O}(1)$ time given $\eta(s_1)$ and $\eta(s_2)$.

If t is a leaf corresponding to an interval $I \in \mathcal{I}''_{\vec{v}}$, then

$$\eta(t) = \big(V(I) - \mathsf{Klee}_I(\mathcal{I}'_{\vec{v}}(t))\big) \cdot w(I).$$

We obtain $\mathsf{Klee}_I(\mathcal{I}'_{\vec{v}}(t))$ by querying $\mathbf{D}'_{\vec{v}}$ with $Q = I$ in $\widetilde{O}(1)$ time, and then compute $\eta(t)$ in $\widetilde{O}(1)$ time. It follows that $\mathbf{D}''_{\vec{v}}$ can be built in $\widetilde{O}(|\mathcal{I}''_{\vec{v}}|)$ time by a bottom-up construction.

After an insertion or deletion in $\mathcal{I}'_{\vec{v}}$, we can update $\mathbf{D}''_{\vec{v}}$ in $\widetilde{O}(1)$ time. Consider an interval I inserted into $\mathcal{I}'_{\vec{v}}$ or deleted from $\mathcal{I}'_{\vec{v}}$. Let t_1 and t_2 be the two leaves of $\mathbf{D}''_{\vec{v}}$ whose corresponding intervals in $\mathcal{I}''_{\vec{v}}$ contain the endpoints of I. Let π_i be the path from the root to t_i, for $i \in \{1, 2\}$. Only nodes on $\pi_1 \cup \pi_2$ may change, and we can update them bottom-up, spending $\widetilde{O}(1)$ time per visited node. Since $|\pi_1|, |\pi_2| = \widetilde{O}(1)$, the total update time is $\widetilde{O}(1)$. We do not separately support insertions/deletions to $\mathcal{I}''_{\vec{v}}$: in our setting, whenever $\mathcal{I}''_{\vec{v}}$ changes, it changes globally, and we rebuild $\mathbf{D}''_{\vec{v}}$ from scratch on the new $\mathcal{I}''_{\vec{v}}$.

At this point, $\mathbf{D}'_{\vec{v}}$ provides $\mathsf{Klee}(\mathcal{I}'_{\vec{v}})$, and the root of $\mathbf{D}''_{\vec{v}}$ stores $\sum_{I \in \mathcal{I}''_{\vec{v}}} V(I \setminus \bigcup \mathcal{I}'_{\vec{v}}) \cdot w(I)$. Applying Fact 15 yields $\mathsf{Klee}_{\square_{\vec{v}} \times \mathbb{R}}(\mathcal{B})$.

Update time

Finally, we consider the update time for a fixed cell $\square_{\vec{v}} \times \mathbb{R}$ when a box B is inserted into $\mathcal{B}$ or deleted from $\mathcal{B}$. If $\square_{\vec{v}} \subseteq \mathbf{proj}_{[d-1]}(B)$, then we insert into $\mathcal{I}'_{\vec{v}}$ (or delete from $\mathcal{I}'_{\vec{v}}$) the interval $\mathbf{proj}_d(B)$. In this case we update $\mathbf{A}$ and $\mathbf{D}'_{\vec{v}}$, and then update $\mathbf{D}''_{\vec{v}}$ as described above; the total cost is $\widetilde{O}(1)$.

Otherwise, we recompute the inside-$\square_{\vec{v}}$ measure function $\kappa^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$ and rebuild $\mathbf{D}''_{\vec{v}}$ on the resulting set $\mathcal{I}''_{\vec{v}}$ of pieces. By Corollary 14, the cost of recomputing $\kappa^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$ matches (up to logarithmic factors) the cost of recomputing $\delta^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$ in the maximum-depth structure. Moreover, rebuilding $\mathbf{D}''_{\vec{v}}$ costs $\widetilde{O}(|\mathcal{I}''_{\vec{v}}|) = \widetilde{O}(|\mathcal{B}_{\vec{v}}|)$, since the number of pieces of $\kappa^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$ is $O(|\mathcal{B}_{\vec{v}}|)$. Therefore, the per-cell update time is the same as in the maximum-depth setting up to logarithmic factors. Applying the same global analysis as in Section 5 yields an overall amortized update time $\widetilde{O}(n^{(d-1)/2})$, proving Theorem 2.

7 Conclusion and open questions

In this paper, we develop fully dynamic data structures for maintaining the maximum depth and Klee's measure of a set of axis-parallel boxes in any constant dimension d, achieving a near-optimal amortized update time of $\widetilde{O}(n^{(d-1)/2})$. While this update time is close to the conjectured lower bound, several directions remain open for future investigation.

A first and perhaps most natural question is whether one can *eliminate the logarithmic factors* in the update time, mirroring the historical progression of static Klee's measure algorithms [29, 12, 13]. A second direction is to improve the preprocessing time and the space complexity. It is plausible that the preprocessing can be reduced to $O(n^{d/2})$ time, whereas our current preprocessing and space usage is comparatively brute-force and requires $\widetilde{O}(n^{(d+1)/2})$ time and space.

Our results address the *exact* problems in the *fully dynamic* setting, but it is also natural to study relaxations. For instance, one may ask whether maintaining a $(1-\varepsilon)$-approximation to the maximum depth or to Klee's measure admits asymptotically faster updates. Another

variant is the *semi-dynamic* setting, where updates are restricted to insertions only or deletions only. It remains open whether these relaxations allow substantially smaller update times.

Since this work focuses on boxes, an obvious next step is to design dynamic maximum-depth data structures for other object families, such as balls, simplices, or more general convex bodies. These extensions appear considerably more challenging, as the maximum-depth problem is already known to be 3SUM-hard even for disks [4].

Finally, from a technical perspective, it is also interesting to explore whether our charging analysis can be applied to other partitioning-based problems involving boxes.

References

1 Peyman Afshani and Timothy M. Chan. On approximate range counting and depth. *Discrete Comput. Geom.*, 42(1):3–21, 2009. doi:10.1007/S00454-009-9177-Z.

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

3 Helmut Alt and Ludmila Scharf. Computing the depth of an arrangement of axis-aligned rectangles in parallel. In *Abstr. 26th Eur. Workshop Comput. Geom. (EuroCG)*, pages 33–36, 2010.

4 Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008. doi:10.1137/060669474.

5 Jérémy Barbay, Timothy M. Chan, Gonzalo Navarro, and Pablo Pérez-Lantero. Maximum-weight planar boxes in time $O(n^2)$ (and better). *Inf. Process. Lett.*, 114(8):437–445, 2014.

6 Jon Louis Bentley. Algorithms for klee's rectangle problem. *Unpublished manuscript*, 1977.

7 Sujoy Bhore and Timothy M. Chan. Dynamic independent set of disks (and hypercubes) made easier. In *Proc. 8th Sympos. Simplicity in Algorithms (SOSA)*, pages 485–495, 2025. doi:10.1137/1.9781611978315.36.

8 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. 36th ACM-SIAM Sympos. Discrete Algs. (SODA)*, pages 2357–2386, 2025. doi:10.1137/1.9781611978322.79.

9 Sujoy Bhore, Martin Nöllenburg, Csaba D. Tóth, and Jules Wulms. Fully dynamic maximum independent sets of disks in polylogarithmic update time. In *Proc. 40th Int. Annu. Sympos. Comput. Geom. (SoCG)*, pages 19:1–19:16, 2024. doi:10.4230/LIPIcs.SOCG.2024.19.

10 Timothy M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. *J. ACM*, 48(1):1–12, 2001. doi:10.1145/363647.363652.

11 Timothy M. Chan. Dynamic coresets. *Discrete Comput. Geom.*, 42(3):469–488, 2009. doi:10.1007/S00454-009-9165-3.

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

13 Timothy M. Chan. Klee's measure problem made easy. In *Proc. 54th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 410–419, 2013. doi:10.1109/FOCS.2013.51.

14 Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. *Discrete Comput. Geom.*, 64(4):1235–1252, 2020. doi:10.1007/S00454-020-00229-5.

15 Timothy M. Chan and Qizheng He. More dynamic data structures for geometric set cover with sublinear update time. *J. Comput. Geom.*, 13(2):90–114, 2021. doi:10.20382/JOCG.V13I2A6.

16 Timothy M. Chan, Qizheng He, Subhash Suri, and Jie Xue. Dynamic geometric set cover, revisited. In *Proc. 33rd ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 3496–3528, 2022. doi:10.1137/1.9781611977073.139.

17 Timothy M. Chan and Konstantinos Tsakalidis. Dynamic orthogonal range searching on the RAM, revisited. *J. Comput. Geom.*, 9(2):45–66, 2018. doi:10.20382/JOCG.V9I2A5.

18 Bernard Marie Chazelle and D. T. Lee. On a circle placement problem. *Computing*, 36(1-2):1–16, 1986. doi:10.1007/BF02238188.

19 Ludwig Danzer, Branko Grünbaum, and Victor Klee. Helly's theorem and its relatives. In *Convexity, Proc. Sympos. Pure Math., Vol. 7, Amer. Math. Soc.*, volume 7, page 101, 1963.

20 David Dobkin and Subhash Suri. Maintenance of geometric extrema. *J. ACM*, 38(2):275–298, 1991. doi:10.1145/103516.103518.

21 D. Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *Discrete Comput. Geom.*, 13(1):111–122, 1995. doi:10.1007/BF02574030.

22 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In *Proc. 36th Int. Sympos. Comput. Geom. (SoCG)*, pages 51:1–51:14, 2020. doi:10.4230/LIPIcs.SOCG.2020.51.

23 John Hershberger and Subhash Suri. Off-line maintenance of planar configurations. *J. Algorithms*, 21(3):453–475, 1996. doi:10.1006/JAGM.1996.0054.

24 Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms*, 4(4):310–323, 1983. doi:10.1016/0196-6774(83)90012-3.

25 Victor Klee. Can the measure of $\bigcup_{i=1}^{n}[a_i, b_i]$ be computed in less than $o(n \log n)$ steps? *Amer. Math. Monthly*, 84(4):284–285, 1977.

26 Jiri Matousek. *Lectures on discrete geometry*, volume 212. Springer Science & Business Media, 2013.

27 Christian Worm Mortensen. Fully dynamic orthogonal range reporting on ram. *SIAM J. Comput.*, 35(6):1494–1525, 2006. doi:10.1137/S0097539703436722.

28 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981. doi:10.1016/0022-0000(81)90012-X.

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

30 Subhash Suri, Jie Xue, Xiongxin Yang, and Jiumu Zhu. Dynamic maximum depth of geometric objects. In *Proc. 41st Int. Annu. Sympos. Comput. Geom. (SoCG)*, pages 77:1–77:13, 2025. doi:10.4230/LIPIcs.SOCG.2025.77.

31 Jan van Leeuwen and Derick Wood. The measure problem for rectangular ranges in d-space. *J. Algorithms*, 2(3):282–300, 1981. doi:10.1016/0196-6774(81)90027-4.

A Low-level implementation details

In this section, we provide the implementation details omitted in Section 5, namely periodic reconstruction and the splitting procedure.

A.1 Reconstruction

Our data structure uses the standard technique of periodic reconstruction, as in [30]. After the i-th reconstruction, let $n_i := |\mathcal{B}|$ denote the number of boxes at that time. We then perform $n_i/2$ update operations before initiating the $(i+1)$-st reconstruction. During this i-th period, i.e., the interval between the i-th and $(i+1)$-st reconstructions, we therefore have $|\mathcal{B}| = \Theta(n_i)$.

Since our data structure supports updates in $O(n^{(d-1)/2} \log n)$ time on a set of size n, we can rebuild it from scratch in $O(n^{(d+1)/2} \log n)$ time by inserting the n boxes one by one. Thus, the $(i+1)$-st reconstruction takes $O(n_i^{(d+1)/2} \log n_i)$ time. Amortizing this cost over the $n_i/2$ operations in the i-th period yields an amortized reconstruction cost of $O(n_i^{(d-1)/2} \log n_i)$ per operation, which keeps the amortized update time at $O(n^{(d-1)/2} \log n)$.

A.2 Splitting

We now describe how to maintain the invariant assumed in Section 5: for each $i \in [d-1]$, the number of corners of boxes in $\mathcal{B}$ lying between any two consecutive hyperplanes in $\mathcal{H}_i$ is always $O(n/r)$.

Fix $i \in [d-1]$ and consider a reconstruction period in which the set size satisfies $|\mathcal{B}| = \Theta(n)$ (where n denotes the size at the beginning of the period). The hyperplanes in $\mathcal{H}_i$ partition $\mathbb{R}^d$ into *slabs* along the x_i-axis. Whenever the number of corners of boxes in $\mathcal{B}$ in the slab defined by two consecutive hyperplanes $H_{a_i-1}, H_{a_i} \in \mathcal{H}_i$ reaches $2^{d+1}n/r$, we insert a new hyperplane H' perpendicular to the x_i-axis between H_{a_i-1} and H_{a_i} so that the numbers of corners in the two new slabs (between H_{a_i-1} and H' and between H' and H_{a_i}) are both exactly $2^d n/r$. Consequently, every cell $\square_{\vec{v}} \times \mathbb{R}$ whose i-th coordinate equals a_i is split into two cells $\square' \times \mathbb{R}$ and $\square'' \times \mathbb{R}$.

By construction, immediately after the split, each of the two new slabs contains at most $2^d n/r$ corners, and a slab is split only once it accumulates $2^{d+1}n/r$ corners. Thus, at all times, the number of corners between any two consecutive hyperplanes in $\mathcal{H}_i$ is at most $2^{d+1}n/r = O(n/r)$.

On the other hand, to make the proof in Section 5 go through, we also need to ensure that the total number of cells remains $O(r^{d-1})$ despite splittings. This is guaranteed by the following fact, which is a direct generalization of Fact 4 in [30].

▶ **Fact 18.** *Over the first t update operations within a reconstruction period (with parameter n fixed at the beginning of the period), the number of splittings along any fixed axis is at most $2tr/n$.*

Proof. Since deletions do not trigger splitting, it suffices to account for insertions among the t operations. Fix an axis $i \in [d-1]$. For each inserted box B, we charge it to the (at most) two slabs along the x_i-axis, defined by consecutive hyperplanes in $\mathcal{H}_i$, that contain corners of B at the time of insertion.

Each such charge corresponds to inserting at most 2^d new corners into the charged slab. Therefore, a slab S must receive at least n/r charges before its number of corners grows from at most $2^d n/r$ to $2^{d+1}n/r$, at which point S is split. Equivalently, every split of a slab S is preceded by at least n/r charges to S. Since each insertion contributes at most two charges along the x_i-axis, the total number of charges along this axis over the first t operations is at most $2t$. It follows that the number of splits along the x_i-axis is at most $\frac{2t}{n/r} = \frac{2tr}{n}$, as claimed. ◀

▶ **Corollary 19.** *Despite splittings, the number of cells remains $O(r^{d-1})$.*

Proof. Consider an arbitrary reconstruction period. By construction, the period contains at most $n/2$ update operations. By Fact 18, the number of slab splittings along each axis is at most

$$\frac{2 \cdot (n/2) \cdot r}{n} = r.$$

Since we start the period with r slabs along each axis, after at most r additional splits the number of slabs along each axis is at most $r + r$. Therefore, the total number of cells is at most $(r+r)^{d-1} = O(r^{d-1})$. ◀

B Missing proofs

B.1 Proof of Fact 8

Proof. Item 1 follows directly from the construction (and maintenance) of $\mathcal{H}_i$: for each $i \in [d-1]$ and each $a \in [r]$, the cells $\{\square_{\vec{v}} \times \mathbb{R} : \vec{v} \in V_i(a)\}$ form exactly the slab between two consecutive hyperplanes in $\mathcal{H}_i$, and by invariant this slab contains $O(n/r)$ corners of boxes in $\mathcal{B}$. Thus $\gamma_i(a) = O(n/r)$.

Item 2 is immediate from the definitions. Fix $i, j \in [d-1]$ and $b \in [r]$. The sets $V_{i,j}(a,b)$ partition $V_j(b)$ as a ranges over $[r]$, and therefore

$$\sum_{a=1}^{r} \gamma_{i,j}(a,b) = \sum_{a=1}^{r} \sum_{\vec{v} \in V_{i,j}(a,b)} \left|\mathsf{cor}(\mathcal{B}) \cap (\square_{\vec{v}} \times \mathbb{R})\right| = \sum_{\vec{v} \in V_j(b)} \left|\mathsf{cor}(\mathcal{B}) \cap (\square_{\vec{v}} \times \mathbb{R})\right| = \gamma_j(b).$$

For Item 3, fix $\vec{v} = (a_1, \ldots, a_{d-1}) \in [r]^{d-1}$ and consider any box $B \in \mathcal{B}_{\vec{v}}$. By definition of $\mathcal{B}_{\vec{v}}$, we have $\mathbf{proj}_{[d-1]}(B) \cap \square_{\vec{v}} \neq \emptyset$ and $\square_{\vec{v}} \not\subseteq \mathbf{proj}_{[d-1]}(B)$. Hence there exists a dimension $i \in [d-1]$ such that $\mathbf{proj}_i(\square_{\vec{v}}) \not\subseteq \mathbf{proj}_i(B)$. Since $\mathbf{proj}_i(\square_{\vec{v}})$ intersects $\mathbf{proj}_i(B)$, at least one endpoint of $\mathbf{proj}_i(B)$ lies in the interior of $\mathbf{proj}_i(\square_{\vec{v}})$. Equivalently, there is a facet F of $\mathbf{proj}_{[d-1]}(B)$ perpendicular to the x_i-axis that intersects $\square_{\vec{v}}$.

Let F' be the corresponding facet of B in $\mathbb{R}^d$, and let $z \in \mathsf{cor}(B)$ be any corner incident to F'. Then z lies in the slab with i-th coordinate equal to a_i, and therefore $z \in \square_{\vec{u}} \times \mathbb{R}$ for some $\vec{u} \in V_i(a_i)$. We charge the box B to this corner z. Doing this for every $B \in \mathcal{B}_{\vec{v}}$ yields

$$|\mathcal{B}_{\vec{v}}| \leq \sum_{i=1}^{d-1} \sum_{\vec{u} \in V_i(a_i)} \left|\mathsf{cor}(\mathcal{B}) \cap (\square_{\vec{u}} \times \mathbb{R})\right| = \sum_{i=1}^{d-1} \gamma_i(a_i),$$

which is exactly Item 3. ◀

B.2 Proof of Fact 13

Fix $i \in [c]$. Every slab $L \in \mathcal{L}_i$ is unbounded in all coordinates except the i-th one. Let $U_i := \bigcup_{L \in \mathcal{L}_i} \mathbf{proj}_i(L \cap \square) \subseteq \mathbf{proj}_i(\square)$. Then $\bigcup_{L \in \mathcal{L}_i} (L \cap \square) = \prod_{j=1}^{i-1} \mathbf{proj}_j(\square) \times U_i \times \prod_{j=i+1}^{c} \mathbf{proj}_j(\square)$, and therefore

$$\mathsf{Klee}_{\square}(\mathcal{L}_i) = V\left(\bigcup_{L \in \mathcal{L}_i} (L \cap \square)\right) = |U_i| \cdot \prod_{j \neq i} |\mathbf{proj}_j(\square)|.$$

Dividing by $V(\square) = \prod_{j=1}^{c} |\mathbf{proj}_j(\square)|$ gives

$$1 - \frac{\mathsf{Klee}_{\square}(\mathcal{L}_i)}{V(\square)} = 1 - \frac{|U_i|}{|\mathbf{proj}_i(\square)|} = \frac{|\mathbf{proj}_i(\square)| - |U_i|}{|\mathbf{proj}_i(\square)|}.$$

Now consider the full slab set $\mathcal{L} = \bigcup_{i=1}^{c} \mathcal{L}_i$. Inside $\square$, the complement of the union is

$$\begin{aligned} \square \setminus \bigcup_{L \in \mathcal{L}} (L \cap \square) &= \square \cap \bigcap_{i=1}^{c} \left(\square \setminus \bigcup_{L \in \mathcal{L}_i} (L \cap \square)\right) \\ &= \bigcap_{i=1}^{c} \left(\prod_{j \neq i} \mathbf{proj}_j(\square) \times \left(\mathbf{proj}_i(\square) \setminus U_i\right)\right) = \prod_{i=1}^{c} \left(\mathbf{proj}_i(\square) \setminus U_i\right). \end{aligned}$$

Taking volumes yields $V(\square) - \mathsf{Klee}_{\square}(\mathcal{L}) = \prod_{i=1}^{c} \big(|\mathbf{proj}_i(\square)| - |U_i|\big)$. Dividing by $V(\square)$ and substituting the expression for each $\mathcal{L}_i$ gives

$$1 - \frac{\mathsf{Klee}_{\square}(\mathcal{L})}{V(\square)} = \prod_{i=1}^{c} \frac{|\mathbf{proj}_i(\square)| - |U_i|}{|\mathbf{proj}_i(\square)|} = \prod_{i=1}^{c} \left(1 - \frac{\mathsf{Klee}_{\square}(\mathcal{L}_i)}{V(\square)}\right).$$

B.3 Proof of Fact 15

Proof. Let $U := \bigcup \mathcal{I}'_{\vec{v}} \subseteq \mathbb{R}$ be the union of the type-1 intervals. For each $p \in \mathbb{R}$, define the cross-section value

$$m(p) := \mathsf{Klee}_{\square_{\vec{v}} \times \{p\}}(\mathcal{B}) = V\left(\left(\bigcup_{B \in \mathcal{B}} B\right) \cap (\square_{\vec{v}} \times \{p\})\right).$$

By Fubini's theorem (or equivalently by additivity of volume along the x_d-axis),

$$\mathsf{Klee}_{\square_{\vec{v}} \times \mathbb{R}}(\mathcal{B}) = \int_{\mathbb{R}} m(p)\, dp.$$

If $p \in U$, then there exists a box $B \in \mathcal{B}$ such that $\square_{\vec{v}} \subseteq \mathbf{proj}_{[d-1]}(B)$ and $p \in \mathbf{proj}_d(B)$, which implies $\square_{\vec{v}} \times \{p\} \subseteq B$ and hence $m(p) = V(\square_{\vec{v}})$. If $p \notin U$, then no such box exists, and the only boxes contributing to the cross section inside $\square_{\vec{v}} \times \{p\}$ are those in $\mathcal{B}_{\vec{v}}$. By definition of the inside-$\square_{\vec{v}}$ measure function, we then have $m(p) = \kappa^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}(p)$. Since $\mathcal{I}''_{\vec{v}}$ is precisely the set of pieces of $\kappa^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}$, for each $I \in \mathcal{I}''_{\vec{v}}$ the function equals the constant $w(I)$ on I.

Therefore,

$$\begin{aligned}
\mathsf{Klee}_{\square_{\vec{v}} \times \mathbb{R}}(\mathcal{B}) &= \int_{p \in U} V(\square_{\vec{v}})\, dp + \int_{p \notin U} \kappa^{\mathcal{B}_{\vec{v}}}_{\square_{\vec{v}}}(p)\, dp \\
&= V(\square_{\vec{v}}) \cdot |U| + \sum_{I \in \mathcal{I}''_{\vec{v}}} w(I) \cdot |I \setminus U| \\
&= \mathsf{Klee}(\mathcal{I}'_{\vec{v}}) \cdot V(\square_{\vec{v}}) + \sum_{I \in \mathcal{I}''_{\vec{v}}} V\left(I \setminus \bigcup \mathcal{I}'_{\vec{v}}\right) \cdot w(I).
\end{aligned}$$

◀

B.4 Proof of Lemma 17

We first observe the following simple fact.

▶ **Fact 20.** *For any disjoint intervals X and Y and any interval set $\mathcal{I}$,* $\mathsf{Klee}_{X \cup Y}(\mathcal{I}) = \mathsf{Klee}_X(\mathcal{I}) + \mathsf{Klee}_Y(\mathcal{I})$.

Let $\{e_1 < \cdots < e_N\}$ be the sorted distinct endpoints of all intervals in $\mathcal{I}$. Define $A_i = (e_i, e_{i+1})$ for $1 \le i \le N-1$, and $A_0 = (-\infty, e_1)$, $A_N = (e_N, \infty)$. Let $\mathcal{A} = \{A_0, \ldots, A_N\}$ be the set of these disjoint intervals. We consider a standard BST $\mathcal{T}$ on $\mathcal{A}$ storing the atomic intervals $A_i \in \mathcal{A}$ at its leaf nodes. For any node $v \in \mathcal{T}$, let A_v denote the interval spanned by its leaf descendants (atomic interval A_i if v is a leaf). Define

$$\kappa(v) = \begin{cases} |A_i| \cdot \mathbf{1}[\exists\, I \in \mathcal{I} : \ell(I) = e_i \text{ or } r(I) = e_{i+1}] \text{ where } I = [\ell(I), r(I)], & \text{if } v \text{ is a leaf,} \\ \displaystyle\sum_{v' \in v.\text{children}} \big(\kappa(v') \cdot \mathbf{1}[\mathcal{I}_{A_v} = \mathcal{I}_{A_{v'}}] + |A_{v'}| \cdot \mathbf{1}[\mathcal{I}_{A_v} \neq \mathcal{I}_{A_{v'}}]\big), & \text{if } v \text{ is internal.} \end{cases}$$

▶ **Fact 21.** $\kappa(v) = \mathsf{Klee}_{A_v}(\mathcal{I}_{A_v})$ *for all $v \in \mathcal{T}$.*

Proof. The claim holds trivially when v is a leaf node. Since A_i is atomic, $\mathcal{I}_{A_i}$ is nonempty when some $I \in \mathcal{I}$ includes A_i and defines one of its endpoints, thus $\mathsf{Klee}_{A_i}(\mathcal{I}_{A_i}) = |A_i|$. Otherwise, $\mathcal{I}_{A_i} = \emptyset$ and the measure is 0. If v is an internal node, then by Fact 20, $\mathsf{Klee}_{A_v}(\mathcal{I}_{A_v}) = \sum_{v' \in v.\text{children}} \mathsf{Klee}_{A_{v'}}(\mathcal{I}_{A_v})$. Thus it suffices to show that for each child v' either $\mathsf{Klee}_{A_{v'}}(\mathcal{I}_{A_v}) = \kappa(v')$ or $|A_{v'}|$. If $\mathcal{I}_{A_v} = \mathcal{I}_{A_{v'}}$ then $\mathsf{Klee}_{A_{v'}}(\mathcal{I}_{A_v}) = \mathsf{Klee}_{A_{v'}}(\mathcal{I}_{A_{v'}}) = \kappa(v')$ trivially by induction. Otherwise $\mathcal{I}_{A_v} \neq \mathcal{I}_{A_{v'}}$, and since $A_{v'} \subseteq A_v$ we cannot have an interval that contains A_v but fails to contain $A_{v'}$, so the difference lies in the existence of some $I' \in \mathcal{I}_{A_v} \setminus \mathcal{I}_{A_{v'}}$. Thus I' is an interval that fully contains $A_{v'}$ but does not contain A_v, consequently every point of $A_{v'}$ lies in some interval of $\mathcal{I}_{A_v}$, and therefore $\mathsf{Klee}_{A_{v'}}(\mathcal{I}_{A_v}) = |A_{v'}|$. ◀

To compute $\mathsf{Klee}_Q(\mathcal{I}_Q)$, let $\mathcal{V}$ denote the $O(\log N)$ canonical nodes whose intervals $\{A_v : v \in \mathcal{V}\}$ form a disjoint partition of Q. By Fact 20, $\mathsf{Klee}_Q(\mathcal{I}_Q) = \sum_{v \in \mathcal{V}} \mathsf{Klee}_{A_v \cap Q}(\mathcal{I}_Q)$. For each $v \in \mathcal{V}$. If $A_v \subseteq Q$, then $\mathsf{Klee}_{A_v \cap Q}(\mathcal{I}_Q) = \mathsf{Klee}_{A_v}(\mathcal{I}_Q) = \mathsf{Klee}_{A_v}(\mathcal{I}_{A_v}) \cdot \mathbf{1}[\mathcal{I}_{A_v} = \mathcal{I}_Q] + |A_v| \cdot \mathbf{1}[\mathcal{I}_{A_v} \neq \mathcal{I}_Q]$. Otherwise if $A_v \not\subseteq Q$, then v is necessarily a leaf corresponding to some atomic interval A_i, and $\mathsf{Klee}_{A_v \cap Q}(\mathcal{I}_Q) = \mathsf{Klee}_{A_i \cap Q}(\mathcal{I}_Q) = \kappa(A_i) \cdot |A_i \cap Q|/|A_i|$. Since $\mathbf{1}[\mathcal{I}_{A_v} = \mathcal{I}_Q]$ can be evaluated in $\widetilde{O}(1)$ time using the auxiliary structure $\mathbf{A}$, and $|\mathcal{V}| = O(\log N)$, the value of $\mathsf{Klee}_Q(\mathcal{I}_Q)$ can be computed in $\widetilde{O}(1)$ time via queries to $\mathcal{T}$.

Furthermore, insertions or deletions of an interval I' in $\mathcal{I}$ can be reflected in $\mathcal{T}$ in $\widetilde{O}(1)$ time. Observe that $\kappa(v)$ remains unchanged for any node v spanned by I', since by definition $I' \notin \mathcal{I}_{A_v} = \{I \in \mathcal{I} : A_v \not\subseteq I\}$ if $A_v \subseteq I'$. Hence, the update only affects leaves incident to endpoints of I' and the leaves' ancestors. Accounting for rotations, splits, and merges in $\mathcal{T}$, at most $O(\log N)$ nodes are changed in $\mathcal{T}$. Let P be the union of root-to-leaf paths from affected nodes. For each $v \in P$, recomputing $\kappa(v)$ using $\mathbf{A}$ takes $\widetilde{O}(1)$ time and restores the invariant $\kappa(v) = \mathsf{Klee}_{A_v}(\mathcal{I}_{A_v})$. Therefore, the total update time is $\widetilde{O}(1)$.