Fast Approximation Algorithms for Piercing Boxes by Points

Pankaj K. Agarwal^{*} Sariel Har-Peled[†]

Rahul Raychaudhury[‡]

Stavros Sintos[§]

November 6, 2023

Abstract

Let $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ be a set of n axis-aligned boxes in \mathbb{R}^d where $d \geq 2$ is a constant. The *piercing problem* is to compute a smallest set of points $\mathcal{N} \subset \mathbb{R}^d$ that hits every box in \mathcal{B} , i.e., $\mathcal{N} \cap \mathbf{b}_i \neq \emptyset$, for $i = 1, \ldots, n$. The problem is known to be NP-Hard. Let $p := p(\mathcal{B})$, the *piercing number* be the minimum size of a piercing set of \mathcal{B} . We first present a randomized $O(\log \log p)$ -approximation algorithm with expected running time $O(n^{d/2} \operatorname{polylog}(n))$. Next, we show that the expected running time can be improved to near-linear using a sampling-based technique, if $p = O(n^{1/(d-1)})$. Specifically, in the plane, the improved running time is $O(n \log p)$, assuming $p < n/\log^{\Omega(1)} n$. Finally, we study the dynamic version of the piercing problem where boxes can be inserted or deleted. For boxes in \mathbb{R}^2 , we obtain a randomized $O(\log \log p)$ -approximation algorithm with $O(n^{1/2} \operatorname{polylog}(n))$ amortized expected update time for insertion or deletion of boxes. For squares in \mathbb{R}^2 , the update time can be improved to $O(n^{1/3} \operatorname{polylog}(n))$.

Our algorithms are based on the multiplicative weight-update (MWU) method and require the construction of a weak ε -net for a point set with respect to boxes. A key idea of our work is to exploit the duality between the piercing set and independent set (for boxes) to speed up our MWU. We also present a simpler and slightly more efficient algorithm for constructing a weak ε -net than in [Ezr10], which is of independent interest. Our approach also yields a simpler algorithm for constructing (regular) ε -nets with respect to boxes for d = 2, 3.

1. Introduction

Problem statement. A *box* is an axis-aligned box in \mathbb{R}^d of the form $\prod_{i=1}^d [\alpha_i, \beta_i]$. A one dimensional box is an *interval*, and a two dimensional box is a *rectangle*. Let $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ be a set of *n* boxes in \mathbb{R}^d . A subset $\mathcal{N} \subset \mathbb{R}^d$ is a *piercing set* of \mathcal{B} if $\mathcal{N} \cap R \neq \emptyset$ for every box $\mathbf{b} \in \mathcal{B}$. The *piercing problem* asks to find a piercing set of \mathcal{B} of the smallest size, which we denote by $p := p(\mathcal{B})$ and call the *piercing number* of \mathcal{B} . Although the piercing problem can be defined over arbitrary geometric objects such as disks and halfspaces, here we focus on boxes. The piercing problem is a fundamental problem in computational geometry and has applications in facility location, sensor networks, etc.

^{*}Department of Computer Science, Duke University, Durham NC 27708. Work by Pankaj Agarwal was partially supported by NSF grants IIS-18-14493, CCF-20-07556, and CCF-22-23870.

[†]Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; sariel@illinois.edu; http://sarielhp.org/. Work by Sariel Har-Peled was partially supported by a NSF AF award CCF-1907400 and CCF-2317241.

[‡]Department of Computer Science, Duke University, Durham NC 27708.

[§]Department of Computer Science. University of Illinois at Chicago, Chicago IL 60607.

The piercing problem is closely related to the classical geometric hitting-set problem: Given a (geometric) range space $\Sigma = (\mathcal{X}, \mathcal{R})$, where \mathcal{X} is a (finite or infinite) set of points in \mathbb{R}^d and $\mathcal{R} \subseteq 2^{\mathcal{X}}$ is a finite family of ranges, defined by simply-shaped regions such as rectangles, balls, hyperplanes etc. That is, each range in Σ is of the form $R \cap \mathcal{X}$, where $R \in \mathcal{R}$ is a geometric shape (strictly speaking, $\Sigma = (\mathcal{X}, \{R \cap \mathcal{X} \mid R \in \mathcal{R}\})$ but with a slight abuse of notation we will use $(\mathcal{X}, \mathcal{R})$ to denote the range space). A subset $H \subseteq \mathcal{X}$ is a hitting set of Σ if $H \cap R \neq \emptyset$ for all $R \in \mathcal{R}$. The hitting-set problem asks for computing a minimum-size hitting set of Σ . The piercing problem is a special case of the hitting-set problem in which $\Sigma = (\mathbb{R}^d, \mathcal{B})$. Instead of letting the set of points be the entire \mathbb{R}^d , we can choose the set of points to be the set of vertices in $\mathcal{A}(\mathcal{B})$, the arrangement of \mathcal{B} ,¹ and the range space is now $\Sigma = (\mathcal{V}, \{b \cap \mathcal{V} \mid b \in \mathcal{B}\})$, where $\mathcal{V} := \mathcal{V}(\mathcal{B})$ is the set of vertices in $\mathcal{A}(\mathcal{B})$. It is easily seen that \mathcal{B} has a piercing set of size p if and only if Σ has a hitting set of size p. Hitting set for general range spaces was listed as one of the original NP-complete problems [Kar72]. Furthermore, the box piercing problem is NP-Complete even in 2D [FPT81], so our goal is to develop an efficient approximation algorithm for the piercing problem.

In many applications, especially those dealing with large data sets, simply a polynomial-time algorithm is not enough, and one desires an algorithm whose running time is near-linear in $|\mathcal{B}|$. In principle, the classical greedy algorithm can be applied to the range space $(\mathcal{V}, \mathcal{B})$, but $|\mathcal{V}| = O(n^d)$, so it will not lead to a fast algorithm. Intuitively, due to the unconstrained choice of points with which to pierce boxes of \mathcal{B} , the piercing problem seems easier than the geometric hitting-set problem and should admit faster solutions and better approximations. In this paper, we make progress towards this goal for a set of boxes in both static and dynamic settings.

Related work. The well-known shifting technique by Hochbaum and Maass [HM85] can be used to obtain a PTAS when \mathcal{B} comprises of unit-squares or near-equal-sized fat objects in any fixed dimension. Effat et al. [EKNS97] designed an O(1)-approximation algorithm for a set of arbitrary "fat" objects that runs in near-linear time in 2d and 3d. Chan [Cha03] gave a separator-based PTAS for arbitrary sized fat objects, with running time $O(n^{\varepsilon^{-d}})$. Chan and Mahmood [CM05] later gave a PTAS for a set of boxes with arbitrary width but unit height. All of the above results consider a restricted setting of boxes. Surprisingly, little is known about the piercing problem for a set of arbitrary axis-aligned boxes in \mathbb{R}^d . By running a greedy algorithm or its variants based on a multiplicative weight update (MWU) method, an $O(\log p)$ -approximation algorithm with running time roughly $O(n^d)$ can be obtained for the box-piercing problem in \mathbb{R}^d . Using the weak ε -net result by Ezra [Ezr10] (see also [AES09]), the approximation factor improves to $O(\log \log p)$. An interesting question is what is the smallest piercing set one can find in near-linear time. Nielsen [Nie00] presented an $O(\log^{d-1} p)$ -approximation divide-and-conquer algorithm that runs in $O(n \log^{d-1} n)$ time. We are unaware of any near-linear time algorithm even with $O(\log p)$ approximation ratio for d > 3. We note that the piercing problem has also been studied in discrete and convex geometry, where the goal is to bound the size of the piercing set for a family of objects with certain properties. See e.g. [AK92, CSZ18].

We conclude this discussion by mentioning that there has been much work on the geometric hitting-set problem. For a range space $\Sigma = (\mathcal{X}, \mathcal{R})$ and weight function $\omega : \mathcal{X} \to \mathbb{R}_{\geq 0}$, a subset $\mathcal{N} \subseteq \mathcal{X}$ is an ε -net if for any $R \in \mathcal{R}$ with $\omega(R) \geq \varepsilon \omega(\mathcal{X})$, we have $R \cap \mathcal{N} \neq \emptyset$. The multiplicative weight update (MWU) method assigns a weight to each point so that every range in \mathcal{R} becomes "1/ep-

¹The arrangement of \mathcal{B} , denoted by $\mathcal{A}(\mathcal{B})$, is the partition of \mathbb{R}^d into maximal connected cells so that all points within each cell are in the interior/boundary of the same set of rectangles. It is well known that $\mathcal{A}(\mathcal{B})$ has $O(n^d)$ complexity [Ede87].

heavy" and then one simply chooses a 1/ep-net. Using the MWU method and results on ε -nets, Brönnimann and Goodrich [BG95] presented a polynomial-time $O(\log p)$ -approximation algorithm for the hitting-set problem for range spaces with finite VC-dimension [VC71]. Later Agarwal and Pan [AP20] presented a more efficient implementation of the MWU method for geometric range spaces. Their approach led to $O((|X| + |\mathcal{R}|) \operatorname{polylog}(n))$ algorithm with $O(\log p)$ -approximation for many cases including a set of rectangles in \mathbb{R}^d (see also [BMR18, CH20]). But it does not lead to a near-linear time algorithm for the piercing problem because $|X| = O(n^d)$ in this case. In another line of work, polynomial-time approximation algorithms for hitting sets based on local search have also been proposed [MR10].

The MWU algorithm essentially solves and rounds the LP associated with the hitting-set problem, see [Har11, Chapter 6]. Thus, the approximability of the problem is strongly connected to the integrality gap of the LP. For the hitting-set problem of points with boxes for $d \leq 3$, Aronov *et al.* [AES09] showed a rounding scheme with integrality gap $O(\log \log p)$. Furthermore, Ezra [Ezr10] showed the same gap holds in higher dimensions if one is allowed to use any point to do the piercing. Surprisingly, Pach and Tardos [PT11] showed that this integrality gap is tight. While a better approximation than the integrality gap can be obtained in a few cases [CH12, MR10], these algorithms require a fundamentally different approach. Thus, a major open problem is to obtain an O(1)-approximation algorithm for the box-piercing problem.

Recently, Agarwal *et al.* [ACS+22] initiated a study of dynamic algorithms for geometric instances of set-cover and hitting-set. Here the focus is on maintaining an approximately optimal hitting-set (resp. set-cover) of a dynamically evolving instance, where in each step a new object may be added or deleted. They introduced fully dynamic sublinear time hitting-set algorithms for squares and intervals. These results were improved and generalized in [CHSX22, CH21]. Khan *et al.* [KLR+23] proposed a dynamic data structure for maintaining a O(polylog(n))-factor approximation of the optimal hitting-set for boxes under restricted settings, but no algorithm with sublinear update time for the general setting is known.

Our results. In this paper, we design an efficient $O(\log \log p)$ -approximation algorithm for the box-piercing problem. Let \mathcal{B} be a set of n boxes in \mathbb{R}^d . A naive way to get an $O(\log \log p)$ -approximation is by using aforementioned MWU [AP20, BG95] based hitting-set algorithms on the range space $(\mathcal{V}, \mathcal{B})$, and use Ezra's [Ezr10] algorithm for computing a weak ε -net instead of computing a strong ε -net. While this naive approach gives the desired approximation, it runs in $\Omega(n^d)$ time. We present the following results.

(A) A NEW MWU AND ITS FAST IMPLEMENTATION. We present two algorithms in Section 2. The first is essentially the one by Agarwal-Pan that computes a hitting set of the range space $(\mathcal{V}, \mathcal{B})$ of size $O(p \log \log p)$. We show that it can be implemented in $O(n^{(d+1)/2} \log^3 n)$ expected time (Theorem 2.3). To achieve the desired running time, we need a data structure to perform all the required operations on \mathcal{V} without ever explicitly constructing it. In Section 5, we present such a data structure, which exploits the properties of the partitioning technique by Overmars and Yap [OY91].

Our main result is, however, a different MWU algorithm tailored for boxes with $O(n^{d/2} \log^{2d+3} n)$ expected running time (Theorem 2.10). It exploits the duality between the piercing-set problem and the independent-set problem, along with fast approximation algorithms for these two problems. The basic idea is to use the aforementioned approximation algorithm to find a large set of independent boxes among the light boxes identified by the MWU algorithm in a round. If the algorithm does not find such an independent set, then we can use a simple piercing-set algorithm

to compute a desired piercing set. Otherwise we double the weight of the boxes in the independent set. The idea of duality and approximation to speedup the MWU is critical to get a near linear running time in the plane. As far as we are aware, the idea of quickly rounding the dual problem and using it to speedup the primal algorithm, in the context of set-cover/hitting-set/piercing-set for rectangles, was not used before. This algorithm also relies on the data structure described in Section 5.

(B) PIERCING, CLUSTERING, AND MULTI-ROUND ALGORITHMS. We show that the natural algorithm of first computing a piercing set \mathcal{P}_0 for a random sample of input boxes, and then piercing the input boxes that are not pierced by \mathcal{P}_0 leads to an efficient piercing algorithm. This algorithm can be extended to run an arbitrary number of rounds. For clustering this idea was described by Har-Peled [Har04] (but the idea is much older see e.g. [GRS98]). In particular, if the algorithm runs ζ rounds, then it needs to compute piercing sets ζ times for sets of boxes of size (roughly) $p^{1-1/\zeta}n^{1/\zeta}$. By picking ζ a sufficiently large constant, we obtain an $O(\zeta \log \log p)$ approximation algorithm that runs in near-linear expected time, provided that $p = O(n^{1/(d-1)})$. See Theorem 3.3 and Corollary 3.4. For d = 2, this leads to the striking result that one can obtain an $O(\log \log p)$ -approximation algorithm with $O(n \log p)$ expected running time provided that $p = O(n/\log^{15} n)$, see Corollary 3.5.

(C) DYNAMIC ALGORITHMS FOR PIERCING. We consider the piercing problem for a set \mathcal{B} of boxes in \mathbb{R}^2 in the *dynamic* setting, i.e., at each step a new box is inserted into or deleted from \mathcal{B} . Our goal is to maintain an (approximately) optimal solution of the current set. We implement a dynamic version of the multi-round sampling based algorithm in Section 3, and attain a randomized Monte Carlo $O(\log \log p)$ -approximation algorithm with $O(n^{1/2} \operatorname{polylog}(n))$ amortized expected update time. The update time improves to $O(n^{1/3} \operatorname{polylog}(n))$ if \mathcal{B} is a set of squares in \mathbb{R}^2 . In principle, our approach extends to higher dimensions but currently we face a few technical hurdles in its efficient implementation (Section 4).

(D) NEW CONSTRUCTIONS OF (WEAK) ε -NETS. We present (in Section 6) a simpler and more efficient algorithm for constructing a weak ε -net then the one in [Ezr10]. In particular, given a set \mathcal{P} of n points in \mathbb{R}^d , a weight function $w : \mathcal{P} \to \mathbb{R}_{\geq 0}$, and a parameter $\varepsilon \in (0, 1)$, it computes a set $\mathcal{N} \subseteq \mathbb{R}^d$ of $O(\varepsilon^{-1} \log \log \varepsilon^{-1})$ points (not necessarily a subset of \mathcal{P}) in $O(n + \varepsilon^{-1} \log^{O(d^2)} \varepsilon^{-1})$ expected time such that for any box **b** with $\omega(\mathbf{b} \cap \mathcal{P}) \geq \varepsilon \omega(\mathcal{P})$, we have $\mathbf{b} \cap \mathcal{N} \neq \emptyset$. The running time can be improved to $O((n + m + \varepsilon^{-1}) \log^d \varepsilon^{-1})$ if we wish to guarantee above property for a given set of m boxes, which is the case in our setting. Our technique also gives an efficient algorithm for constructing an ε -net of size $O(\varepsilon^{-1} \log \log \varepsilon^{-1})$ for rectangles in d = 2, 3, which is somewhat simpler than the one in [AES09].

Building on the earlier work [AES09, Ezr10, PT11], our work brings the key insights of the results to the forefront. Our main new ingredient is using poly-logarithmic different grids to "capture" the distribution of the point sets. This idea appears in the work of Pach and Tardos [PT11] in the construction of the lower-bound, so it is not surprising that it is useful in the construction itself (i.e., upper-bound).

2. Piercing set via multiplicative weight update (MWU) algorithm

We describe two multiplicative-weight-update (MWU) based $O(\log \log p)$ -approximation algorithms for computing a piercing set for \mathcal{B} . Let $\mathcal{V} = \mathcal{V}(\mathcal{B})$ denote the set of vertices in the arrangement $\mathcal{A}(\mathcal{B})$. As already mentioned, there is always an optimal piercing set $\mathcal{N} \subseteq \mathcal{V}$ of \mathcal{B} . The basic idea

Dim	preprocessing	Std. operations	sample	ref
d = 1	$O(n\log n)$	$O(\log n)$	$O(\log n)$	Lemma 5.1
d = 2	$O(n^{3/2}\log n)$	$O(\sqrt{n}\log n)$	$O(\log n)$	Lemma 5.4
d > 2	$O(n^{(d+1)/2}\log n)$	$O(n^{(d-1)/2}\log n)$	$O(\log n)$	Lemma 5.4

Figure 2.1: We describe a data-structure for maintaining (implicitly) the vertices of an arrangement of boxes, under operations weight, double, halve, insert, delete, see Definition 2.1.

is to reweight the points of \mathcal{V} such that all boxes become *heavy*. Specifically, we compute a weight function $\omega : \mathcal{V} \to \mathbb{Z}^+$, such that for any box $\mathbf{b} \in \mathcal{B}$, we have $\omega(\mathbf{b} \cap \mathcal{V}) = \sum_{p \in \mathbf{b} \cap \mathcal{V}} \omega(p) \ge \omega(\mathcal{V})/2pe$. We then compute a weak $\frac{1}{2pe}$ -net \mathcal{N} of the range space $(\mathcal{V}, \mathcal{B})$ with respect to the above weight function. By definition \mathcal{N} is a piercing set of \mathcal{B} . The two algorithms differ in how the weights are updated. Before describing the algorithms, we give the specifications of a data structure used by both algorithms. We defer the implementation details of the data structure to Section 5.

For a multi-set $S \subseteq B$ of boxes, and a point $p \in \mathbb{R}^d$, let $S \sqcap p = \{b \in S \mid p \in b\}$ be the multi-set of all boxes in S containing p, let

$$w_{\mathcal{S}}(p) := 2^{|\mathcal{S} \sqcap p|} \tag{2.1}$$

be the **doubling weight** of p. For a finite set of points $X \subseteq \mathbb{R}^d$, let $w_{\mathcal{S}}(X) := \sum_{x \in X} w_{\mathcal{S}}(p)$.

Definition 2.1 (Implicit arrangement data-structure). Let \mathcal{B} be a set of axis-aligned boxes in \mathbb{R}^d (known in advance). Let $\mathcal{C} \subseteq \mathcal{B}$ be a set of *active* boxes (initially empty) and let $\mathcal{V}(\mathcal{C})$ be the set of vertices of the arrangement $\mathcal{A}(\mathcal{C})$. Let $\mathcal{S} \subseteq \mathcal{B}$ be a multi-set of *update* boxes (initially empty). \mathcal{S} induces a weight function on the vertices in $\mathcal{V}(\mathcal{C})$ using Eq. (2.1). Here, we require a data-structure that supports the following operations:

- (I) weight(b): given a box b, compute $w_{\mathcal{S}}(\mathcal{V}(\mathcal{C}) \cap b)$.
- (II) double(b): given a box $b \in \mathcal{B}$, adds a copy of b to the multi-set of update boxes \mathcal{S} .
- (III) halve(b): given a box $b \in \mathcal{B}$, removes a copy of b from the multi-set of update boxes \mathcal{S} .
- (IV) sample: returns a random point $p \in \mathcal{V}(\mathcal{C})$, with probability $w_{\mathcal{S}}(p)/w_{\mathcal{S}}(\mathcal{V}(\mathcal{C}))$.
- (V) **insert**(b): inserts b into the set of active boxes C.
- (VI) delete(b): removes b from the set of active boxes C.

Figure 2.1 summarizes the performance of the data structure. In particular, it can be constructed in $O(n^{(d+1)/2} \log n)$ time, each of the above operations can be supported in $O(n^{(d-1)/2} \log n)$ time per operation, except for **sample**, which takes only $O(\log n)$ time. See Lemma 5.4. Note that the **halve** and **delete** operations are not used by the basic algorithm.

2.1. Basic MWU algorithm

The algorithm is a small variant of the Agarwal-Pan algorithm [AP20]. The algorithm performs an exponential search on a value k, and it stops when $k \ge p$ and $k \le 2p$. Specifically, in the *i*th stage, k is set to 2^i . We next describe such a stage for a fixed value of k (and i).

In the beginning of the stage, $\omega(p) = 1$ for all $p \in \mathcal{V}$. Let $\varepsilon = \frac{2}{3k}$. A box **b** is ε -*light* if $\omega(\mathcal{V} \cap \mathbf{b}) < \varepsilon \omega(\mathcal{V})$, and ε -*heavy* otherwise. The algorithm proceeds in rounds. In each round, it

scans all the boxes in \mathcal{B} trying to find an ε -light box. If an ε -light box **b** is found, the algorithm performs a *doubling* operation on its points. That is, it doubles the weight of each of the points in $\mathcal{V} \cap \mathbf{b}$. The algorithm performs the doubling on **b** repeatedly until **b** becomes ε -heavy (in relation to the updated weight $\omega(\mathcal{V})$). The algorithm then resumes the scan for ε -light boxes (over the remaining boxes). Importantly, the algorithm never revisits a box during a round (thus a box that becomes heavy during a round might become light again in that round). Let $\ell = \lfloor 1/\varepsilon \rfloor \leq 2k$. If ℓ doubling operations are performed in a round, the algorithm aborts the round, and proceeds to the next round. If a round was completed without ℓ weight-doubling being performed, the algorithm computes a weak (ε/e)-net \mathcal{N} of (\mathcal{V}, \mathcal{B}) using the algorithm described below in Section 6 (specifically, Lemma 6.12).

Finally, if the number of rounds exceeds $\tau = c \ln(|\mathcal{V}|/k)$ at any stage, where c > 0 is a suitably large constant, the algorithm decides the guess for k as too small, doubles the value of k, and continues to the next stage, till success.

In order to implement the above algorithm, we use the implicit arrangement data structure from Definition 2.1. At the beginning of each stage, we build an instance of the data structure on \mathcal{B} and add all the boxes to the active set using the insert operation. The doubling step with a box b is performed using double(b) and the ε -lightness of a box is tested using the weight(b) operation.

Analysis. The following lemma proves the correctness of the algorithm.

. . . .

Lemma 2.2 (AP20). If $k \in [p/2, p]$, then the algorithm returns a piercing set for \mathcal{B} , of size $O(p \log \log p)$, where $p = p(\mathcal{B})$ is the size of the optimal piercing set of \mathcal{B} .

Proof: This claim is well known [AP20, Lemma 3.1], but we include a proof for the sake of completeness.

Initially, the weight W_0 of all the vertices of \mathcal{V} is $m = |\mathcal{V}| = O(n^d)$. Let W_i be the weight of \mathcal{V} after the *i*th doubling operation, and observe that

$$p2^{\lfloor i/p \rfloor} \le W_i \le (1+\varepsilon)^i W_0 \le m \exp(\varepsilon i).$$

The lower bound follows as every doubling operation must double the weight of one of the p points in the optimal piercing set (and to minimize this quantity, this happens in a round robin fashion). The upper bound follows readily from the ε -lightness of the box being doubled. Taking i = tp, and taking the log of both sides (in base 2), we have

$$\log p + t \le \log m + \varepsilon t p \log e \le \log m + \frac{2}{3k} t p \cdot 1.45 \le \log m + 0.97t \frac{p}{k}.$$

Assuming $k \ge p$, this readily implies that $t = O(\log \frac{m}{k})$. Namely, the algorithm performs at most $\tau = O(k \log(m/k))$ doubling operations in a stage, if the guess $k \ge p$.

Since every round performs exactly $\ell = \lfloor 1/\varepsilon \rfloor = \lfloor 3k/2 \rfloor \ge k$ doubling operations in each round, except the last one, it follows that the algorithm performs at most $\tau/k = O(\log \frac{m}{k})$ rounds.

So consider the last round. Assume the weight of \mathcal{V} at the start of the round was W. At the end of the round, the total weight of \mathcal{V} is at most $(1 + \varepsilon)^{\ell}W \leq \exp(\varepsilon \lfloor 1/\varepsilon \rfloor)W \leq eW$. Every box during this round, must have been heavy (at least for a little while), which implies that it had weight $\geq \varepsilon W$, as weights only increase during the algorithm execution. This implies that all the boxes are ε/e -heavy at the end of the round. Thus, the weak ε/e -net the algorithm computes must pierce all the boxes of \mathcal{B} .

The above lemma shows that the algorithm succeeds if it stops, and it must stop in a stage if k is sufficiently large.

Theorem 2.3. Let \mathcal{B} be set of n axis-aligned boxes in \mathbb{R}^d , for some fixed $d \ge 2$, and let $p = p(\mathcal{B})$ be the piercing number of \mathcal{B} . The above algorithm computes, in $O(n^{(d+1)/2} \log^3 n)$ expected time, a piercing set of \mathcal{B} of size $O(p \log \log p)$.

Proof: The above implies that if the algorithm outputs a piercing set, then it is of the desired size (it might be that the algorithm stops at an earlier stage than expected with a guess of k that is smaller than p). Thus, it must be that if the guess for the value of k is too small, then the algorithm double weights in vain, and performs too many rounds (i.e., their number exceeds τ), and the algorithm continues to the next stage.

Let $h = \lceil \log p \rceil$ and $m = |\mathcal{V}|$. Overall, the algorithm performs at most

$$\sum_{i=1}^{h} O(2^{i} \log m) = O(p \log m) = O(p \log n)$$
(2.2)

doubling operations (i.e., **double**). Every round requires n weight operations. There are $O(\log m)$ rounds in each stage, and there are h stages. We conclude that the algorithm performs

$$O(n\log m\log p) = O(n\log^2 n)$$

weight operations. Thus, the overall running time (including preprocessing and activating all boxes), is

$$O(n^{(d+1)/2}\log n + n \cdot n^{(d-1)/2}\log n \cdot \log^2 n) = O(n^{(d+1)/2}\log^3 n).$$

Finally, the algorithm in Section 6 for computing a weak ε -net first chooses a random sample \mathcal{Q} of \mathcal{V} of size $O(\varepsilon^{-1}\log\varepsilon^{-1})$ and then computes a weak $\frac{\varepsilon}{2}$ -net of $(\mathcal{Q}, \mathcal{B})$. Using sample, \mathcal{Q} can be computed in $O(|\mathcal{Q}|\log n)$ time. Combining this with Lemma 6.12, we conclude that a ε/e -net of $(\mathcal{V}, \mathcal{B})$ of size $O(p \log \log p)$ can be computed in $O((n + \varepsilon^{-1}) \log^d n)$ time, which is dominated by the data-structure's overall running time.

2.2. An improved MWU algorithm

We now present an alternative algorithm for piercing that exploits LP duality. Operationally, the improved algorithm differs from the basic one in a few ways. The algorithm does not use the implicit arrangement data-structure to maintain the weights on \mathcal{V} directly. Moreover, instead of doubling the weight of light boxes one at a time, the algorithm performs "batch doubling", i.e., it doubles the weights of a collection of boxes at the same time. The algorithm also does not compute the weights of boxes exactly. Instead, it approximates the weights using a suitable sample. This sample is periodically recomputed by a process we call "batch sampling" that uses the implicit arrangement data structure in $\mathbb{R}^{(d-1)}$. Before describing the improved algorithm, we explore the duality between piercing sets and independent sets, and discuss the details of batch sampling.

LP duality, piercing and independence numbers. Let \mathcal{B} be a set of boxes in \mathbb{R}^d , and let \mathcal{V} be the set of vertices of the arrangement $\mathcal{A}(\mathcal{B})$. A subset $\mathcal{X} \subseteq \mathcal{B}$ is an *independent set* if no pair of boxes of \mathcal{X} intersect. The *independence number* of \mathcal{B} , denoted by $i(\mathcal{B})$, is the size of the largest independent set $\mathcal{X} \subseteq \mathcal{B}$. Observe that $i(\mathcal{B}) \leq p(\mathcal{B})$, as each box in an independent set must be

$$p^{*}(\mathcal{B}) = \min \sum_{v \in \mathcal{V}} x_{v}$$
$$\forall \mathbf{b} \in \mathcal{B} \qquad \sum_{v \in \mathbf{b} \cap \mathcal{V}} x_{v} \ge 1$$
$$\forall v \in \mathcal{V} \qquad x_{v} \ge 0.$$

$$\begin{split} i^*(\mathcal{B}) = \max \sum_{\mathbf{b} \in \mathcal{B}} y_{\mathbf{b}} \\ \forall v \in \mathcal{V} \qquad \sum_{\mathbf{b} \in \mathcal{B}: v \in \mathbf{b}} y_{\mathbf{b}} \leq 1 \\ \forall \mathbf{b} \in \mathcal{B} \qquad y_{\mathbf{b}} \geq 0. \end{split}$$

The LP for the fractional piercing number.

Figure 2.2: LPs for the piercing and Independence problems. By duality, we have $p^*(\mathcal{B}) = i^*(\mathcal{B})$.

pierced, but no point can pierce more than one box in the independent set. Let $p^* = p^*(\mathcal{B})$ denote the fractional piercing number of \mathcal{B} , i.e., this is the minimum piercing number of \mathcal{B} when solving the associated LP, see Figure 2.2. The dual LP, is the fractional independence LP – it computes (fractionally) a maximum independent set of boxes in \mathcal{B} . The value of this LP $i^* := i^*(\mathcal{B})$ is the *fractional Independence number* of \mathcal{B} . By LP duality $p^* = i^*$, see Figure 2.2. It is known, and also implied by Lemma 2.2, that $p^* \leq p \leq c_1 p^* \log \log p^*$, for some constant $c_1 > 0$. This implies that $p^* \geq p/(c_1 \log \log p)$.

We need the following standard algorithms for computing independent set and piercing set of boxes. Better results are known [CC09, GKM+22, Mit21], but they are unnecessary for our purposes.

Lemma 2.4 (AKS97). Let \mathcal{B} be a set of n boxes in \mathbb{R}^d . An independent subset of boxes of \mathcal{B} of size $\Omega(i^*/\log^{d-1} n)$ can be computed in $O(n\log^{d-1} n)$ time, where $i^* = i^*(\mathcal{B})$ is the fractional independence number of \mathcal{B} .

Proof: This algorithm is well known and we sketch it here for completeness – it works by induction on the dimension. For d = 1, a greedy algorithm picking the first interval with the minima right endpoint computes the optimal independent set — the running time of this algorithm is O(n) after sorting. It is straightforward to verify that one can assume that the LP solution in this case is integral, and thus $i^* = i(\mathcal{B})$.

For d = 2, compute the vertical line such that its *x*-coordinate is the median of the *x*-coordinates of the endpoints of the boxes. Compute the optimal independent set of rectangles intersecting this line (which is just the one dimensional problem), and now recurse on the two subsets of rectangles that do not intersect the median line. This results in a partition of the set of rectangles \mathcal{B} into $O(\log n)$ sets, such that for each set, we have the optimal (fractional) independent set. Clearly, one of them has to be of size $\Omega(i^*/\log n)$.

For d > 2, the same algorithm works by approximating the optimal solution along the median of the first coordinate (i.e., d - 1 subproblem), and then recursively on the two subproblems. The bounds stated readily follows.

The piercing problem can be solved exactly by the greedy algorithm in one dimension (i.e., add the leftmost right endpoint of an input interval to the piercing set, remove the intervals that intersect it, and repeat). For higher dimensions, one can perform the same standard divide and conquer approach, used in Lemma 2.4, and get the following.

Lemma 2.5 (Nie00). Let \mathcal{B} be a set of n boxes in \mathbb{R}^d for $d \geq 2$. A piercing set \mathcal{P} of \mathcal{B} of size $O(p^* \log^{d-1} n)$ can be computed in $O(n \log^{d-1} n)$ time.

Corollary 2.6. Let \mathcal{B} be a set of n boxes in \mathbb{R}^d for $d \ge 2$. Then the following two sets can be computed in $O(n \log^{d-1} n)$ time:

- (i) An independent set $\mathcal{B}' \subseteq \mathcal{B}$ of \mathcal{B} of size $\Omega(\frac{p}{\log^{d-1} n \log \log n})$, and
- (ii) a piercing set $\mathcal{P} \subseteq \mathbb{R}^d$ of \mathcal{B} of size $O(p \log^{d-1} n)$.

Proof: By Lemma 2.4, an independent set \mathcal{B}' of size $\Omega\left(\frac{i^*}{\log^{d-1}n}\right)$ can be computed in $O(n\log^{d-1}n)$ time, and by Lemma 2.5, a piercing set \mathcal{P} of size $O(p\log^{d-1}n)$ can be computed in $O(n\log^{d-1}n)$ time. By Lemma 2.2, we have

$$|\mathcal{B}'| = \Omega\left(\frac{i^*}{\log^{d-1} n}\right) = \Omega\left(\frac{p}{\log^{d-1} n \log \log n}\right),$$

Batch sampling. For the purposes of our improved algorithm, we need to support the following "batch sampling" operation. Given a set \mathcal{B} of n boxes in \mathbb{R}^d , a multiset (i.e., list) $\mathcal{S} \subseteq \mathcal{B}$ of boxes such that $|\mathcal{S}| = O(n \log n)$, and a parameter r > 0 such that $r = O(n \log n)$, the task at hand is to compute a random subset \mathcal{R} of r vertices of $\mathcal{A}(\mathcal{B})$, where each vertex $v \in \mathcal{A}(\mathcal{B})$ is sampled independently with probability $w_{\mathcal{S}}(p)/w_{\mathcal{S}}(\mathcal{V})$, see Eq. (2.1). This procedure can be implemented using the data structure described in Definition 2.1, but one can do better, by sweeping along the x_d -axis and constructing the data structure in one lower dimension, as follows.

Let \mathcal{B}_{\downarrow} be the collection of (d-1)-dimensional projections of the boxes in \mathcal{B} , to the hyperplane $x_d = 0$. Let \mathcal{V}' be the set of vertices in $\mathcal{A}(\mathcal{B}_{\downarrow})$, the arrangement of \mathcal{B}_{\downarrow} in \mathbb{R}^{d-1} . Let \mathcal{F} be the set of x_d -coordinates of all the vertices of the boxes of \mathcal{B} .

Build \mathcal{D} , an instance of the (d-1)-dimensional data structure of Definition 2.1 on \mathcal{B}_{\downarrow} . Importantly, initially all the boxes of \mathcal{B}_{\downarrow} are inactive. Observe that $\mathcal{V} \subseteq \mathcal{V}' \times \mathcal{F}$. Assume \mathcal{F} is sorted and the ith point (in sorted order) is denoted by e_i . We perform two space-sweeps along the x_d -axis. In step *i* of a sweep, \mathcal{D} represents the point-set $\mathcal{J}_i := (\mathcal{V}' \times \{e_i\}) \cap \mathcal{V}$ lying on a $\mathbb{R}^{(d-1)}$ -dimensional subspace, and we compute $\alpha(i)$, the total weight of vertices in \mathcal{J}_i

At any step *i* of the first sweep, for every box $\mathbf{b} \in \mathcal{B}$ that starts (resp. ends) at e_i call **insert**(b) (resp. **delete**(b)). This activates/deactivates all the vertices on the boundary on **b**. Similarly, for every $\mathbf{b} \in \mathcal{S}$ that begins (resp. ends) at e_i , use **double**(b) (resp. **halve**(b)) operation on \mathcal{D} to double (resp. halve) the weight of the points in $\mathcal{J}_i \cap \mathbf{b}$. Next, use the **weight** operation on \mathcal{D} to get the weight of all the points in \mathcal{J}_i and store it in $\alpha[i]$. At the end of the first sweep, independently sample r numbers from the set $\{1, ..., 2n\}$ where the integer j is sampled with probability $\alpha(j) / \sum_{i=1}^{2n} \alpha(i)$.

Let $\delta(j)$ denote the number of times j was sampled.

Next, reset \mathcal{D} and begin the second sweep. In step *i* of the second sweep, perform the same operations except replace the weight-computing step with taking $\delta(i)$ independent samples from \mathcal{J}_i using the sample operation on \mathcal{D} .

Observe that at the end of the first sweep $\alpha(i) / \sum_{j=1}^{2n} \alpha(j)$ is the total probability mass of all the points in \mathcal{J}_i . Using this fact, it is easy to verify that at the end of the second sweep the sampled points correspond to the desired subset \mathcal{R} .

Initializing \mathcal{D} takes $O(n^{d/2} \log n)$ time, and performing each step of the sweep takes

$$O(n^{(d-2)/2}\log n)$$

time. Each sweep handles $O(|\mathcal{S}|) = O(n \log n)$ events, hence, \mathcal{R} can be computed in $O(n^{d/2} \log^2 n)$ time. We thus obtain the following:

Lemma 2.7. Let \mathcal{B} be a set of n boxes in \mathbb{R}^d , and let \mathcal{S} be a multiset of boxes such that $|\mathcal{S}| = O(n \log n)$. Let ω be the doubling weight function induced by \mathcal{S} over the vertices \mathcal{V} of $\mathcal{A}(\mathcal{B})$, see Eq. (2.1). Given a parameter r > 0 such that $r = O(n \log n)$, a random subset $\mathcal{R} \subset \mathcal{V}$ of size r, where each vertex of \mathcal{V} is sampled with probability proportional to its weight, can be computed in $O(n^{d/2} \log^2 n)$ time.

The new MWU algorithm. As in the previous algorithm, the algorithm performs an exponential search on k until $k \leq 2p$. For a particular guess k, the new algorithm also works in rounds. The difference is how each round is implemented. Instead of doubling the weight of a light box as soon as we find one, we proceed as follows. Let $\omega : \mathcal{V} \to \mathbb{R}_{\geq 0}$ be the weight function at the beginning of the current round. Observe that for any point $p \in \mathcal{V}$, $\omega(p) = w_{\mathcal{S}}(p)$ where \mathcal{S} is the multi-set of boxes that doubled their weights so far, see Eq. (2.1). At the beginning of each round, we process \mathcal{B} and \mathcal{S} to generate a random subset $\mathcal{R} \subset \mathcal{V}$ of size $O(k \log n)$, such that for a box $\mathbf{b} \in \mathcal{B}$ one can determine whether it is light, i.e., $\omega(\mathbf{b}) \leq \omega(\mathcal{V})/4k$ by checking if $|\mathbf{b} \cap \mathcal{R}| \leq |\mathcal{R}|/4k$. By processing \mathcal{R} into an orthogonal range-counting data structure [Aga04], we can compute $|\mathbf{b} \cap \mathcal{R}|$, in $O(\log^{d-1} n)$ time, for any $\mathbf{b} \in \mathcal{B}$, by querying the data structure with \mathbf{b} . By repeating this for all boxes of \mathcal{B} , we compute a set $\mathcal{L} \subseteq \mathcal{B}$ of light boxes.

Next, we compute an independent set of boxes $\mathcal{I} \subseteq \mathcal{L}$ using Lemma 2.4. There are several possibilities:

- If $|\mathcal{I}| \geq 2k$, then the guess for k is too small. We double the value of k, and restart the process.
- If $|\mathcal{I}| < c_1 k / \log^{2d-1} n$, then by Corollary 2.6, a piercing set \mathcal{P} of \mathcal{L} of O(k) points can be computed in $O(n \log^{d-1} n)$ time. The remaining boxes of $\mathcal{H} = \mathcal{B} \setminus \mathcal{L}$ are (say) 1/4.01*k*-heavy. By applying Lemma 6.12 to \mathcal{H} and \mathcal{R} , we compute a weak 1/4.01*k*-net \mathcal{N} for these boxes of size $O(k \log \log k)$ in $O(n \log^d n)$ expected time. The set $\mathcal{P} \cup \mathcal{N}$ is the desired piercing set of \mathcal{B} of size $O(p \log \log p)$.
- If $|\mathcal{I}| \geq c_1 k / \log^{2d-1} n$, then we update \mathcal{S} , the multi-set of boxes doubled so far, to $\mathcal{S} = \mathcal{S} \cup \mathcal{I}$. The algorithm now continues to the next round.

Remark 2.8. It is interesting to observe that the batch doubling operation in the above algorithm corresponds to merely updating the list S. The actual work associated with the doubling is in regenerating the sample \mathcal{R} at the beginning of the next round. This is done by batch sampling as described below.

Computing the random sample \mathcal{R} in each round. Let $\omega : \mathcal{V} \to \mathbb{R}_{\geq 0}$ be the weight function in the beginning of a particular round. Our goal is to compute a sample $\mathcal{R} \subset \mathcal{V}$, such that we can check if a box $\mathbf{b} \in \mathcal{B}$ is light by checking if $|\mathbf{b} \cap \mathcal{R}| \leq |\mathcal{R}|/4k$. Recall that \mathcal{S} denotes the list of boxes for which the MWU algorithm had doubled the weights. Moreover, recall that the weight of a vertex $v \in \mathcal{V}$ is $2^{|\mathcal{S} \cap v|}$, where $|\mathcal{S} \cap v|$ denotes the number of boxes in \mathcal{S} that contain v. We use the concept of relative approximations.

Definition 2.9. Let $\Sigma = (\mathcal{X}, \mathcal{R})$ be a finite range space, and $\mathcal{R} \subset \mathcal{X}$, and let $\omega : \mathcal{X} \to \mathbb{R}_{\geq 0}$ be a weight function. For a range $\mathbf{r} \in \mathcal{R}$, let $\overline{m}(\mathbf{r}) = \omega(\mathbf{r} \cap \mathsf{X})/\omega(\mathsf{X})$ and $\overline{s}(\mathbf{r}) = |\mathbf{r} \cap \mathcal{R}|/|\mathcal{R}|$. Then, \mathcal{R} is a *relative* (p, ε) -*approximation* of Σ if for each $\mathbf{r} \in \mathcal{R}$ we have:

- (i) If $\overline{m}(\mathbf{r}) \ge p$, then $(1 \varepsilon)\overline{m}(\mathbf{r}) \le \overline{s}(\mathbf{r}) \le (1 + \varepsilon)\overline{m}(\mathbf{r})$.
- (ii) If $\overline{m}(\mathbf{r}) \leq p$, then $\overline{s}(\mathbf{r}) \leq (1 + \varepsilon)p$.

It is known [HS11, Har11] that if the VC-dimension of Σ is d, then a random sample \mathcal{R} of size $O\left(\frac{1}{\varepsilon^2 p} \left[d \log \frac{1}{p} + \log \frac{1}{\varphi} \right] \right)$, where each point is chosen with probability proportional to its weight, is a relative (p, ε) -approximation with probability $\geq 1 - \varphi$. In view of this result, we can detect light boxes of \mathcal{B} (with respect to \mathcal{S}) as follows. Set $\varepsilon = 0.01$, $p = \frac{\varepsilon}{20k}$, $\varphi = \frac{1}{n^{O(1)}}$. We chose a random subset $\mathcal{R} \subseteq \mathcal{V}$ of $r = O(\frac{k}{\varepsilon^2} \log n)$ points. Then we have the following property for each box $\mathbf{b} \in \mathcal{B}$.

- (I) If $\overline{m}(\mathbf{b}) = \omega(\mathbf{b} \cap \mathcal{V})/\omega(\mathcal{B}) \ge (\varepsilon/20k)$ then $0.99\overline{m}(\mathbf{b}) \le \overline{s}(\mathbf{b}) \le 1.01\overline{m}(\mathbf{b})$.
- (II) If $\overline{m}(\mathbf{b}) \leq \varepsilon/20k$ then $\overline{s}(\mathbf{b}) \leq 1.01/20k$.

Therefore to check if a box is **b** is light, it suffices to check $|\mathbf{b} \cap \mathcal{R}|$. As for computing the sample \mathcal{R} , observe that this is exactly what batch sampling is designed for, see Lemma 2.7.

Analysis. The correctness of the new MWU follows from the previous one. We now analyze the running time. Each round except the last round doubles the weight of $\Omega(k/\log^{2d} n)$ boxes. Since the total number of weight-doubling operations performed by the algorithm is $O(p \log n)$, see [AP20], the algorithm stops within $O(\log^{2d} n)$ rounds. In a particular round, the algorithm uses batch sampling to recompute the random subset \mathcal{R} which it uses to identify the set $\mathcal{L} \subseteq \mathcal{B}$ of light boxes. Lemma 2.7 shows that the cost of batch sampling is bounded by $O(n^{d/2} \log^2 n)$ time. As discussed above, once \mathcal{R} is computed, \mathcal{L} can be identified in $O(n \log^{d-1} n)$ time. Finding an independent set $\mathcal{I} \subseteq \mathcal{L}$ using Lemma 2.4 also takes $O(n \log^{d-1} n)$ time. Updating the multi-set \mathcal{S} of boxes whose weights have been doubled so far also takes O(n) time. The algorithm computes a piercing set and a weak net only in the last round. Together, they take $O(n \log^d n)$ time. The running time for these steps is dominated by the the time for the batch sampling in each round. Considering the need to do an exponential search for p, we obtain the following:

Theorem 2.10. Let \mathcal{B} be set of n axis-aligned boxes in \mathbb{R}^d , for $d \ge 2$, and let $p = p(\mathcal{B})$ be the piercing number of \mathcal{B} . A piercing set of \mathcal{B} of size $O(p \log \log p)$ can be computed in $O(n^{d/2} \log^{2d+3} n)$ expected time.

3. Multi-round piercing algorithm

Let \mathcal{B} be a set of (closed) boxes in \mathbb{R}^d , and let $\mathcal{V} = \mathcal{V}(\mathcal{B})$ be the set of vertices of the arrangement $\mathcal{A}(\mathcal{B})$. When considering a piercing set for \mathcal{B} , one can restrict the selection of piercing points to points of \mathcal{V} . Two point sets $\mathcal{Q}, \mathcal{Q}'$ are *equivalent* for \mathcal{B} if for all faces (of all dimensions) f of the arrangement of \mathcal{B} , we have that $|\mathcal{Q} \cap f| = |\mathcal{Q}' \cap f|$. Since $|\mathcal{V}| = O(n^d)$, it follows that the number of non-equivalent (i.e., distinct) piercing sets of size $\leq t$ is bounded by $O(n^{dt})$.

The key insight is that a piercing set for a sufficiently large, but not too large, sample is a piercing set for almost all the boxes.

Lemma 3.1. Let \mathcal{B} be a set of n boxes in \mathbb{R}^d , $\delta \in (0,1)$ and t > 0 be parameters, and let $\mathcal{S} \subseteq \mathcal{B}$ be a random sample of size $O(\frac{dt}{\delta} \log n)$. If \mathcal{S} can be pierced by a set \mathcal{Q} of t points, then at most δn boxes of \mathcal{B} are not pierced by \mathcal{Q} , and this holds with probability $\geq 1 - 1/n^{O(d)}$.

Proof: Let \mathcal{F} be the collection of all piercing sets of \mathcal{B} with at most t points, where no two sets in \mathcal{F} are equivalent for \mathcal{B} . Let $\mathcal{F}' = \{X \in \mathcal{F} \mid |\mathcal{B} \setminus X| \geq \delta n\}$ be all the "bad" piercing set in \mathcal{F} that fail to stab $\geq \delta n$ boxes of \mathcal{B} , where $\mathcal{B} \setminus X = \{b \in \mathcal{B} \mid b \cap X = \emptyset\}$. By the above, we have that $m = |\mathcal{F}'| \leq O(n^{dt})$. Fix a "bad" set $X \in \mathcal{F}'$. Let $u = c\frac{dt}{\delta} \ln n$ be the size of $|\mathcal{S}|$, where c is a sufficiently large constant. The probability that the set X is a piercing set for the sample \mathcal{S} is at most

$$\psi = (1 - \delta)^u \le \exp(-\delta u) = \exp(-c \cdot d \cdot t \cdot \ln n) = \frac{1}{n^{cdt}}$$

In particular, by the union bound, the probability that any set of \mathcal{F}' will be a valid piercing set for \mathcal{S} is at most $|\mathcal{F}'| \psi < 1/n^{O(d)}$, for c sufficiently large.

3.1. A piercing algorithm via sampling

Lemma 3.1 suggests a natural algorithm for piercing – pick a random sample from \mathcal{B} , compute (or approximate) a piercing set for it, compute the boxes this piercing set misses, and repeat the process for several rounds. In the last round, hopefully, the number of remaining boxes is sufficient small than one can apply the piercing approximation algorithm directly to \mathcal{B} . We thus get the following.

Lemma 3.2. Let \mathcal{B} be a set of n boxes in \mathbb{R}^d , and let $\zeta > 1$ be a parameter. Furthermore, assume that we are given an algorithm **Pierce** that for m boxes, can compute a $O(\log \log p)$ approximate to their piercing set in $T_{\mathbf{P}}(m)$ time, where p is the size of the optimal piercing set for the given set. Then, one can compute a piercing set for \mathcal{B} of size $O(\zeta p \log \log p)$ in expected time

$$O\Big(\zeta T_{\mathbf{P}}\big(p^{1-1/\zeta}n^{1/\zeta}\log n\big)+n\log^{d-1}p\Big).$$

Proof: We assume that we have a number k such that $p \leq k \leq 2p$, where p is the size of the optimal piercing set for \mathcal{B} . To this end, one can perform an exponential search for this value, and it is easy to verify that this would not effect the running time of the algorithm.

The algorithm performs ζ rounds. Let $\mathcal{B}_0 = \mathcal{B}$. Let $\delta = (k/n)^{1/\zeta}$. In the *i*th round, for $i = 1, \ldots, \zeta - 1$, we pick a random sample \mathcal{S}_i from \mathcal{B}_{i-1} of size

$$m = O(dk\delta^{-1}\log n) = O(dk^{1-1/\zeta}n^{1/\zeta}\log n) = O(p^{1-1/\zeta}n^{1/\zeta}\log n).$$

In the ζ th round, we set $S_{\zeta} = \mathcal{B}_{\zeta-1}$. Now, we approximate the optimal piercing set for S_i , by calling $\mathcal{Q}_i \leftarrow \mathbf{Pierce}(S_i)$. If the piercing set \mathcal{Q}_i is too large – that is, $|\mathcal{Q}_i| \gg k \log \log k$, then the guess for k is too small, and the algorithm restarts with a larger guess for k. Otherwise, the algorithm builds a range tree for \mathcal{Q}_i , and streams the boxes of \mathcal{B}_{i-1} through the range tree, to compute the set $\mathcal{B}_i = \mathcal{B}_{i-1} \setminus \mathcal{Q}_i$, the boxes in \mathcal{B}_{i-1} not pierced by \mathcal{Q}_i . By Lemma 3.1, we have $|\mathcal{B}_i| \leq \delta |\mathcal{B}_{i-1}| \leq \delta^i n$ with high probability. If $|\mathcal{B}_i| > \delta |\mathcal{B}_{i-1}|$, we repeat round i, so assume $|\mathcal{B}_i| \leq \delta |\mathcal{B}_{i-1}|$. In particular $|\mathcal{S}_{\zeta}| = |\mathcal{B}_{\zeta-1}| \leq \delta^{\zeta-1}n \leq p^{1-1/\zeta}n^{1/\zeta}$. Therefore, the total time spent by **Pierce**(.) in ζ rounds is $O(\zeta T_{\mathbf{P}}(p^{1-1/\zeta}n^{1/\zeta}\log n))$. Finally, during the first $\zeta - 1$ iterations, computing \mathcal{B}_i takes

$$\sum_{i=1}^{\zeta-1} O(|\mathcal{B}_{i-1}|\log^{d-1}|\mathcal{Q}_i|) = \sum_{i=1}^{\zeta-1} O(\delta^i n \log^{d-1}(p \log \log p)) = O(n \log^{d-1} p)$$

time. Clearly, $\cup_i Q_i$ is the desired piercing set.

We can use the algorithm of Theorem 2.10, for the piercing algorithm. For this choice,

$$T_{\mathbf{P}}(m) = O(m^{d/2} \log^{2d+3} m).$$

We then get an approximation algorithm with running time

$$O\left(\zeta\left(p^{1-1/\zeta}n^{1/\zeta}\right)^{d/2}\operatorname{polylog}(n)+n\log^{d-1}p\right)$$

We thus get our second main result.

Theorem 3.3. Let \mathcal{B} be a set of n axis-aligned boxes in \mathbb{R}^d , for $d \ge 2$, and let $\zeta > 0$ be an integer. A piercing set of \mathcal{B} of size $O(\zeta p \log \log p)$ can be computed in

$$O\left(\zeta \boldsymbol{\mu}^{d/2-d/2\zeta} n^{d/2\zeta} \operatorname{polylog}(n) + n \log^{d-1} \boldsymbol{\mu}\right).$$

expected time, where p = p(B) is the size of the optimal piercing set.

The above algorithm provides a trade-off between the approximation factor and the running time. It readily leads to a near linear time algorithm if the piercing set is sufficiently small. For example, by choosing $\zeta = d$, we obtain the following:

Corollary 3.4. Let \mathcal{B} be a set of n axis-aligned boxes in \mathbb{R}^d for some fixed $d \geq 2$, and assume $p(\mathcal{B}) = O(n^{1/(d-1)})$. Then, a piercing set of \mathcal{B} of size $O(dp \log \log p)$ can be computed in $O(n \operatorname{polylog}(n))$ expected time.

If the piercing set is slightly sublinear, the above leads to an approximation algorithm with running time $O(n \log n)$.

Corollary 3.5. Let \mathcal{B} be a set of n axis-aligned rectangles in \mathbb{R}^2 for some fixed $d \ge 2$, and assume that it can be pierced by $p = O(n/\log^{15} n)$ points. Then, a piercing set of \mathcal{B} of size $O(p \log \log p)$ can be computed in $O(n \log p)$ expected time.

Proof: Pick a random sample $S \subseteq B$ of size $O(n/\log^7 n)$. The algorithm of Theorem 2.10 yields in O(n) time a piercing set Q for S, of size $u = O(p \log \log p)$. Preprocess Q for orthogonal range emptiness queries – this takes $O(p \log^2 p)$ time, and one can decide if a rectangle is not pierced by Q in $O(\log p)$ time. Lemma 3.1 implies that at most δn rectangles unpierced by Q, where

$$\delta = \frac{p \log^8 n}{n}.$$

Namely, the unhit set has size $\delta n = O(n/\log^7 n)$. Running time algorithm of Theorem 2.10 on this set of rectangles, takes O(n) time, and yields a second piercing set Q' of size $O(p \log \log p)$. Combining the two sets results in the desired piercing set.

4. Dynamic Algorithm for piercing

We present a data structure for maintaining a near-optimal piercing set for a set \mathcal{B} of boxes in \mathbb{R}^2 as boxes are inserted into or deleted from \mathcal{B} . By adapting the multi-round sampling based algorithm described in Section 3, we obtain a Monte Carlo algorithm that maintains a piercing set of $O(p \log \log p)$ size with high probability and that can update the piercing set in $O^*(n^{1/2})$ amortized expected time per update. (The $O^*()$ notation hides polylogarithmic factors). The update time can be improved to $O^*(n^{1/3})$ if \mathcal{B} is a set of squares in \mathbb{R}^2 .

Overview of the Algorithm. We observe that the size of the optimal piercing set changes by at most one when a box is inserted or deleted. We periodically reconstruct the piercing set using a faster implementation of the multi-round sampling based algorithm in Section 3, as described below. More precisely, if s is the size of the piercing set computed during the previous reconstruction, then we reconstruct the piercing set after $\lceil s/2 \rceil$ updates. To expedite the reconstruction, we maintain \mathcal{B} in a data structure as follows. We map a box $\mathbf{b} = [a_1, a_2] \times [b_1, b_2]$ to the point $\mathbf{b}^* = (a_1, b_1, a_2, b_2)$ in \mathbb{R}^4 , and let \mathcal{B}^* be the resulting set of points in \mathbb{R}^4 . We store \mathcal{B}^* into a 4-dimensional dynamic range tree T, which is a 4-level tree. Each node v of T is associated with a *canonical subset* $\mathcal{B}^*_v \subseteq \mathcal{B}^*$ of points. Let \mathcal{B}_v be the set of boxes corresponding to \mathcal{B}^*_v . For a box \Box in \mathbb{R}^4 , $\Box \cap \mathcal{B}^*$ can be represented as the union of $O(\log^4 n)$ canonical subsets, and they can be computed in $O(\log^4 n)$ time. The size of T is $O(n \log^4 n)$, and it can be updated in $O(\log^4 n)$ amortized time per insertion/deletion of point. See [BCK008].

Between two consecutive reconstructions, we use a lazy approach to update the piercing set, as follows: Let \mathcal{P} be the current piercing set. When a new box **b** is inserted, we insert it into T. If $\mathcal{P} \cap \mathbf{b} = \emptyset$, we choose an arbitrary point p inside **b** and add p to \mathcal{P} . When we delete a box **b**, we simply delete \mathbf{b}^* from T but do not update \mathcal{P} . If $\lceil s/2 \rceil$ updates have been performed since the last reconstruction, we discard the current \mathcal{P} and compute a new piercing set as described below.

We show below that a piercing set of size $s := O(p \log \log p)$ of \mathcal{B} can be constructed in

$$O^*\left((\boldsymbol{p}n)^{1/2} + \min\{\boldsymbol{p}^2, n\}\right)$$

expected time, where p is the size of the optimal piercing set of \mathcal{B} . This implies the amortized expected update time is $O^*((n/p)^{1/2} + \min\{p, n/p\})$, including the time spent in updating T. The second term is bounded by $n^{1/2}$, so the amortized expected update time is $O^*(n^{1/2})$.

Reconstruction algorithm. Here is how we construct the piercing set of boxes in \mathbb{R}^2 . Let \mathcal{B} be the current set of boxes. We follow the algorithm in Theorem 3.3 and set the number of rounds to 2. More precisely, perform an exponential search on the value of k, the guess for the size of the optimal piercing set, every time we reconstruct the piercing set. For a fixed k, the reconstruction algorithm consists of the following steps:

- (I) Choose a random sample \mathcal{B}_1 of \mathcal{B} of size $r = c_1(kn)^{1/2}$, where c_1 is a suitable constant.
- (II) Construct a piercing set \mathcal{P}_1 of \mathcal{B}_1 of size $s = O(k \log \log k)$ in $O^*(r)$ time using the algorithm in Section 2.2.
- (III) Compute $\mathcal{B}_2 \subseteq \mathcal{B}$, the subset of boxes that are not pierced by \mathcal{P}_1 . If $|\mathcal{B}_2| > c_2(kn)^{1/2}$, where c_2 is a suitable constant, we return to Step 1. As described below, this step can be computed in $O^*(\min\{k^2, n\} + (kn)^{1/2})$ time.
- (IV) Compute a piercing set \mathcal{P}_2 of \mathcal{B}_2 , again using the algorithm in Section 2.2.
- (V) Return $\mathcal{P}_1 \cup \mathcal{P}_2$.

The expected running time of this algorithm is $O^*(\min\{k^2, n\} + (kn)^{1/2})$, as desired.

Computing \mathcal{B}_2 . We now describe how to compute \mathcal{B}_2 efficiently using T. If $p \geq n^{1/2}$, then we simply preprocess \mathcal{P} into a 2-dimensional range tree in $O(s \log s)$ time. By querying with each box in \mathcal{B} , we can compute \mathcal{B}_2 in $O(n \log n)$ time [BCK008]. The total time spent is $O(n \log n)$. So assume $p < n^{1/2}$. For a point $p = (x_p, y_p) \in \mathbb{R}^2$, let $Q_p \subset \mathbb{R}^4$ be the orthant $Q_p = \{(x_1, x_2, x_3, x_4) \mid x_1 \leq x_p, x_2 \leq y_p, x_3 \geq x_p, x_4 \geq y_p\}$. Then, a box $b \subset \mathbb{R}^2$ contains p if and only if $\mathcal{B}^* \in Q_p$.

Therefore, an input box $\mathbf{b} \in \mathcal{B}$ is not pierced by \mathcal{P}_1 if $\mathbf{b}^* \notin \bigcup_{p \in \mathcal{P}} Q_p$. Let $\mathcal{K} = \mathbb{R}^4 \setminus \bigcup_{p \in \mathcal{P}} Q_p$. It is well known that the complexity of \mathcal{K} is $O(s^2)$. Furthermore, \mathcal{K} can be partitioned into $O(s^2)$ boxes with pairwise-disjoint interiors, as follows.

Let $\mathcal{Q} = \{Q_p \mid p \in \mathcal{P}\}$, and let $\hat{\mathcal{P}} = \{(x_p, y_p, x_p, y_p) \mid p \in \mathcal{P}\}$ be their corners. We sort $\hat{\mathcal{P}}$ by the x_4 -coordinates of its points. Let Δ be an x_4 -interval between the x_4 -coordinates of two consecutive points of \mathcal{P} . For any value $a \in \Delta$, the cross-section \mathcal{Q}_a of \mathcal{Q} with the hyperplane $h_a: x_4 = a$ is a collection of s 3-dimensional octants, and $\mathcal{K}_a := \mathcal{K} \cap h_a$ is the complement of the union of \mathcal{Q}_a . Furthermore, the cross-section \mathcal{K}_a remains the same for any value of $a \in \Delta$. It is well known that the complexity of \mathcal{K}_a is O(s), and that it can be partitioned into a set \mathcal{R}_a of 3-dimensional boxes in $O^*(s)$ time. Hence, we can partition \mathcal{K} inside the slab $\mathbb{R}^3 \times \Delta$ by the set $\mathcal{R}_\Delta = \{R \times \Delta \mid R \in \mathcal{R}_a\}$. By repeating this procedure for all x_4 -intervals between two consecutive points of $\hat{\mathcal{P}}$, we partition \mathcal{K} into a family \mathcal{R} of $O(s^2)$ boxes.

Next, we query T with each box $R \in \mathcal{R}$. The query procedure returns a set V_R of $O(\log^4 n)$ nodes of T such that $\mathcal{B}^* \cap R = \bigcup_{v \in V_R} \mathcal{B}^*_v$. We thus obtain a set V of $O(s^2 \log^4 n)$ nodes of T such that $\mathcal{B}_2^* = \bigcup_{v \in V} \mathcal{B}_v^*$. If $\sum_{v \in V} |\mathcal{B}_v^*| \le c_2(kn)^{1/2}$, we return $\bigcup_{v \in V} \mathcal{B}_v$ as \mathcal{B}_2 . Otherwise, we return NULL. The total time spent by this procedure is $O^*(\min\{n, k^2\} + (kn)^{1/2})$. Putting everything together

we obtain the following.

Theorem 4.1. A set \mathcal{B} of n boxes in \mathbb{R}^2 can be stored in a data structure so that a piercing set of \mathcal{B} of size $O(p \log \log p)$ can be maintained with high probability under insertion and deletion of boxes with amortized expected time $O(n^{1/2} \operatorname{polylog}(n))$ per insertion or deletion; p is the piercing number of \mathcal{B} .

Dynamic algorithm for squares in 2D. If we have squares instead of boxes, then the reconstruction time reduces to $O^*(p^{2/3}n^{1/3})$, which leads to an amortized update time of $O^*((n/p)^{1/3}) =$ $O^*(n^{1/3})$. We proceed in a similar manner as before. There are two differences. First, we now choose a random sample of size $O(k^{2/3}n^{1/3})$, and the algorithm works in three rounds. After the first round, we have a piercing set \mathcal{P}_1 of size $O(p \log \log p)$, and we need to represent the set of squares not pierced by \mathcal{P}_1 as $O^*(p)$ canonical subsets, so that we can choose a random sample \mathcal{B}_2 from this subset of squares. After the second round, we have a piercing set \mathcal{P}_2 of \mathcal{B}_2 of size $O(p \log \log p)$. Finally, we find the subset $\mathcal{B}_3 \subseteq \mathcal{B}$ of squares not pierced by $\mathcal{P}_1 \cup \mathcal{P}_2$ and compute a piercing set of \mathcal{B} . It suffices to describe how we compactly represent the set \mathcal{B}_2 .

We map a square, which is centered at a point c and of radius (half side length) a, to the point $b^* = (c, a) \in \mathbb{R}^3$. A point $p = (x_p, y_p) \in \mathbb{R}^2$ is now mapped to the cone $C_p = \{(x, y, z) \in \mathbb{R}^3 \mid z \in \mathbb{R}^3 \}$ $\|(x,y)-p\|_{\infty} \leq z, z \geq 0\}$ with the square cross-section and p its apex; C_p is the graph of the L_{∞} -distance function from p. It is easily seen that $p \in \mathbf{b}$ if and only if $\mathbf{b}^* \in C_p$. Hence, a box $\mathbf{b} \in \mathcal{B}_2$ if it lies below all the cones of $C = \{C_p \mid p \in \mathcal{P}_1\}$. Using a 3D orthogonal range-searching data structure, we can compute \mathcal{B}_2 as the union of $O^*(k)$ canonical subsets. We omit the details from here and obtain the following.

Theorem 4.2. A set \mathcal{B} of n squares in \mathbb{R}^2 can be stored in the data structure described above so that a piercing set of \mathcal{B} of size $O(p \log \log p)$ can be maintained with high probability under insertion and deletion of boxes in $O(n^{1/3} \operatorname{polylog}(n))$ amortized expected time per insertion or deletion.

5. Data structure for maintaining weights of boxes

In this section, we describe how to implement the data structure needed for our algorithms whose specifications are given in Definition 2.1. Let \mathcal{B} be a set of axis-aligned boxes in \mathbb{R}^d . For simplicity, we assume the boxes in \mathcal{B} are in general position. Let \mathcal{C} be a set of *active* boxes (initially empty) and let $\mathcal{V}(\mathcal{C})$ be the set of vertices of the arrangement $\mathcal{A}(\mathcal{C})$. Let $\mathcal{S} \subseteq \mathcal{B}$ be a multi-set of *update* boxes (initially empty). For a point $p \in \mathcal{V}$, recall that the *doubling weight* of p is defined to be $w_{\mathcal{S}}(p) := 2^{|\mathcal{S} \sqcap p|}$, where let $\mathcal{S} \sqcap p = \{\mathbf{b} \in \mathcal{S} \mid p \in \mathbf{b}\}$ is the multi-set of all boxes in \mathcal{S} containing p. We require a data-structure that supports the following operations:

- (I) weight(b): given a box b, computes $w_{\mathcal{S}}(\mathcal{V}(\mathcal{C}) \cap b)$.
- (II) double(b): given a box $b \in \mathcal{B}$, adds a copy of b to the multi-set of update boxes \mathcal{S} .
- (III) halve(b): given a box $b \in \mathcal{B}$, removes a copy of b from the multi-set of update boxes \mathcal{S} .
- (IV) sample: returns a random point $p \in \mathcal{V}(\mathcal{C})$ with probability $w_{\mathcal{S}}(p)/w_{\mathcal{S}}(\mathcal{V}(\mathcal{C}))$.
- (V) **insert**(b): inserts b into the set of active boxes C.
- (VI) delete(b): removes b from the set of active boxes C.

For a set $Z \subseteq \mathbb{R}^d$ and $i \in \llbracket d \rrbracket$, let $Z_{\downarrow x_i} = \{x_i \mid (x_1, \ldots, x_d) \in Z\}$ be the **projection** of Z to the x_i -axis. For a multiset S of boxes in \mathbb{R}^d and $i \in \llbracket d \rrbracket$, let $S_{\downarrow x_i} = \{\mathbf{b}_{\downarrow x_i} \mid \mathbf{b} \in S\}$ be the multiset of intervals resulting from projecting S on the x_i -axis.

We first describe the data structure for the line in Section 5.1, then for the plane in Section 5.2, and finally extend it to higher dimensions in Section 5.3.

5.1. Data structure in 1D

A segment tree with minor tweaking provides the desired data-structure for 1D, as described next.

The set of vertices \mathcal{V} is simply the endpoints of the intervals of \mathcal{B} . We construct the segment tree T of \mathcal{B} , where the endpoints of \mathcal{B} are stored at the leaves of T (thus, a node v in T corresponds to an "interval" which is the set of all endpoints stored in its subtree). The idea is to update the weights in a lazy fashion — for completeness we describe this in detail, but this is by now folklore.

We modify the segment tree so that each internal node v has a total weight $\omega(v)$ of all the vertices in its subtree, and a count $\alpha(v)$. In any given point in time, if v has two children x and y, we have that $\omega(v) = 2^{\alpha(v)}(\omega(x) + \omega(y))$. The value of $\alpha(v)$ is pushed down to its children in a lazy fashion. Specifically, whenever the algorithm traverses from a node v to one of its children, it adds the value of $\alpha(v)$ to $\alpha(x)$ and $\alpha(y)$, and sets $\alpha(v)$ to zero. It immediately also recomputes $\omega(x)$ and $\omega(y)$. Similarly, whenever the traversal returns from a child, the weight of the parent node is recomputed.

For the operation weight(b), the data-structure locates the $O(\log n)$ nodes of T such that their (disjoint) union covers the interval **b**, and it returns the sum of their weights (note, that this propagates down the values of $\alpha(\cdot)$ at all the ancestors of these nodes to them – thus, all the ancestors have $\alpha(\cdot)$ with value zero). The operation **double(b)** (resp. **halve(b)**) works in a similar fashion, except that the data-structure increases (resp. decreases) the $\alpha(\cdot)$ of the all the $O(\log n)$ nodes of T covering **b** by one.

The operation **sample** performs a random traversal down the tree. If the traversal is at node v, with children x and y, it continues to x with probability $\omega(x)/\omega(v)$, and otherwise it continues to y. The function returns the endpoint stored in the leaf where the traversal ends.

Finally, if we want to support **insert** and **delete**, we initially set the weight of endpoints associated with each interval to zero. When an interval $(a, b) \in \mathcal{B}$ is inserted to the set of active intervals \mathcal{C} , we locate the leaves corresponding to a and b and set their weights to one. We then traverse up the tree from the leaves recomputing $\omega(\cdot)$ for the ancestors of either leaf. The deletion sets the weight of the endpoints to zero and proceeds analogously. Therefore, we obtain the following:

Lemma 5.1. Let \mathcal{B} be a set of n intervals in \mathbb{R}^1 . A data-structure of size $O(n \log n)$ can be constructed in $O(n \log n)$ time that supports each of the operations described in Definition 2.1 in $O(\log n)$ time.

5.2. Data Structure in the plane

Basic idea. Let \mathcal{B} be the set of input rectangles, and let φ be an axis-parallel box that does not contain a vertex of a rectangle of \mathcal{B} in its interior. Any edge e of a rectangle $\mathbf{b} \in \mathcal{B}$ that intersects the interior of φ , cuts it into two pieces by the line supporting e. Let X (resp. Y) be the set of all the x-values (resp. y-values) of all edges that are orthogonal to the x-axis (resp. y-axis) and that intersect φ . Clearly, the vertices of \mathcal{V} in the interior of φ are formed by the Cartesian product $X \times Y$ (we refer to this set somewhat informally, as a grid). Weights on such a grid of vertices can be maintained implicitly by maintaining weights on the sets X and Y separately (using the one dimensional data-structure). It is easy to verify that all the operations of Definition 2.1 decompose into these one-dimensional data-structures.

Thus, the data-structure is going to decompose the arrangement of \mathcal{B} into cells, where the vertices inside each cell would be represented by such a grid. Importantly, each box would appear directly in only $O(\sqrt{n})$ grids. A technicality is that update box might contain many such cells in their interior (and thus their grids are contained inside the update box). Fortunately, by storing these cells in an appropriately constructed tree, such updates could be handled by implicit updates on the nodes of this tree.

Partitioning scheme. Let \mathcal{B} be a set of axis-aligned boxes in \mathbb{R}^2 , and let \Box be a rectangle that contains all boxes of \mathcal{B} in its interior. We first partition \Box into $\lceil 2\sqrt{n} \rceil$ vertical slabs, i.e., by drawing y-parallel edges, so that each slab contains at most \sqrt{n} vertical edges of boxes in \mathcal{B} . Next, we further partition each slab σ into $O(\sqrt{n})$ rectangles by drawing horizontal edges as follows. If a corner (vertex) ξ of a box of \mathcal{B} lies in σ , we partition σ by drawing a horizontal edge passing through ξ . Finally, if a rectangle ϱ in the subdivision of σ intersects more than \sqrt{n} horizontal edges of \mathcal{B} , we further partition ϱ into smaller rectangles by drawing horizontal edges, so that it intersects at most \sqrt{n} horizontal edges. Let Π be the resulting rectangular subdivision of \Box . Observe that Π has O(n) cells. By construction, no vertex of \mathcal{B} lies in the interior of a cell of Π , the boundary of any box intersects $O(\sqrt{n})$ cells of Π , and each cell φ intersects at most $2\sqrt{n}$ edges of \mathcal{B} . For a cell φ of Π , let $\mathcal{V} \cap \varphi$ be the set of all vertices of $\mathcal{V} = \mathcal{V}(\mathcal{B})$ that lie in the *interior* of φ . Observe that we have $\mathcal{V} \cap \varphi = (\mathcal{V} \cap \varphi)_{\downarrow x} \times (\mathcal{V} \cap \varphi)_{\downarrow y}$.

Weights decompose in a cell. Consider a multiset $S_{\varphi} \subseteq \mathcal{B}$ that intersect the interior of a cell φ . The multiset S_{φ} can be partitioned into two multisets of intervals:

$$\mathcal{X}_{\varphi} = \{ \mathsf{b}_{\downarrow x} \mid \mathsf{b} \in \mathcal{S}_{\varphi} \text{ and } \varphi_{\downarrow x} \not\subset \mathsf{b}_{\downarrow x} \} \qquad \text{and} \qquad \mathcal{Y}_{\varphi} = \{ \mathsf{b}_{\downarrow y} \mid \mathsf{b} \in \mathcal{S}_{\varphi} \text{ and } \varphi_{\downarrow y} \not\subset \mathsf{b}_{\downarrow y} \}.$$

Note, that a box $\mathbf{b} \in S_{\varphi}$ contributes an interval (more formally, its multiplicity of intervals) to exactly one of these sets, as no box of S_{φ} can have a vertex in the interior of φ .

Lemma 5.2. Let φ be a cell of Π , $S_{\varphi} \subseteq \mathcal{B}$ a multiset of rectangles whose boundaries intersect the interior of φ , and let b be a rectangle. For $U = \mathcal{V} \cap \varphi \cap \mathsf{b}$, $X = U_{\downarrow x}$ and $Y = U_{\downarrow y}$, we have that $w_{S_{\varphi}}(U) = w_{\mathcal{X}_{\varphi}}(X) \cdot w_{\mathcal{Y}_{\varphi}}(Y)$,

Proof: Since no box $\mathbf{b} \in S_{\varphi}$ has a vertex in the interior of φ , it follows that either the vertical or the horizontal edges of \mathbf{b} intersect φ (but not both). As such, we have $U = X \times Y$. Furthermore, for $p = (x, y) \in U$, we have that $|p \sqcap S_{\varphi}| = |x \sqcap \mathcal{X}_{\varphi}| + |y \sqcap \mathcal{Y}_{\varphi}|$, which implies that $w_{S_{\varphi}}(p) = w_{\mathcal{X}_{\varphi}}(x) \cdot w_{\mathcal{Y}_{\varphi}}(y)$. As such, we have

$$w_{\mathcal{S}_{\varphi}}(U) = \sum_{p \in U} w_{\mathcal{S}_{\varphi}}(p) = \sum_{(x,y) \in X \times Y} w_{\mathcal{X}_{\varphi}}(x) \cdot w_{\mathcal{Y}_{\varphi}}(y) = \sum_{x \in X} w_{\mathcal{X}_{\varphi}}(x) \cdot \sum_{y \in Y} w_{\mathcal{Y}_{\varphi}}(y) = w_{\mathcal{X}_{\varphi}}(X) \cdot w_{\mathcal{Y}_{\varphi}}(Y).$$



Figure 5.1: $\Delta = I_x \times I_y$ denotes a cell in Π . $\mathcal{S}_{\varphi} = \{R_1, R_2, R_3, R_4\}$ intersects Δ . Let $R_i = R_i^x \times R_i^y$ for $i \in \{1, 2, 3, 4\}$. $\mathcal{X}_{\Delta} = \{R_2^x, R_3^x, R_4^x\}$, $\mathcal{Y}_{\Delta} = \{R_1^y, R_5^y\}$. Observe that $w_{\mathcal{S}_{\varphi}}(\{p_1\}) = w_{\mathcal{X}_{\Delta}}(\{p_1\}_{\downarrow x}) \cdot w_{\mathcal{Y}_{\Delta}}(\{p_1\}_{\downarrow y}) = 16$ and $w_{\mathcal{S}_{\varphi}}(\{p_2\}) = w_{\mathcal{X}_{\Delta}}(\{p_2\}_{\downarrow x}) \cdot w_{\mathcal{Y}_{\Delta}}(\{p_2\}_{\downarrow y}) = 32$.

Tree data structure. We construct a balanced binary tree T storing the cells of Π in the leaves – the top levels form a balanced binary tree over the slabs, say from left to right. A "leaf" that stores a slab is then the root of a balanced binary tree on the cells within the slab, with cells ordered from bottom to top. Each node u of T is associated with a rectangle \Box_u . The root is associated with \Box itself and each leaf is associated with the corresponding cell of Π . For an internal node u with children w and z, $\Box_u = \Box_w \cup \Box_z$. Let $\mathcal{V}_u := \mathcal{V} \cap \operatorname{int}(\Box_u)$. If u is an internal node with w and z as children and e being the common edge of \Box_w and \Box_z (i.e. $e = \Box_w \cap \Box_z$), then $\mathcal{V}_u = \mathcal{V}_w \cup \mathcal{V}_z \cup (\mathcal{V} \cap e)$. If e does not lie in the horizontal edge of a rectangle of \mathcal{B} then $\mathcal{V} \cap e = \emptyset$. If $e \in \partial \mathbf{b}$ for some rectangle $\mathbf{b} \in \mathcal{B}$ then $\mathcal{V}_u \cap e$ is the set of intersection points of $e \cap \partial \mathbf{b}$ with the x-edges of \mathcal{B} that intersect e.



Figure 5.2: The figure represents the partition Π and the tree T built on Π . σ is a strip of Π , Δ is a cell of σ . T_{σ} is the tree built on the cells of σ . R is a rectangle in \mathcal{B} . The highlighted nodes represent $C(\mathsf{b})$. (Note that the axes are flipped.)

Let $\mathbf{b} = \mathbf{b}_x \times \mathbf{b}_y$ be an arbitrary rectangle. The rectangle \mathbf{b} is **long**, at a node u, if $\Box_u \subseteq \mathbf{b}$, and **short** if $\Box_u \cap \partial \mathbf{b} \neq \emptyset$. A node u is **canonical** for \mathbf{b} , if (i) u is a leaf and \mathbf{b} is short at u, or (ii) if \mathbf{b} is long at u but short at p(u), the parent of u in T. Let $C(\mathbf{b})$ denote the set of canonical nodes for \mathbf{b} and let $C^*(\mathbf{b})$ be the set of ancestors of nodes in $C(\mathbf{b})$ including $C(\mathbf{b})$ itself. By the construction of Π , \mathbf{b} is short at $O(\sqrt{n})$ leaves of T, therefore $|C(\mathbf{b})| = |C^*(\mathbf{b})| = O(\sqrt{n} \log n)$.

For simplicity, in the subsequent discussion we assume that all boxes are active. Notice that the construction of T is not affected by the set of active boxes. Later, we describe how to insert or delete from the set of active boxes C while preserving the invariants of the data structure. Recall that S is the multi-set of update boxes. For a node $u \in T$, let $S_u \subseteq S$ (resp. $\mathcal{L}_u \subseteq S$) be the set of update boxes that are short (resp. long) at u. Note that u is a canonical node for all boxes in $\mathcal{L}_u \setminus \mathcal{L}_{p(u)}$ and also for S_u if u is a leaf. We maintain two values at each node u of T:

- (i) $\lambda_u = |\mathcal{L}_u \setminus \mathcal{L}_{p(u)}|$: the number of boxes of \mathcal{S} that are long at u but short at p(u); and
- (ii) $\omega_u = w_{\mathcal{S}_u}(\mathcal{V}_u)$: the total weight of points in \mathcal{V}_u with respect to the update boxes that are short at u.

The following inequalities follow easily from the definitions for any rectangle b:

$$w_{\mathcal{S}}(\mathcal{V}_u) = w_{\mathcal{S}_u}(\mathcal{V}_u) \times w_{\mathcal{L}_u}(\mathcal{V}_u) = \omega_u \times \prod_{v \in \operatorname{anc}(u)} 2^{\lambda_v}$$
(5.1)

$$w_{\mathcal{S}}(\mathcal{V}_u \cap \mathsf{b}) = w_{\mathcal{S}_u}(\mathcal{V}_u \cap \mathsf{b}) \times \prod_{v \in \operatorname{anc}(u)} 2^{\lambda_v},$$
(5.2)

where $\operatorname{anc}(u)$ is the set of ancestors of u, including u itself. Let u be an internal node of T with children y and z, and let $e_u = \Box_y \cap \Box_z$ be the common edge shared by \Box_y and \Box_z . Then, for any rectangle **b**, we have

$$\omega_u = \omega_y 2^{\lambda_y} + \omega_z 2^{\lambda_z} + w_{\mathcal{S}_u} (\mathcal{V} \cap e_u) \tag{5.3}$$

and
$$w_{\mathcal{S}_u}(\mathcal{V}_u \cap \mathsf{b}) = w_{\mathcal{S}_y}(\mathcal{V}_w \cap \mathsf{b})2^{\lambda_y} + w_{\mathcal{S}_z}(\mathcal{V}_z \cap \mathsf{b})2^{\lambda_z} + w_{\mathcal{S}_u}(\mathcal{V} \cap e_u),$$
 (5.4)

If e_u is a vertical edge then $\mathcal{V} \cap e_u = \emptyset$, so the last term is positive only if e_u is a horizontal edge of a cell in Π .

By Eq. (5.3), ω_u can be maintained recursively if the value at the leaves of T is known, and one can compute $w_{\mathcal{S}_u}(\mathcal{V} \cap e_u)$ efficiently. To this end, we use the "lazy segment tree" data structure from Section 5.1, as secondary data structures, to maintain ω_u at the leaves. The same data structure will also be used to maintain $w_{\mathcal{S}_u}(\mathcal{V} \cap e_u)$.

Let z be a leaf of T, and let \Box_z be the cell in Π corresponding to z. Let $X = (\Box_z \cap \mathcal{V})_{\downarrow x}$ and $Y = (\Box_z \cap \mathcal{V})_{\downarrow y}$. Note that, $|X| = |Y| = O(\sqrt{n})$ because only $O(\sqrt{n})$ rectangle boundaries intersect the interior of \Box_z . Then we construct lazy segment trees Ψ_z^x and Ψ_z^y on X, and Y respectively. Recall that $S_z \subseteq S$ is the set of update rectangles that are short at z. We can decompose S_z into the following multi-sets of intervals:

$$\mathcal{X}_z = \{ \mathsf{b}_{\downarrow x} \mid \mathsf{b} \in \mathcal{S}_z \text{ and } (\Box_z)_{\downarrow x} \not\subset \mathsf{b}_{\downarrow x} \} \qquad \text{and} \qquad \mathcal{Y}_z = \{ \mathsf{b}_{\downarrow y} \mid \mathsf{b} \in \mathcal{S}_z \text{ and } (\Box_z)_{\downarrow y} \not\subset \mathsf{b}_{\downarrow y} \}.$$

By Lemma 5.2, $\omega_z = w_{\mathcal{S}_z}(\mathcal{V}_z) = w_{\mathcal{X}_z}(X) \cdot w_{\mathcal{Y}_z}(Y)$. Hence, we can maintain ω_z by maintaining Ψ_z^x and Ψ_z^y . Furthermore, for each internal node u with children v, w such that \Box_u lies in a slab of Π , we also have a lazy segment tree Ψ_u^e for handling the vertices in $\mathcal{V} \cap e$ where $e = \Box_v \cap \Box_w$. We omit the details from here.

This completes the description of the data structure. The tree T, as well as the secondary data structures at all leaves and relevant internal nodes of T can be constructed and initialized in $O(n^{3/2} \log n)$ time.

Operations on \mathcal{T} . We now briefly describe how to perform the operations mentioned in Section 2.1 using the data structure.

• weight(b): Given a rectangle $\mathbf{b} = \mathbf{b}_x \times \mathbf{b}_y$, the goal is to return $w_{\mathcal{S}}(\mathcal{V}_{\rho} \cap \mathbf{b})$ where ρ is the root of T. We start at the root node and visit T in a top-down manner to find the set of canonical nodes in T with respect to \mathbf{b} . Then, starting from the canonical set of nodes we visit T bottom-up computing $w_{\mathcal{S}_u}(\mathcal{V}_u \cap \mathbf{b})$ for each node u we visit. In particular, for each internal node u we visit, if \mathbf{b} is long at u then $w_{\mathcal{S}_u}(\mathcal{V}_u \cap \mathbf{b}) = \omega_u$. If \mathbf{b} is short at u, and w, z are its children then,

$$w_{\mathcal{S}_u}(\mathcal{V}_u \cap \mathsf{b}) = 2^{\lambda_w} w_{\mathcal{S}_w}(\mathcal{V}_w \cap \mathsf{b}) + 2^{\lambda_z} w_{\mathcal{S}_z}(\mathcal{V}_z \cap \mathsf{b}) + w_{\mathcal{S}_u}(\mathcal{V} \cap e)$$

where $e = \Box_w \cap \Box_z$. We recursively compute each of the terms. If u is a leaf node, we compute $w_{\mathcal{S}_u}(\mathcal{V}_u \cap \mathbf{b})$ by calling **weight** (\mathbf{b}_x) and **weight** (\mathbf{b}_y) on Ψ_u^x and Ψ_u^y , respectively and taking their product. Lemma 5.2 guarantees correctness in this case. In the case of the internal nodes, correctness follows from Eq. (5.1), Eq. (5.2), Eq. (5.3), and Eq. (5.4).

• double(b): Given a rectangle $\mathbf{b} = \mathbf{b}_x \times \mathbf{b}_y \in \mathcal{B}$, we first add a copy of \mathbf{b} to the set of update boxes \mathcal{S} . We then need to modify the data structure to ensure all the invariants are preserved. We start at the root of T and we traverse T in a top-down manner. If \mathbf{b} is long at u, we increment λ_u by 1. If \mathbf{b} is short at u and u is an internal node with children w and z, and $e = \Box_w \cap \Box_z$, we first update Ψ_u^e and then recurse on w and z. If we reach a leaf u, and vertical (resp. horizontal) edges of \mathbf{b} intersect \Box_u , we call double(\mathbf{b}_x) (resp. double(\mathbf{b}_y)) on Ψ_u^x (resp. Ψ_u^y). We also update the values of ω_u for every node u we visit. It is easy to verify that the operation preserves the invariants captured by Eq. (5.1), Eq. (5.2), Eq. (5.3), and Eq. (5.4).

- halve(b): Given a rectangle b = b_x × b_y ∈ B, we proceed analogously to the way we do for double. We start at the root of T and traverse in a top-down manner. If b is long at u, we decrement λ_u by 1. If b is short at u and u is an internal node with children w and z and e = □_w ∩ □_z, we first update Ψ^e_u and then recurse on w and z. If we reach a leaf u, and vertical (resp. horizontal) edges of b intersect □_u, we call halve(b_x) (resp. halve(b_y)) on Ψ^x_u (resp. Ψ^y_u).
- sample: We start at the root of T and we traverse T in a top-down manner. For a node u with children w, z, we pick one of w, z and $e = \Box_w \cap \Box_z$ with probabilities $w_{\mathcal{S}}(\mathcal{V} \cap \Box_w), w_{\mathcal{S}}(\mathcal{V} \cap \Box_z),$ and $w_{\mathcal{S}}(\mathcal{V} \cap e)$ respectively. These probabilities can easily be computed using the secondary data structures and the values maintained at each node, as done in the weight operation. If e is picked, we then use the sample operation on the secondary data structure Ψ_z^e to sample a point from $\mathcal{V} \cap e$. If w or z is picked we recurse on it. If at any stage we reach a leaf l, we pick a and b by calling sample on Ψ_l^x and Ψ_l^y , respectively, and return the point (a, b).
- insert(b): Given a rectangle $\mathbf{b} = \mathbf{b}_x \times \mathbf{b}_y$, we first add a copy of \mathbf{b} to the set of active boxes C. Although a new active box does not affect weights of existing vertices, it can add new vertices thereby making the value of ω_u inconsistent for a node u. We proceed in the same way we do for **double**, except we do not modify the λ_u values and use the **insert** operation on the secondary data structures. We also update the values of ω_u for every node u we visit. It is easy to verify that the operation preserves the invariants captured by Eq. (5.1), Eq. (5.2), Eq. (5.3), and Eq. (5.4).
- delete(b): Given a rectangle $b = b_x \times b_y$, we first remove a copy of b to the set of active boxes C. The rest of the procedure is the same as **insert**, except we use the **delete** operation on the secondary data structures in the leaves.

Running time. As discussed above, a rectangle **b** has at most $O(\sqrt{n} \log n)$ canonical nodes. Hence, to compute **weight**(·), **double**(·), **halve**(·), **insert**(·), or **delete**(·) operations we need to visit $O(\sqrt{n} \log n)$ nodes. Also, $O(\sqrt{n})$ secondary data structure operations are performed each of which takes $O(\log n)$ time. Hence, the running time for these operations is bounded by $O(\sqrt{n} \log n)$. For **sample**, it is clear that we visit $O(\log n)$ nodes and perform a total of O(1) secondary data structure operations. Hence, **sample** takes $O(\log n)$ time.

5.3. Extending to higher dimensions

The basic idea of 2D can be extended to higher dimensions. Namely, let φ be an axis-aligned box that does not contain any (d-2)-face of a box $\mathbf{b} \in \mathcal{B}$. If a facet f of a box $\mathbf{b} \in \mathcal{B}$ intersects the interior of φ then f splits φ into two boxes by the hyperplane supporting f. For i = 1, ..., d, let X_i be the x_i -values of the facets of the boxes in \mathcal{B} that are orthogonal to the x_i -axis and that intersect φ . Then $\mathcal{V}(\mathcal{B}) \cap \varphi = X_1 \times ... \times X_d$. We thus extend the 2D data structure to higher dimensions.

Partitioning scheme. To extend the above data structure to \mathbb{R}^d , for d > 2, we generalize the partition scheme described above. The partition scheme works recursively on the dimensions – first applying a hyperplane cut every time \sqrt{n} faces orthogonal to the first axis are encountered. Now, we project on the boxes that intersect a slab into its boundary hyperplane. We construct a partition on this hyperplane, applying the construction in \mathbb{R}^{d-1} , and then lift the partition back to the slab. We now briefly describe the specifics.

Let \Box be a box containing all of \mathcal{B} . A facet of a box is an *i-side* if it is normal to the x_i -axis.

We first partition \Box into $\lceil 2\sqrt{n} \rceil$ slabs, by drawing hyperplanes parallel to the 1-sides of the boxes so that each slab contains at most \sqrt{n} 1-sides of boxes in \mathcal{B} . These are *level-1* slabs. Next, we further partition each level-1 slab by drawing hyperplanes along the 2-sides. If a corner (vertex) ξ of a box of \mathcal{B} lies in σ , we partition σ by drawing a hyperplane parallel to the 2-side passing through ξ . Finally, if a cell φ in the subdivision of σ intersects more than \sqrt{n} 2-sides of \mathcal{B} , we further partition φ by drawing hyperplanes parallel to the 2-sides, so that it intersects at most \sqrt{n} 2-sides. The above steps partition each level-1 slab σ into $O(\sqrt{n})$ level-2 slabs. Next, we repeat the previous steps for each level-2 slab γ , i.e. we partition it using hyperplanes along the 3-sides such that no cell in the resulting subdivision contains more than $O(\sqrt{n})$ 3-sides of boxes in \mathcal{B} . We continue these steps on the resulting level-3 slabs and so on. For more details see [OY91].

Let Π be the resulting partition. By construction, Π is a subdivision of \Box into boxes. We have $|\Pi| = O(n^{d/2})$, no vertex of \mathcal{B} lies in the interior of a cell of Π , the boundary of any box intersects at most $O(n^{(d-1)/2})$ cells of Π , and each cell intersects at most $O(\sqrt{n})$ faces of \mathcal{B} . Borrowing the definition from [OY91], for a cell $\varphi \in \Pi$, a box $\mathbf{b} \in \mathcal{B}$ is an *i-pile* with respect to φ if $\partial \mathbf{b} \cap \varphi \neq \emptyset$ and for all $j \neq i$, the *j*th interval of \mathbf{b} spans the *j*th interval of φ . The partition ensures that if a box $\mathbf{b} \in \mathcal{B}$, has $\partial \mathbf{b} \cap \operatorname{int}(\varphi) \neq \emptyset$ then it is an *i*-pile with respect to φ for some *i*.

For a cell $\varphi \in \Pi$, let E^i_{φ} denote the set of *i*-sides of \mathcal{B} that intersect φ , clipped within φ . Let $Z^i_{\varphi} = (\mathcal{V} \cap \varphi)_{\downarrow x_i}$ be the set of x_i -coordinates of the *i*-sides in E^i_{φ} . Let $\mathcal{V}_{\varphi} \subseteq \mathcal{V}$ be the subset of vertices that lie in the interior of φ . Then $\mathcal{V}_{\varphi} = \{e_1 \cap \cdots \cap e_d \mid e_1 \in \mathsf{E}^1_{\varphi}, \ldots, e_d \in \mathsf{E}^d_{\varphi}\}$. The following is the straightforward extension of Lemma 5.2 to higher dimensions.

Lemma 5.3. Let φ be a cell of Π , let $S_{\varphi} \subseteq \mathcal{B}$ be a multiset of boxes whose boundaries intersect φ , let $\mathcal{I}_{\varphi}^{i} = \{\mathbf{b}_{\downarrow x_{i}} \mid \mathbf{b} \in S_{\varphi} \text{ and } \varphi_{\downarrow x_{i}} \not\subset \mathbf{b}_{\downarrow x_{i}}\}$ be the multiset of x_{i} -projections of boxes in S_{φ} whose *i*-sides intersect φ . For an arbitrary box \mathbf{b} , we have

$$w_{\mathcal{S}_{\varphi}}(\mathcal{V}_{\varphi} \cap \mathsf{b}) = \prod_{i=1}^{d} w_{\mathcal{I}_{\varphi}^{i}} (Z_{\varphi}^{i} \cap \mathsf{b}_{\downarrow x_{i}}).$$
(5.5)

We build a *d*-dimensional tree T on Π analogously to the 2D case. Let z be a leaf of T, and let \Box_z be the cell corresponding to z. We construct a segment tree Ψ_z^i on $Z_{\Box_z}^i$ for each $i \in \{1, .., d\}$. The operations of the data structure as well as the analysis are identical to the 2D case and hence we omit the details. We get the following lemma.

Lemma 5.4. Let \mathcal{B} be a set of n axis-aligned rectangles in \mathbb{R}^d . A data-structure can be constructed in $O(n^{(d+1)/2} \log n)$ time that supports every operation specified in Definition 2.1.

6. Weak ε -net for boxes

Let \mathcal{P} be a set of n points in \mathbb{R}^d , and let $\varepsilon \in (0, 1)$ be a parameter. We describe an algorithm for choosing a set $\mathcal{N} \subset \mathbb{R}^d$ of $O(\varepsilon^{-1} \log \log \varepsilon^{-1})$ points such that for any box b in \mathbb{R}^d with $|\mathsf{b} \cap \mathcal{P}| \ge \varepsilon |\mathcal{P}|$, $\mathcal{B} \cap \mathcal{N} \neq \emptyset$. For $d \le 3$, the algorithm ensures that $\mathcal{N} \subseteq \mathcal{P}$, i.e. it computes an ε -net. Our overall approach is similar to [AES09, Ezr10], but we believe our algorithm is simpler. The algorithm works in three stages. The first stage chooses a set $\mathcal{X} \subseteq \mathcal{P}$ of size $n = O(\varepsilon^{-1} \log \varepsilon^{-1})$ points so that for any box b with $|\mathsf{b} \cap \mathcal{P}| \ge \varepsilon n$, we have $|\mathsf{b} \cap \mathcal{X}| \ge \frac{\varepsilon}{2}n$. It thus suffices to pierce all $\frac{\varepsilon}{2}$ -heavy boxes with respect to \mathcal{X} . The second stage computes a set C of $O(n \log^t n)$ canonical boxes, for some constant $t \ge 1$, such that for any box b , if $|\mathcal{X} \cap \mathsf{b}| \ge \frac{\varepsilon}{2}n$ then there exists a box $\mathsf{b}' \in C$ such that $\mathsf{b}' \subseteq \mathsf{b}$ and $|\mathsf{b}' \cap \mathcal{X}| \geq \frac{\varepsilon}{2^{d+1}} n$. Finally, we construct a set \mathcal{N} of $O(\varepsilon^{-1} \log \log \varepsilon^{-1})$ points that pierces every $\frac{\varepsilon}{2^{d+1}}$ -heavy box of C with respect to \mathcal{X} . Putting these steps together, we obtain a weak ε -net of \mathcal{P} of size $O(\varepsilon^{-1} \log \log \varepsilon^{-1})$ with respect to boxes. The second stage is the most involved, so we postpone the construction of C to the end.

6.1. Reducing the number of points

The following lemma follows from the known bound on relative ε -approximation mentioned above but we give a direct proof here.

Lemma 6.1. Let \mathcal{P} be a set of n points in \mathbb{R}^d , let $\omega : \mathcal{P} \to \mathbb{R}_{\geq 0}$ be a weight function, let $\varepsilon \in (0, 1)$ be a parameter, and let $\mathcal{N} \subseteq \mathcal{P}$ be a random subset of size $n = O(\varepsilon^{-1} \log \varepsilon^{-1})$. Then for every box \mathbf{b} with $\omega(\mathbf{b} \cap \mathcal{P}) \geq \varepsilon \omega(\mathcal{P})$, we have $|\mathbf{b} \cap \mathcal{N}| \geq (\varepsilon/2)n$. This holds with probability at least $1 - \varepsilon^{O(d)}$.

(Here the sampling is done with repetition, and the probability of a point to be chosen is proportional to its weight.)

Proof: For simplicity, we prove this lemma for the unweighted case. It easily extends to the weighted case. Let H_i be a minimal set of hyperplanes orthogonal to the *i*th axis, such that there are at most $\varepsilon n/(16d)$ points of \mathcal{P} between two consecutive hyperplanes of H_i . We refer to the region lying

between two adjacent hyperplanes of H_i as a *slab*. Set $H = \bigcup_{i=1}^{d} H_i$. $|H| = O(d\varepsilon^{-1})$ where the constant hiding in the O-notation is independent of d. H can be computed in $O(n \log \varepsilon^{-1})$ time.

We refer to a box whose all 2d facets lie on the hyperplanes of H as a *canonical* box. Given a box h that contains at least εn points of \mathcal{P} let by be the largest canonical box that is

Given a box **b** that contains at least εn points of \mathcal{P} , let \mathbf{b}_H be the largest canonical box that is contained in **b**, i.e. \mathbf{b}_H is obtained by shrinking **b** so that its facets lie on H. Since $\mathbf{b} \setminus \mathbf{b}_H$ can be covered by 2d slabs of H, and each slab contains at most $\frac{\varepsilon n}{16d}$ points of \mathcal{P} , it follows that

$$0 \le |\mathsf{b} \cap \mathcal{P}| - |\mathsf{b}_H \cap \mathcal{P}| \le \frac{\varepsilon}{8}n$$

Let \mathcal{B} be the set of all $\frac{7\varepsilon}{8}$ -heavy canonical boxes. Observe that $|\mathcal{B}| = O(|H|^{2d}) = O((d\varepsilon^{-1})^{2d})$. Let \mathcal{N} be a random sample of \mathcal{P} with repetition of $n = c\varepsilon^{-1}\log\varepsilon^{-1}$ points for some constant c > 1. (Technically \mathcal{N} is a multiset, but for simplicity assume it to be a set.)

Consider any box $\mathbf{b}_H \in \mathcal{B}$, and let $\alpha = \frac{|\mathbf{b}_H \cap \mathcal{P}|}{\varepsilon n} \geq \frac{7}{8}$. Let $Y = |\mathcal{N} \cap \mathbf{b}_H|$. The variable Y is a sum of n random binary variables, each being 1 with probability $\frac{|\mathbf{b}_H \cap \mathcal{P}|}{n} = \alpha \varepsilon$. Therefore,

$$\mu = \mathbf{E}[Y] = \alpha \varepsilon n \ge \frac{7}{8} \varepsilon n$$

Using Chernoff inequality², we have

$$\begin{split} \gamma &= \mathbf{Pr} \Big[Y \leq \frac{\varepsilon}{2} n \Big] \leq \mathbf{Pr} \Big[Y \leq (1 - \frac{3}{7}) \frac{7}{8} \varepsilon n \Big] \\ &\leq \mathbf{Pr} \Big[Y \leq (1 - \frac{3}{7}) \mu \Big] \leq \exp \left(-\mu \frac{(3/7)^2}{2} \right) \\ &\leq \exp \left(-\frac{7}{8} \cdot \frac{(3/7)^2}{2} \cdot \varepsilon n \right) \leq \exp \left(-\frac{c}{14} \log \frac{1}{\varepsilon} \right) \\ &\leq \varepsilon^{-c/14}, \end{split}$$

²Specifically, $\mathbf{Pr}[Y < (1-\delta)\mathbf{E}[Y]] \le \exp(-(\delta^2/2)\mathbf{E}[Y]).$

Hence, the probability of a box in \mathcal{B} containing $\frac{\varepsilon}{2}n$ points of \mathcal{N} is at most $|\mathcal{B}|\gamma \leq \varepsilon^{-O(d)}$ assuming $c = \Theta(d \log d)$. Hence, with probability at least $1 - \varepsilon^{-O(d)}$, \mathcal{N} has the desired property.

Remark 6.2. The above lemma provides us with a stronger result than just an ε -net, it shows that a sample of size $\Theta(\varepsilon^{-1} \log \varepsilon^{-1})$ has the property that any ε -heavy box contains at least $\Theta(\log \varepsilon^{-1})$ points of the sample. On the other hand, Lemma 6.1 proves a weaker property than the one we get from relative approximations [HS11] but this weaker property is sufficient for our purposes.

Mapping the points to a grid. Assume that

$$n = 2^{\ell} = \alpha \varepsilon^{-1} \log \varepsilon^{-1}, \quad \text{for some integer} \quad \ell = \log n = O(\log \varepsilon^{-1}).$$
 (6.1)

For simplicity, assume that all the points of \mathcal{Q} have unique coordinates. In particular, for a point $p \in \mathcal{Q}$, let $G(p) = (i_1, i_2, \ldots, i_d)$ be the integral point, where i_j is the rank of the *j*th coordinate of *p* among the *n* values of the points of \mathcal{Q} in this coordinate. Consider the "gridified" point set $\mathcal{X} = G(\mathcal{Q}) = \{G(p) \mid p \in \mathcal{Q}\}$. Clearly, it is enough to construct a weak $\varepsilon/2$ -net for \mathcal{X} , as we can then map it back to a net for \mathcal{Q} .

6.2. Piercing heavy canonical boxes

We defer the construction of a set C of $O(n \log^t n) O(\frac{\varepsilon}{2^{d+1}})$ -heavy canonical boxes to Section 6.3, but describe an efficient algorithm for constructing a piercing set of C of size $O(\varepsilon^{-1} \log \log \varepsilon^{-1})$.

Lemma 6.3. Let \mathcal{X} be a set of $\mathbf{n} = O(\varepsilon^{-1} \log \varepsilon^{-1})$ points in \mathbb{R}^d , and let \mathcal{F} be a set of $s = O(\varepsilon^{-1} \log^t \varepsilon^{-1})$ boxes in \mathbb{R}^d , for some constant t, such that for each $\mathbf{b} \in \mathcal{F}$, $|\mathbf{b} \cap \mathcal{X}| \ge (\varepsilon/2^{d+1})\mathbf{n}$. A set $\mathcal{N} \subseteq \mathcal{X}$ of size $O(\varepsilon^{-1} \log \log \varepsilon^{-1})$ that pierces all the boxes of \mathcal{F} can be computed in $O(\mathbf{n} \log \mathbf{n} + s \log^{d-1} \varepsilon^{-1})$ time, and furthermore $|\mathbf{b} \cap \mathcal{N}| = \Omega(\log \log \varepsilon^{-1})$ for every $\mathbf{b} \in \mathcal{F}$.

Proof: Pick a random sample \mathcal{N}_1 with repetition from \mathcal{X} of size $m = \alpha \varepsilon^{-1} \log \log \varepsilon^{-1}$, where $\alpha > 0$ is a sufficiently large constant. For a specific box $\mathbf{b} \in \mathcal{F}$,

$$\tau := \Pr[\mathbf{b} \cap \mathcal{N}_1 \neq \emptyset] = \left(1 - \frac{|\mathbf{b} \cap \mathcal{X}|}{|\mathcal{X}|}\right)^m \le \exp\left(-\frac{\varepsilon}{2^{d+1}}m\right) = \exp\left(-\frac{\alpha}{2^{d+1}}\log\log\varepsilon^{-1}\right)$$
$$< \frac{1}{\log^{2t}\varepsilon^{-1}},$$

by picking α to be sufficiently large. By linearity of expectations, the expected number of boxes in \mathcal{F} not pierced by \mathcal{N}_1 is at most $\tau s = O((\varepsilon^{-1} \log^t \varepsilon^{-1})/\log^{2t} \varepsilon^{-1}) = O(\varepsilon^{-1}/\log^t \varepsilon^{-1})$. We now add a point from \mathcal{X} to each box of \mathcal{F} that is not pierced by \mathcal{N}_1 to a new set \mathcal{N}_2 . Let $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$. Clearly, $\mathbf{E}[|\mathcal{N}|] = O(\varepsilon^{-1} \log \log \varepsilon^{-1})$, and it pierces all boxes of \mathcal{F} . By preprocessing \mathcal{N}_1 into an orthogonal range searching data structure, the subset of boxes of \mathcal{F} not pierced by \mathcal{N}_1 can be computed in $O((m+s)\log^{d-1}m)$ time [Aga04]. Again using a range searching data structure, \mathcal{N}_2 can be computed in $O(\tau s \log^{d-1} n)$ expected time.

The stronger claim follows by working a little harder. Let **b** be any box in \mathcal{F} , and consider $Z = |\mathbf{b} \cap \mathcal{N}_1|$. The variable Z is a sum of m random independent binary variables, with the probability of each of them being 1 is at least $\frac{\varepsilon}{2d+1}$. Therefore,

$$\mu = \mathbf{E}[Z] \geq \frac{\varepsilon}{2^{d+1}} \cdot m$$

As such, by Chernoff's bound,

$$\tau' = \mathbf{Pr}\left[Z < \frac{\varepsilon m}{2^{d+2}}\right] \le \mathbf{Pr}[Z < \mu/2] \le \exp(-\mu/8) < \frac{1}{\log^{2t} \varepsilon^{-1}},$$

for α sufficiently large constant. Let $\mathcal{F}^{<} = \{ \mathbf{b} \in \mathcal{F} \mid |\mathbf{b} \cap \mathcal{N}_{1}| < \frac{\varepsilon}{2^{d+2}}m \}$. Arguing as above, the expected size of $\mathcal{F}^{<}$ is at most

$$\tau' |\mathcal{F}| = O(\varepsilon^{-1} / \log^t \varepsilon^{-1}).$$
(6.2)

For each box $\mathbf{b} \in \mathcal{F}^{<}$, we pick at most $\frac{\varepsilon}{2^{d+2}}m$ points from $\mathbf{b} \cap \mathcal{X}$ into a new set \mathcal{N}'_2 so that $|\mathbf{b} \cap \mathcal{X}| \geq \frac{\varepsilon}{2^{d+2}}m = \Omega(\log \log \varepsilon^{-1}).$

Set $\mathcal{N}' = \mathcal{N}_1 \cup \mathcal{N}'_2$. Since $\mathbf{E}[|\mathcal{F}^<|] = O(\varepsilon^{-1}/\log^t \varepsilon^{-1}),$

$$\mathbf{E}[|\mathcal{N}'|] = O(\varepsilon^{-1}\log\log\varepsilon^{-1}) + O(\log\log\varepsilon^{-1}) \cdot \mathbf{E}[|\mathcal{F}^{<}|] = O(\varepsilon^{-1}\log\log\varepsilon^{-1})$$

Clearly, for every $\mathbf{b} \in \mathcal{F}$, $|\mathbf{b} \cap \mathcal{N}'| = \Omega(\log \log \varepsilon^{-1})$.

6.3. Constructing canonical boxes

Given \mathcal{X} , it is easy to construct a set of $O(n^{2d}) = \varepsilon^{-O(1)}$ canonical boxes (each facet containing a point of \mathcal{X}), but we show a much smaller canonical set is sufficient. In particular, we describe the construction of a set of $O(n \log^t n) \frac{\varepsilon}{2d+1}$ -heavy canonical boxes mentioned above.

Construction for d = 2. We first describe a simple construction for d = 2, and then a different construction in higher dimensions.

Lemma 6.4. Let \mathcal{X} be a set of n points in the plane. For a parameter k > 0, a set \mathcal{F} of $O(n \log n)$ boxes, each of them containing exactly k points of \mathcal{X} , can be computed such that any rectangle b that contains at least 4k points of \mathcal{X} fully contains one of the rectangles of \mathcal{F} .

Proof: Let ℓ be a horizontal line that partitions \mathcal{X} into two equal sets. For every point of $p \in \mathcal{X}$, consider the vertical segment connecting it to its projection on ℓ . We translate this "curtain" segment e to the left until the rectangle R swept by e contains k points, and we add R to \mathcal{F} . We compute a similar rectangle by translating e to the right. We now apply the construction recursively on the points of \mathcal{X} above (resp. below) ℓ . This procedure results in a family of \mathcal{F} with at most $O(n \log n)$ rectangles each containing exactly k points.

Let **b** be a rectangle with $|\mathbf{b} \cap \mathcal{X}| \ge 4k$. Let ℓ be the first line in the recursive construction that intersects **b**. Assume that $\mathbf{b} \cap \mathcal{X}$ contains more points above ℓ than below it. Let **b'** be the portion of **b** above ℓ . The top edge of **b'** contains a point $p \in \mathcal{X}$. Let *s* be the vertical segment connecting *p* to its projection on ℓ . The segment *s* partitions **b'** into two closed rectangles, one of them must contain at least $|\mathbf{b}' \cap \mathcal{P}|/2 \ge |\mathbf{b} \cap \mathcal{P}|/4 \ge k$ points. In particular, at least one of the two rectangles in \mathcal{F} , induced by (p, ℓ) , must be contained in **b'**, and thus in **b**, thus implying the claim.

Construction for $d \ge 3$. The set of canonical boxes in higher dimensions is constructed in two steps.

Definition 6.5. Let $\llbracket \ell \rrbracket = \{0, 1, 2, \dots, \ell\}$, see Eq. (6.1). Given $i = (i_1, \dots, i_d) \in \llbracket \ell \rrbracket^d$, we associate with it the box $\mathbf{b}_i = [0, 2^{i_1}] \times [0, 2^{i_2}] \times \dots \times [0, 2^{i_d}]$. Such a box naturally tiles the bounding cube $[0, n]^d \supset \mathcal{X}$ into a grid. Let H_i denote this grid, and let $\widehat{\mathcal{B}}$ be the set of all boxes that appear as grid cells in any of these grids.



Figure 6.1: Illustration of the proof of Lemma 6.6.

Note, that the number of different grids we have is $n_H = (\ell + 1)^d = O(\log^d \varepsilon^{-1}).$

Lemma 6.6. Let **b** be an $\varepsilon/2$ -heavy box for \mathcal{X} (i.e., $|\mathbf{b} \cap \mathcal{X}| \ge (\varepsilon/2)n$), where $n = |\mathcal{X}|$. There exists a witness grid box $\mathbf{c} \in H_i$, for some $i \in \llbracket h \rrbracket^d$, such that

(i) $|\mathbf{b} \cap \mathbf{c} \cap \mathcal{X}| \ge (\varepsilon/2^{d+1})n$, and

(ii) b contains a corner of c, and c contains a corner of b.

Proof: Let $c_0 = [0, n]^d$ be the initial cell. We partition c_0 into half by a hyperplane orthogonal to the first axis. This results in two grid cells c'_1 and c''_1 . If one of them contains **b**, then we continue the argument recursively on the cell containing **b** – namely, we repeatedly half the cell along the first axis until the partitioning hyperplane intersects **b**. If $|c'_1 \cap b \cap \mathcal{X}| \ge |c''_1 \cap b \cap \mathcal{X}|$ then we set $c_1 = c'_1$ (otherwise, $c_1 = c''_1$).

We now continue applying this argument to c_1 but on the second dimension (again, halving until hitting b). We repeat this process for all d coordinates. Clearly, $|c_d \cap b \cap \mathcal{X}| \ge |b \cap \mathcal{X}|/2^d$, as we lose a factor of two when moving to the next dimension. It is easy to verify that c_d has the required properties.

Let \mathcal{S} be a set of m points in \mathbb{R}^d lying in a box \mathbf{g} . For an integer $k \ge 0$, a box $\mathbf{e} \subseteq \mathbf{g}$ is a k-crate if \mathbf{e} shares a vertex of \mathbf{g} , $|\operatorname{int}(\mathbf{e}) \cap \mathcal{S}| \le k$, and it is maximal under containment, where $\operatorname{int}(\mathbf{e})$ denotes the interior of \mathbf{e} . We refer to a 0-crate simply as a crate. Let $\mathcal{T}_k(\mathcal{S}, \mathbf{g})$ denote the set of k-crates with respect to \mathcal{S} and \mathbf{g} . By Lemma 6.6, an $\frac{\varepsilon}{2}$ -heavy box \mathbf{b} contains a $\frac{\varepsilon}{2^{d+2}}$ -crate $\mathbf{e} = \mathbf{b} \cap \mathbf{g}$, where $\mathbf{g} \in \widehat{\mathcal{B}}$ is a grid cell. As such, it suffices to pierce all $\frac{\varepsilon}{2^{d+2}}$ -crates for boxes of $\widehat{\mathcal{B}}$. The following lemma bounds the number of k-crates in \mathbb{R}^3 .

Lemma 6.7 (A). Let S be a set of m points lying in a box c in \mathbb{R}^3 , Then (i) $\mathcal{T}_0(S, c) = O(m)$ (ii) $\mathcal{T}_k(S, c) = O(mk^2)$

Proof: Although (i) is well known, we sketch a proof for the sake of completeness. We assume **c** has a corner at the origin, and it is contained in the positive octant. Consider a crate **b** with a vertex at the origin. It must have three points of S on its faces not-adjacent to the origin, and let p and q be the two points that maximizes the x and y coordinates, respectively, out of these three points. Projecting **b**, p and q to the xy-plane results in a rectangle **b'**, having points p' and q' on its boundary. We use the portion of $\partial \mathbf{b}'$ connecting p' and q' that does not go through the origin (its an L shaped curve) and add it to a set Γ . In the end of the process, we have a (degenerate

drawing) of a planar graph, induced on the vertices of S' (the projection of S to the *xy*-plane), with the edges of Γ . Importantly, the curves of Γ do not cross (but they can share "legs"), and there are no parallel edges in Γ . It follows immediately from Euler's formula that Γ has linear size, which also bounds the number of crates adjacent to this corner of c. Applying the same claim to the other corners of C readily implies the claim.

As for (ii), since a k-crate is defined by three points of S (that lie on the faces of the crate), the Clarkson-Shor technique for bounding the number of $\leq k$ -sets in a point set implies the number of k-crates is $O(m(k+1)^2)$.

Lemma 6.8. For d = 3, let $k = \lfloor (\varepsilon/2^5)n \rfloor = O(\log \varepsilon^{-1})$, and let $\mathcal{T} = \bigcup_{\mathbf{c} \in \widehat{\mathcal{B}}} \mathcal{T}_k(\mathcal{X} \cap \mathbf{c}, \mathbf{c})$ be the set of all k-crates induced by $\widehat{\mathcal{B}}$ (see Definition 6.5). Then $|\mathcal{T}| = O(\varepsilon^{-1}\log^6 \varepsilon^{-1})$.

Proof: Since every grid has at most $O(nk^2)$ crates, and there are $O(\log^3 \varepsilon^{-1})$ grids, we conclude that $|\mathcal{T}| = O(nk^2\log^3 \varepsilon^{-1}) = O(\varepsilon^{-1}\log^6 \varepsilon^{-1})$.

For d = 3, the above set \mathcal{T} is the desired set of heavy boxes that we want to pierce. However, for d = 4, the number of crates is $\Theta(n^2)$. The example in four dimensions follows readily by taking an example with n/2 crates in two dimensions (anchored at the origin) using n/2 points, and duplicating it on the other two dimensions. Clearly, the number of crates in this product four dimensional space is $\geq (n/2)^2$. This is why we cannot choose k-crates over all grid cells for $d \geq 4$ as the desired set of canonical boxes. Instead, we consider crates of "sparse" grid cells. In particular, we call a cell $\mathbf{c} \in \widehat{\mathcal{B}}$ massive if it contains at least $w = \ell^{d+2} = \Theta(\log^{d+2} \varepsilon^{-1})$ points of \mathcal{X} .

Observe that for a fixed grid there are $\leq n/w$ massive grid cells. Over all grids there are $O((n/w)n_H) = O(\varepsilon^{-1})$ massive cells. We can pierce all the crates that arise from massive cells, by simply adding the vertices of the massive cells to the new (weak) net. So we consider k-crates for non-massive grid cells. To this end, let \mathcal{L} be the set of all non-massive grid cells in $\widehat{\mathcal{B}}$. Clearly, we have

$$n_1 := |\mathcal{L}| = O(\ell^d n) = O(\varepsilon^{-1} \log^{d+1} \varepsilon^{-1}).$$

A grid cell that is not massive, has at most w points. As such, it has at most $n_2 = 2^d {w \choose d} = O(\log^{2d^2} \varepsilon^{-1}) k$ -crates, where $k = \lfloor (\varepsilon/2^{d+1})n \rfloor$.

Lemma 6.9. Let $k = \lceil (\varepsilon/2^{d+1})n \rceil$, and let \mathcal{T} be the set of all the k-crates that are in grid cells of $\widehat{\mathcal{B}}$ that contains at most $w = \Theta(\log^{d+2} \varepsilon^{-1})$ points of \mathcal{X} . We have that $|\mathcal{T}| = O(\varepsilon^{-1}\log^{3d^2} \varepsilon^{-1})$.

Proof: By the above, we have $|\mathcal{T}| \leq n_1 n_2 = O(\varepsilon^{-1} \log^{3d^2} \varepsilon^{-1}).$

6.4. Putting it together

Lemma 6.4 (for d = 2) and Lemma 6.8 (for d = 3) imply that there exists a set C of canonical boxes such that any $\frac{\varepsilon}{2}$ -heavy box **b** with respect to \mathcal{X} contains a box $\mathbf{b}' \in C$ with $|\mathbf{b}' \cap \mathcal{X}| \geq \frac{\varepsilon}{2^{d+1}}n$. Furthermore $|C| = O(\varepsilon^{-1} \log^{O(1)} \varepsilon^{-1})$. Hence, by combining this with Lemma 6.3, we obtain the following.

Theorem 6.10. Given a set \mathcal{P} of n points in \mathbb{R}^d for d = 2 or 3, and a parameter $\varepsilon \in (0, 1)$, a subset $\mathcal{N} \subseteq \mathcal{P}$ of size $O(\varepsilon^{-1} \log \log \varepsilon^{-1})$ can be computed in $O(n \operatorname{polylog} n)$ time that is an ε net for \mathcal{P} with respect to boxes. A somewhat stronger property holds – for any ε -heavy box \mathbf{b} , $|\mathbf{b} \cap \mathcal{N}| = \Omega(\log \log \varepsilon^{-1})$. *Proof:* For d = 2, the proof follows immediately from Lemma Lemma 6.4. For d = 3, let **b** be a box $\frac{\varepsilon}{2}$ heavy box with respect to \mathcal{X} . Then by Lemma 6.6, **b** has a witness $\mathbf{c} \in H_i$, for some $i \in [\![\ell]\!]^3$, such that $\mathbf{b}' = \mathbf{b} \cap \mathbf{c}$ contains at least $\varepsilon/2^4$ fraction of the points of \mathcal{X} . Namely, **b**' contains fully a k-crate of \mathcal{X} defined by \mathbf{c} , where $k = (\varepsilon/16)n = O(\log \varepsilon^{-1})$. Namely, it is enough to stab all the crates of \mathcal{T} . We can now deploy the argument of Lemma 6.3 to these k-crates, as their number is bounded by $O(\varepsilon^{-1}\log^6 \varepsilon^{-1})$.

For $d \geq 4$, we cannot choose a subset of \mathcal{X} because of the large number of k-crates. Instead we choose the vertices of massive grid cells and construct crates for only non-massive cells. The vertices of massive cells are not points of \mathcal{P} and we only obtain a weak ε -net..

Theorem 6.11. Given a set \mathcal{P} of n points in \mathbb{R}^d , and a parameter ε , a set \mathcal{N} of $O(\frac{1}{\varepsilon} \log \log \frac{1}{\varepsilon})$ points can be computed in $O(n + \frac{1}{\varepsilon} \log^{O(d^2)} \frac{1}{\varepsilon})$ time, such that every ε -heavy box **b** with respect to \mathcal{P} , contains at least one point of \mathcal{N} .

Proof: We follow the same steps as the previous constructions, reducing the task to computing $\varepsilon/2$ -net for \mathcal{X} . As suggested above, we add the corners of the massive cells to the computed net – this requires adding $O(\varepsilon^{-1})$ points to the weak net, and let \mathcal{N}_1 be this set. We only have to worry about crates that arise from "light" grid cells.

We next compute a piercing set $\mathcal{N}_2 \subseteq \mathcal{X}$ for all these crates using Lemma 6.3 (for \mathcal{X} and \mathcal{T}) of size $O(\varepsilon^{-1} \log \log \varepsilon^{-1})$. Clearly $\mathcal{N}_1 \cup \mathcal{N}_2$ is the desired weak ε -net for boxes.

The running time readily follows by first sampling \mathcal{P} to compute the set \mathcal{X} , and then computing for each grid cell that is not massive its set of k-crates. The sampling of the net \mathcal{N}_2 , and the fixup can then be done in time proportional to the size of the set system, which is as stated.

6.5. An improved algorithm for weak ε -net

Lemma 6.12. Given a set \mathcal{P} of n points in \mathbb{R}^d , a parameter $\varepsilon \in (0, 1)$, and a set \mathcal{B} of m boxes in \mathbb{R}^d that all contain at least εn points of \mathcal{P} . Then, a weak ε -net \mathcal{N} of size $O(\frac{1}{\varepsilon} \log \log \frac{1}{\varepsilon})$ that pierces every box $\mathbf{b} \in \mathcal{B}$ can be computed in $O(m \log^d \frac{1}{\varepsilon})$ expected time.

Proof: If $m \leq 1/\varepsilon$, then we pick a point from each box of \mathcal{B} , and we are done. Otherwise, we roughly follow the above construction. We first sample a set \mathcal{Q} of size $n = O(\varepsilon^{-1} \log \varepsilon^{-1})$, and compute \mathcal{X} from it, as done above. This takes $O(\varepsilon^{-1} \log^2 \varepsilon^{-1})$ time. Next, we map every box of \mathcal{B} to the appropriate box in $[0, n]^d$. We shrink it so that its corners all have integral coordinates. This takes $O(m \log \varepsilon^{-1})$ time. Let \mathcal{B}' be the resulting set of boxes. Next, we choose a (second) sample \mathcal{N}_2 from \mathcal{X} of size $O(\varepsilon^{-1} \log \log \varepsilon^{-1})$. Using range tree built for \mathcal{N}_2 , we compute all the boxes in \mathcal{B}' not being hit by \mathcal{N}_2 . Let \mathcal{B}'' be this set of boxes. This takes $O((m + \varepsilon^{-1}) \log^{d-1} \varepsilon^{-1})$ time [Aga04]. We compute for each box of \mathcal{B}'' its witness using the constructive algorithm of Lemma 6.6. This requires at most $O(d \log \varepsilon^{-1})$ orthogonal range counting queries, which overall takes $O((md \log \varepsilon^{-1} + \varepsilon^{-1}) \log^{d-1} \varepsilon^{-1}) = O(m \log^d \varepsilon^{-1})$ time.

For each unstabled box **b**, let **c** be its witness. If **c** is massive, we add its vertices to the weak ε -net. Otherwise, we compute any k-crate **c'** of **c** that is contained in **b** \cap **c** and pick any point of $\mathcal{X} \cap \mathbf{c'}$ to the net. Computing the crate itself can be implemented using binary search on each of the d-coordinates - so the crate is canonical and contains a point of \mathcal{X} on its one of its faces not attached to the grid cell vertex. The last step requires $O(d\log \varepsilon^{-1})$ orthogonal range searching query per

failed non-massive witness. Thus, the total running time is thus $O(m \cdot (d \log \varepsilon^{-1}) \cdot \log^{d-1} \varepsilon^{-1}) = O(m \log^d \varepsilon^{-1})$. A minor technicality is that when encountering a crate that was encountered before, we can skip stabbing it, as a point to this effect was already added to he net.

The resulting set is a weak ε -net as it is just a different implementation of the construction done above. The overall running time is $O(m \log^d \varepsilon^{-1})$.

Remark 6.13. Throughout this section we used unweighted points. However, Lemma 6.1 only requires access to the point set via sampling $O(\varepsilon^{-1} \log \varepsilon^{-1})$ points (with repetition, proportional to the weights of the points). In particular, all the results and algorithms of this section holds for weighted point sets.

References

- [ACS+22] P. K. Agarwal, H. Chang, S. Suri, A. Xiao, and J. Xue. Dynamic geometric set cover and hitting set. *ACM Trans. Algorithms*, 18(4): 40:1–40:37, 2022.
- [AES09] B. Aronov, E. Ezra, and M. Sharir. Small-size epsilon-nets for axis-parallel rectangles and boxes. Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009, 639–648, 2009.
- [Aga04] P. K. Agarwal. Range searching. Handbook of Discrete and Computational Geometry.
 Ed. by J. E. Goodman and J. O'Rourke. 2nd. Boca Raton, FL, USA: CRC Press LLC, 2004. Chap. 36, pp. 809–838.
- [AK92] N. Alon and D. J. Kleitman. Piercing convex sets. Proc. 8th Annu. Sympos. Comput. Geom. (SoCG), 157–160, 1992.
- [AKS97] P. K. Agarwal, M. J. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. Proceedings of the 9th Canadian Conference on Computational Geometry, Kingston, Ontario, Canada, August 11-14, 1997,
- [AP20] P. K. Agarwal and J. Pan. Near-linear algorithms for geometric hitting sets and set covers. *Discrete Comput. Geom.*, 63(2): 460–482, 2020.
- [BCK008] M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars. Computational geometry: algorithms and applications. 3rd. Springer, 2008.
- [BG95] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. Discrete Comput. Geom., 14(4): 463–479, 1995.
- [BMR18] N. Bus, N. H. Mustafa, and S. Ray. Practical and efficient algorithms for the geometric hitting set problem. *Discret. Appl. Math.*, 240: 25–32, 2018.
- [CC09] P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009, 892–901, 2009.
- [CH12] T. M. Chan and S. Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete Comput. Geom.*, 48(2): 373–392, 2012.
- [CH20] T. M. Chan and Q. He. Faster approximation algorithms for geometric set cover. *Proc.* 36th Int. Annu. Sympos. Comput. Geom. (SoCG), vol. 164. 27:1–27:14, 2020.

- [CH21] T. M. Chan and Q. He. More dynamic data structures for geometric set cover with sublinear update time. 37th International Symposium on Computational Geometry, SoCG 2021, June 7-11, 2021, Buffalo, NY, USA (Virtual Conference), vol. 189. 25:1– 25:14, 2021.
- [Cha03] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. J. Algorithms, 46(2): 178–189, 2003.
- [CHSX22] T. M. Chan, Q. He, S. Suri, and J. Xue. Dynamic geometric set cover, revisited. Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022, 3496–3528, 2022.
- [CM05] T. M. Chan and A. Mahmood. Approximating the piercing number for unit-height rectangles. *Proc. 17th Canad. Conf. Comput. Geom.* (CCCG), 15–18, 2005.
- [CSZ18] M. Chudnovsky, S. Spirkl, and S. Zerbib. Piercing axis-parallel boxes. Electron. J. Comb., 25(1): 1, 2018.
- [Ede87] H. Edelsbrunner. Algorithms in combinatorial geometry. Vol. 10. Springer Science & Business Media, 1987.
- [EKNS97] A. Efrat, M. J. Katz, F. Nielsen, and M. Sharir. Dynamic data structures for fat objects and their applications. Algorithms and Data Structures, 5th International Workshop, WADS '97, Halifax, Nova Scotia, Canada, August 6-8, 1997, Proceedings, vol. 1272. 297–306, 1997.
- [Ezr10] E. Ezra. A note about weak epsilon-nets for axis-parallel boxes in d-space. *Inf. Process. Lett.*, 110(18-19): 835–840, 2010.
- [FPT81] R. J. Fowler, M. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Inf. Process. Lett.*, 12(3): 133–137, 1981.
- [GKM+22] W. Gálvez, A. Khan, M. Mari, T. Mömke, M. R. Pittu, and A. Wiese. A 3-approximation algorithm for maximum independent set of rectangles. Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022, 894–905, 2022.
- [GRS98] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA, 73–84, 1998.
- [Har04] S. Har-Peled. Clustering motion. Discrete Comput. Geom., 31(4): 545–565, 2004.
- [Har11] S. Har-Peled. *Geometric Approximation Algorithms*. Vol. 173. Math. Surveys & Monographs. Boston, MA, USA: Amer. Math. Soc., 2011.
- [HM85] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. J. Assoc. Comput. Mach., 32(1): 130–136, 1985.
- [HS11] S. Har-Peled and M. Sharir. Relative (p, ε) -approximations in geometry. Discrete Comput. Geom., 45(3): 462–496, 2011.

- [Kar72] R. M. Karp. Reducibility among combinatorial problems. Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA, 85–103, 1972.
- [KLR+23] A. Khan, A. Lonkar, S. Rahul, A. Subramanian, and A. Wiese. Online and dynamic algorithms for geometric set cover and hitting set. 39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA, vol. 258. 46:1-46:17, 2023.
- [Mit21] J. S. B. Mitchell. Approximating maximum independent set for rectangles in the plane. 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022, 339–350, 2021.
- [MR10] N. H. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Discrete Comput. Geom.*, 44(4): 883–895, 2010.
- [Nie00] F. Nielsen. Fast stabbing of boxes in high dimensions. *Theor. Comput. Sci.*, 246(1-2): 53–72, 2000.
- [OY91] M. H. Overmars and C. Yap. New upper bounds in klee's measure problem. *SIAM J. Comput.*, 20(6): 1034–1045, 1991.
- [PT11] J. Pach and G. Tardos. Tight lower bounds for the size of epsilon-nets. *Proceedings of the twenty-seventh annual symposium on Computational geometry*, 458–463, 2011.
- [VC71] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2): 264–280, 1971. eprint: https://doi.org/10.1137/1116025.