Near-Linear Approximation Algorithms for Geometric Hitting Sets*

Pankaj K. Agarwal† Esther Ezra‡ Micha Sharir§

March 23, 2011

Abstract

Given a range space $(X, \mathcal{R})$, where $\mathcal{R} \subset 2^X$, the *hitting set* problem is to find a smallest-cardinality subset $H \subseteq X$ that intersects each set in $\mathcal{R}$. We present *near-linear-time* approximation algorithms for the hitting set problem in the following geometric settings: (i) $\mathcal{R}$ is a set of planar regions with small union complexity. (ii) $\mathcal{R}$ is a set of axis-parallel d-dimensional boxes in $\mathbb{R}^d$. In both cases X is either the entire $\mathbb{R}^d$, or a finite set of points in $\mathbb{R}^d$. The approximation factors yielded by the algorithm are small; they are either the same as, or within very small factors off the best factors known to be computable in polynomial time.

*Work on this paper has been supported by a joint grant, no. 2006/194, from the US-Israel Binational Science Foundation. Work by Pankaj Agarwal is also supported by NSF under grants CNS-05-40347, CCF-06 -35000, IIS-07-13498, and CCF-09-40671, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, by an NIH grant 1P50-GM-08183-01, and by a DOE grant OEG-P200A070505. Work by Micha Sharir has also been supported by NSF Grants CCF-05-14079 and CCF-08-30272, by Grants 155/05 and 338/09 from the Israel Science Fund, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University. A preliminary version of this paper has appeared in *Proc. 25^{th} Annu. ACM Sympos. Comput. Geom.* 2009, pp. 23–32. Part of this work was done while Esther Ezra was at Duke University.

†Department of Computer Science, Duke University, Durham, NC 27708-0129, USA; *pankaj@cs.duke.edu.*

‡Department of Computer Science, Courant Institute of Mathematical Sciences, New York, NY 10012, USA; *esther@courant.nyu.edu.*

§School of Computer Science, Tel Aviv University, Tel Aviv 69978 Israel and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA; *michas@post.tau.ac.il.*

1 Introduction

A *range space* $(X, \mathcal{R})$ is a pair consisting of an underlying universe X of objects and a family $\mathcal{R}$ of subsets of X (called *ranges*). A subset $H \subseteq X$ is called a *hitting set* of $(X, \mathcal{R})$ if it intersects every (nonempty) range in $\mathcal{R}$. In many geometric applications, X is the entire $\mathbb{R}^d$ and the ranges in $\mathcal{R}$ are simply-shaped regions (halfspaces, balls, simplices, etc.) that are subsets of $\mathbb{R}^d$. Several applications, however, require X to be a finite point set, in which case we regard each range $R \in \mathcal{R}$ as the intersection $R \cap X$; with a slight abuse of notation, we denote the range space $(X, \{R \cap X \mid R \in \mathcal{R}\})$ as $(X, \mathcal{R})$. We refer to this version of the problem as the *discrete model*, and to the version in which $X = \mathbb{R}^d$ as the *continuous model*. In both cases, the (*geometric*) *hitting-set* problem is to find a smallest set $H \subseteq X$ that intersects every range in $\mathcal{R}$; we call H an *optimal hitting set* for $(X, \mathcal{R})$. Let $\kappa := \kappa(X, \mathcal{R})$ denote the size of such an optimal hitting set. We note that, in general, the hitting-set problem, as just formulated, is meaningful only when $\mathcal{R}$ is a finite set of regions (taken from some infinite class).

The hitting-set problem is NP-Hard, under both discrete and continuous models, even for very simple geometric settings in $\mathbb{R}^2$, e.g., when $\mathcal{R}$ is a set of unit disks or squares [24, 26]. Therefore attention has mostly focused on developing polynomial-time approximation algorithms. In some applications, such as sensor networks, database systems, information retrieval, and computer vision, the sets X and $\mathcal{R}$ change dynamically and it is necessary to construct a hitting set repeatedly (or just update it dynamically); see, e.g., [5, 9]. It is desirable in such situations to obtain a trade-off between the size of the hitting set and the time spent in computing it—one may prefer to have a slightly larger hitting set if it can be computed much faster. Motivated by these applications, we study the problem of computing a hitting set in *near-linear time* and present algorithms of this kind for several special cases, in which the approximation factors that we obtain are either asymptotically the same, or are larger by only a very small, generally (sub-)logarithmic factor, than the best factors known to be computable in (less efficient) polynomial time. The running time of algorithms for the hitting-set problem is measured in terms of both $|X|$ and $|\mathcal{R}|$ in the discrete model, or in terms of $|\mathcal{R}|$ alone in the continuous model.

Previous results. The well-known greedy algorithm gives a polynomial-time $(\log n)$-approximation for a hitting set [43, Chapter 2], and the known lower-bound results suggest that this is the best approximation factor one can hope to achieve in polynomial time for general range spaces [23]. However, by exploiting the underlying geometry, one can obtain polynomial-time algorithms with better approximation factors for many geometric hitting-set problems. These algorithms employ and adapt a wide range of novel techniques, including variants of the greedy algorithm, dynamic programming, LP-relaxation, and ε-nets. It is beyond the scope of this paper to give a comprehensive review of the known results. We only mention a few results that are directly relevant to our results, and we refer the reader to [12, 13, 14, 21, 28] and the references therein for more details.

A polynomial-time $(1+\varepsilon)$-approximation algorithm, for any $\varepsilon > 0$, is known for the continuous model if $\mathcal{R}$ is a set of "fat" objects [14]; see also [28]. Nielsen [37] proposed an $O(n \log \kappa)$-time algorithm for computing a hitting set of size $O(\kappa \log^{d-1} \kappa)$ for the continuous model if $\mathcal{R}$ is a set of n boxes in $\mathbb{R}^d$. These techniques, however, do not extend to the discrete model. Recently, Mustafa and Ray have presented a local-search based PTAS for the discrete model in which $\mathcal{R}$ is a set of

pseudo-disks in $\mathbb{R}^2$ or halfspaces in $\mathbb{R}^3$ [36].

Given a parameter $0 < \varepsilon < 1$, an *ε-net* for a range space $(X, \mathcal{R})$ (with X finite) is a subset $N \subseteq X$ with the property that N intersects every range $R \in \mathcal{R}$ whose size is at least $\varepsilon|X|$. In other words, N is a hitting set for all the "heavy" ranges. By a seminal result of Haussler and Welzl [27], a range space of *finite VC-dimension* δ (see [27] for the definition) has an ε-net of size $O((\delta/\varepsilon)\log(\delta/\varepsilon))$. Improved bounds on the size of ε-nets are known in several special cases. For example, ε-nets of size $O(1/\varepsilon)$ exist when X is a point set and $\mathcal{R}$ is a set of halfspaces in two or three dimensions, pseudo-disks in the plane, or translates of a fixed convex polytope in 3-space; see [29, 34, 40]. Recently, Aronov *et al.* [11] have shown the existence of ε-nets of size $O((1/\varepsilon)\log\log(1/\varepsilon))$ for the case where $\mathcal{R}$ is a set of n axis-parallel rectangles or boxes in $\mathbb{R}^2$ or $\mathbb{R}^3$, respectively. A remarkable recent result of Pach and Tardos [39] shows that this bound is worst-case tight. Similar bounds have been obtained for other geometric range spaces, by Varadarajan [17, 42] and by King and Kirkpatrick [32].

Generalizing a technique by Clarkson [15], Brönnimann and Goodrich [13] presented a general method for computing a hitting set for a range space, under the discrete model, using ε-nets. More precisely, let $(X, \mathcal{R})$ be a range space for which an ε-net of size $O((1/\varepsilon)\varphi(1/\varepsilon))$, where $\varphi(\cdot)$ is a slowly monotonically-nondecreasing *sublinear* function, can be constructed in polynomial time. Brönnimann and Goodrich present an *iterative reweighted sampling* scheme to compute a hitting set $H \subseteq X$ for $\mathcal{R}$ of size $O(\kappa\varphi(\kappa))$ in polynomial time (as a matter of fact, their technique is applied under a wider context and exploits the existence of *weighted* ε-nets—see [13] and Section 5 for further details). Hence, a hitting set of size $O(\kappa\log\kappa)$ can be computed, under the discrete model, for any range space with finite VC-dimension. The size reduces to $O(\kappa\log\log\kappa)$ if $\mathcal{R}$ is a set of axis-parallel rectangles in $\mathbb{R}^2$ or boxes $\mathbb{R}^3$, or to $O(\kappa)$ in each of the cases mentioned above when there exists a (polynomial-time computable) linear-size ε-net. Several other instances with improved bounds are listed in [11]. A related technique due to Even *et al.* [21], which exploits LP-relaxation, results in similar approximation factors obtained in polynomial time. They solve an instance of linear programming that corresponds to the original hitting-set problem. Using the fractional solution of LP as weights of points in X, they construct an ε-net for the weighted range space and argue that such an ε-net is a hitting set for the original (unweighted) range space. The known results for the discrete model are summarized in Table 1.

Both the greedy and the Brönnimann–Goodrich (or the Even et al.) algorithms terminate in $O(\kappa\log|\mathcal{R}|)$ stages. A straightforward implementation of the greedy algorithm takes $\Omega(|X||\mathcal{R}|)$ time per stage, which can be improved to $O((|X|+|\mathcal{R}|)\mathrm{polylog}(|\mathcal{R}|))$ (expected) time per stage if the complexity of the union of the regions in $\mathcal{R}$ is near linear [12]. Similarly, a straightforward implementation of the Brönnimann–Goodrich algorithm takes $\Omega(|X|+|\mathcal{R}|)$ time (per stage). Thus the lower bounds on the running time of these approaches are of the form $\Omega((|X|+|\mathcal{R}|)\kappa\log n)$. It is not clear whether the running time of these algorithms can be improved to $O((|X|+|\mathcal{R}|)\mathrm{polylog}(|\mathcal{R}|))$ or at least to $O((|X|+|\mathcal{R}|+\kappa^{O(1)})\mathrm{polylog}(|\mathcal{R}|))$. Although near-linear algorithms are known for computing a hitting set under the continuous model in several special cases (e.g., an $O(1)$-approximation algorithm when the objects are fat [14]), no such algorithm is known under the discrete model even for very simple cases (e.g., the case of points and disks in the plane).

ranges	*approximation factor*
finite VC dimension	$\log \kappa$
axis parallel boxes in $\mathbb{R}^2, \mathbb{R}^3$	$\log \log \kappa$
fat triangles in $\mathbb{R}^2$	$\log \log \kappa$
translates of a 3d convex polytope	$O(1)$
pseudo-disks in $\mathbb{R}^2$ halfspaces in $\mathbb{R}^3$	$1+\varepsilon$

Table 1. Summary of best-known polynomial-time approximation algorithms for the discrete model. The running time is $(mn)^{O(1/\varepsilon^2)}$ for the last row and $(mn)^{O(1)}$ for the rest.

Our results. We present near-linear algorithms for computing a hitting set of approximately optimal size, under both the discrete and continuous models, for several special but useful cases. We propose two main approaches in this paper; the first one is a variant of the greedy algorithm, and the second one is based on the iterative reweighted sampling scheme. The first approach incorporates *shallow cuttings*, and, as shown by our analysis, yields small approximation factors (obtained in nearly-linear time) for families of planar regions of small union complexity. The second approach provides a sophisticated implementation of the iterative reweighted sampling scheme of Brönnimann and Goodrich [13] for the case of d-boxes in $\mathbb{R}^d$—see below.

We begin in Section 2 by describing an algorithm for computing a shallow cutting of a set of planar regions, augmented with some additional procedures that will be used by our hitting-set algorithms. Although algorithms for constructing shallow cuttings are known [4, 34], adapting them to support the additional procedures required in our scenario is nontrivial. We use a randomized incremental approach to construct shallow cuttings, based on a variant of the algorithm of Agarwal *et al.* [3] for constructing levels in arrangements. As a by-product, we obtain an efficient algorithm to approximate the largest "depth" in an arrangement of planar regions, which, to our knowledge, is faster than previous algorithms for this problem.

We first consider the case when $\mathcal{R}$ is a set of n closed, connected planar regions of constant description complexity[1] and X is a set of m points in $\mathbb{R}^2$. We assume that the union complexity of any subset of $\mathcal{R}$ of size r is $O(r\varphi(r))$, where $\varphi(r)$ (as defined above) is a slowly monotonically-nondecreasing *sublinear* function (ideally, $o(\log r)$ or at worst $O(\mathrm{polylog}(r))$). We describe in Section 3 a randomized algorithm that computes, in $O((m + n\varphi(\kappa)) \log^2 \kappa \log n)$ expected time, a hitting set $H \subseteq X$ for $\mathcal{R}$ of size $O(\kappa\varphi(\kappa) \log n)$. For the continuous model (i.e., when $X = \mathbb{R}^2$), the expected running time is $O(n\varphi(\kappa) \log^2 \kappa \log n)$, yielding a similar approximation factor as in the discrete case. Roughly speaking, our approach can be regarded as a variant of the greedy algorithm, in which we carefully choose $O(\kappa\varphi(\kappa))$ points at each stage, and ensure that the algorithm terminates within $O(\log n)$ stages. In contrast, the greedy algorithm terminates in $O(\kappa \log n)$ stages and chooses a single point at each stage. We obtain a faster algorithm by reducing the number of stages, but pay an extra $\varphi(\kappa)$ factor in the output size.

Our algorithm is applicable for a variety of classes of planar regions with small union complex-

[1] A set has *constant description complexity* if it is a semi-algebraic set defined by a constant number of polynomial equations and inequalities of constant maximum degree; see [8] for details.

ity; see [7] for a comprehensive survey. Here we just mention a few such cases for which we obtain near-linear hitting-set algorithms with small approximation factors:

(i) The approximation factor is $O(\log n)$ for pseudo-disks [31].

(ii) The approximation factors are $O(\log n)$ for fat wedges [38] and $O(\log n \log^* \kappa)$ for fat triangles [10, 22].

(iii) The approximation factor is $2^{O(\log^* \kappa)} \log n$ for locally γ-fat objects [10]; the constant of proportionality in the exponent depends on γ and on the description complexity of the objects.

(iv) The approximation factor is $O(\alpha(\kappa) \log n)$ for regions bounded by Jordan arcs having three intersections per pair, in the sense of [20].

We next consider, in Section 4, the case in which $\mathcal{R}$ is a set of n axis-parallel rectangles in $\mathbb{R}^2$, and X a set of m points in $\mathbb{R}^2$. The union complexity of $\mathcal{R}$ can be quadratic in the worst case, so we cannot apply the previous algorithm directly. We present an algorithm that computes a hitting set $H \subseteq X$ for $\mathcal{R}$ of size $O(\kappa \log n)$ in time $O((m+n)\log^2 n)$. Using an appropriate range-searching data structure, we can extend the algorithm to higher dimensions: Given a set X of m points in $\mathbb{R}^d$ and a set $\mathcal{R}$ of n d-dimensional boxes (referred to as *d-boxes*, or simply as *boxes*) in $\mathbb{R}^d$, a hitting set of size $O(\kappa \log n)$ can be computed in $O((m+n)\log^d n)$ time. The algorithm does not seem to extend to the continuous model without increasing the running time significantly.

Finally, we present in Section 5 a fast implementation of the iterative reweighted-sampling scheme of Brönnimann and Goodrich for the case of axis-parallel d-boxes in $\mathbb{R}^d$. Given a set X of m points and a set $\mathcal{R}$ of n axis-parallel d-boxes in $\mathbb{R}^d$, we can compute a hitting set $H \subseteq X$ for $\mathcal{R}$ of $O(\kappa \log \kappa)$ size in $O((m+n+\kappa^{d+1})\mathrm{polylog}(mn))$ randomized expected time. The main ingredient of our algorithm is a data structure that, after $O((m+n)\mathrm{polylog}(mn))$ preprocessing time, can implement each stage of the Brönnimann–Goodrich algorithm in $O(\kappa^d \mathrm{polylog}(mn))$ time. We improve the approximation factor to $O(\log\log\kappa)$, for $d = 2, 3$, using the recent bound of Aronov *et al.* [11], mentioned above, on the size of ε-nets in this case, while keeping the expected running time $O((m+n+\kappa^{d+1})\mathrm{polylog}(mn))$.

2 Arrangements and Shallow Cuttings

Arrangements and levels. Let $\mathcal{R}$ be a set of n (closed) connected planar regions of constant description complexity. We refer to the boundaries of regions in $\mathcal{R}$ as *boundary curves*. Without loss of generality, we assume that the regions in $\mathcal{R}$ are in general position, that is, no point is incident to more than two boundary curves, and when two curves meet at a point, they cross each other there. Let $\mathcal{A}(\mathcal{R})$ denote the arrangement of $\mathcal{R}$ [8], and let $\mathcal{U}(\mathcal{R}) = \bigcup \mathcal{R}$ denote the union of $\mathcal{R}$. The combinatorial complexity of $\mathcal{A}(\mathcal{R})$ is the number of its vertices, edges, and two-dimensional faces, and the combinatorial complexity of $\mathcal{U}(\mathcal{R})$ is the number of vertices and edges of $\mathcal{A}(\mathcal{R})$ that appear along $\partial\mathcal{U}(\mathcal{R})$. In the sequel we consider families $\mathcal{R}$ as above that satisfy the following property: the complexity of the union of any $r \leq n$ regions of $\mathcal{R}$ is near linear in r. Specifically, let $f(r)$ denote the maximum complexity of the union of a subset of r regions in $\mathcal{R}$, for $r \leq n$. Then we

address the case where $f(r) \leq r\varphi(r)$, where $\varphi(\cdot)$ is a slowly growing sublinear function (e.g., $\varphi(r) = O(\log r)$). We then call regions of the above kind *well behaved*.

The *depth* of a point $p \in \mathbb{R}^2$ in $\mathcal{A}(\mathcal{R})$, denoted by $\Delta(p, \mathcal{R})$, is the number of regions $R \in \mathcal{R}$ that contain p in their interior. For $0 \leq l \leq n$, the *l-level* of $\mathcal{A}(\mathcal{R})$, denoted by $\mathcal{A}_l(\mathcal{R})$, is the set of all points whose depth is l; for example, the 0-level is the closure of the complement of $\mathcal{U}(\mathcal{R})$. Let $\mathcal{A}_{\leq l}(\mathcal{R})$ denote the set of points whose level is at most l, i.e., $\mathcal{A}_{\leq l}(\mathcal{R}) = \bigcup_{j=0}^{l} \mathcal{A}_j(\mathcal{R})$. The complexity of $\mathcal{A}_{\leq l}(\mathcal{R})$ is the number of vertices, edges, and faces of $\mathcal{A}(\mathcal{R})$ that lie in $\mathcal{A}_{\leq l}(\mathcal{R})$. It suffices to consider only vertices because the number of edges and faces is proportional to $|\mathcal{R}|$ plus the number of vertices. A well-known result by Clarkson and Shor [16] implies that the complexity of $\mathcal{A}_{\leq l}(\mathcal{R})$, for $l \geq 1$, is $O(l^2 f(n/l)) = O(nl\varphi(n/l))$ (it is at most $f(n) \leq n\varphi(n)$ for $l = 0$). In particular, if $l^* = \max_{p \in \mathbb{R}^2} \Delta(p, \mathcal{R})$, then the complexity of $\mathcal{A}(\mathcal{R})$ is $O(nl^*\varphi(n/l^*))$. Thus if the union complexity is small and the arrangement is "shallow" (i.e., its maximum depth is small), the overall arrangement complexity is also relatively small.

In what follows, when we consider shallow levels of an arrangement, we will always assume, for the sake of simplicity, that $l \geq 1$, to avoid a more cumbersome notation; for example, without such an assumption we would have to write the bound on the complexity of $\mathcal{A}_{\leq l}(\mathcal{R})$ as $O((l+1)^2 f(n/(l+1))) = O(n(l+1)\varphi(n/(l+1)))$, to accommodate also the case $l = 0$.

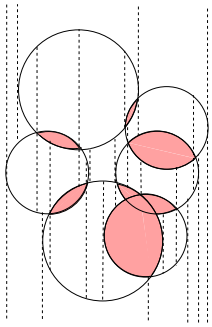

Figure 1. Vertical decomposition of $\mathcal{A}_{\leq 1}$; the level of the shaded regions is ≥ 2.

Vertical decomposition. The *vertical decomposition* of a face C of $\mathcal{A}(\mathcal{R})$, denoted by $C^{\shortparallel}$, is obtained by erecting, within C, a vertical segment through each vertex and locally x-extremal point on ∂C in both $(+y)$- and $(-y)$-directions until it meets ∂C again (or else extends to $\pm\infty$). $C^{\shortparallel}$ partitions C into *pseudo-trapezoidal* cells, each bounded by at most two arcs of ∂C and at most two vertical segments; some of the cells may be degenerate — they may be unbounded, or they may have fewer than four edges even when bounded. The number of pseudo-trapezoids in $C^{\shortparallel}$ is proportional to the number of vertices plus the number of x-extremal points in C. The vertical decompositions of $\mathcal{A}(\mathcal{R})$ and of $\mathcal{A}_{\leq l}(\mathcal{R})$, denoted by $\mathcal{A}^{\shortparallel}(\mathcal{R})$ and $\mathcal{A}^{\shortparallel}_{\leq l}(\mathcal{R})$, respectively, are obtained by constructing the vertical decomposition of each of the cells of the respective portion of the plane; see Figure 1. By the result of Clarkson and Shor [16], the number of pseudo-trapezoids in $\mathcal{A}^{\shortparallel}_{\leq l}(\mathcal{R})$ is $O\left(nl\varphi(n/l)\right)$; A pseudo-trapezoid τ in $\mathcal{A}^{\shortparallel}(\mathcal{R})$ is *defined* by a set $\mathcal{D}(\tau)$ of at most four regions, in the sense that τ

is a cell in $\mathcal{A}^{\shortparallel}(\mathcal{D}(\tau))$, and $\mathcal{D}(\tau)$ is minimal with respect to this property. We call a region $\tau \subseteq \mathbb{R}^2$ a *primitive pseudo-trapezoid* if there exists a subset $\mathcal{D} = \mathcal{D}(\tau)$ of $\mathcal{R}$ of the above kind. Note that the interior of a primitive pseudo-trapezoid may intersect the boundary of some of the regions in $\mathcal{R} \setminus \mathcal{D}$.

Shallow cuttings. Let $l \geq 0$ and $1 \leq r \leq n$ be two parameters. A collection Ξ of (possibly unbounded) pairwise-disjoint primitive pseudo-trapezoids is called a $(1/r)$-*cutting* of $\mathcal{A}_{\leq l}(\mathcal{R})$, also known as a *shallow* $(1/r)$-*cutting*, if the union of the cells in Ξ covers $\mathcal{A}_{\leq l}(\mathcal{R})$, and the interior of each cell $\tau \in \Xi$ is crossed by at most n/r boundary curves of regions of $\mathcal{R}$. The *size* of Ξ is the number of its cells. The set $\mathcal{R}_\tau$ of regions of $\mathcal{R}$ whose boundary curves cross a cell $\tau \in \Xi$ is called the *conflict list* of τ. The notion of shallow cutting was originally introduced by Matoušek [34] for arrangements of halfspaces in $\mathbb{R}^d$, and this notion was subsequently extended to other classes of regions (see, e.g., [4]). It is shown in [34] (see also [4]) that there exists a $(1/r)$-cutting of $\mathcal{A}_{\leq l}(\mathcal{R})$ of size $O(qr\varphi(r/q))$, where $q = l(r/n) + 1$. (Here the bound also applies to $l = 0$.)

The proof of the existence of such a cutting, for both cases considered in [4, 34] (and for the case considered here), is constructive and is based on a two-level random sampling. Specifically, we choose a random subset $\mathcal{G} \subseteq \mathcal{R}$ of r regions, compute $\mathcal{A}^{\shortparallel}(\mathcal{G})$, and discard those cells of $\mathcal{A}^{\shortparallel}(\mathcal{G})$ that do not intersect $\mathcal{A}_{\leq l}(\mathcal{R})$. For each remaining cell τ, consider its conflict list $\mathcal{R}_\tau$. If $|\mathcal{R}_\tau| \leq n/r$, we add τ to the final cutting Ξ. Otherwise, if $(t-1)n/r \leq |\mathcal{R}_\tau| \leq tn/r$, for some integer $t \geq 2$, we choose a random sample $\mathcal{G}_\tau \subseteq \mathcal{R}_\tau$ of $O(t \log t)$ regions, compute $\mathcal{A}^{\shortparallel}(\mathcal{G}_\tau)$, clip each cell of $\mathcal{A}^{\shortparallel}(\mathcal{G}_\tau)$ within τ (possibly partitioning further each clipped cell into $O(1)$ proper pseudo-trapezoids), and add the clipped cell to Ξ if it intersects $\mathcal{A}_{\leq l}(\mathcal{R})$. As follows, e.g., from the argument given in [34], Ξ is a $(1/r)$-cutting of $\mathcal{A}_{\leq l}(\mathcal{R})$ with high probability, and its expected size is $O(qr\varphi(r/q))$.

Our shallow-cutting algorithm is implemented somewhat differently from the classical construction of [34] (or of [4]), using a variant of the randomized incremental procedure of Agarwal *et al.* [3] for constructing $\mathcal{A}_{\leq l}(\mathcal{R})$. Still, from a high-level point of view, both techniques implement the same two-stage sampling scheme described in [4, 34]. The reason for deviating from the standard approach is that our implementation allows us to keep track of additional information that we need to maintain—see below.

Let $1 \leq r, l \leq n$ be the given parameters. Let $\langle R_1, \ldots, R_n \rangle$ be a random permutation of $\mathcal{R}$, and let $\mathcal{R}_i = \langle R_1, \ldots, R_i \rangle$. In the first stage, we insert the regions in $\mathcal{R}_r$ in order, one at a time. The algorithm maintains, after each insertion step i (at which R_i is inserted) the following structures.

(i) A collection Ξ_i of certain cells of $\mathcal{A}^{\shortparallel}(\mathcal{R}_i)$ whose union covers $\mathcal{A}_{\leq l}(\mathcal{R})$ (not all the cells necessarily intersect $\mathcal{A}_{\leq l}(\mathcal{R})$).

(ii) The *conflict lists*: For each cell $\tau \in \Xi_i$, we maintain the set $\mathcal{R}_\tau \subseteq \mathcal{R} \setminus \mathcal{R}_i$ of those regions whose boundary curves cross (the interior of) τ; we also store the inverse data, by associating with each region $R \in \mathcal{R} \setminus \mathcal{R}_i$, the list of all cells τ with $R \in \mathcal{R}_\tau$. Set $n_\tau := |\mathcal{R}_\tau|$.

(iii) For each cell $\tau \in \Xi_i$, we store the level l_τ, with respect to $\mathcal{A}(\mathcal{R})$, of some arbitrarily chosen interior point $p \in \tau$.

The cells $\tau \in \Xi_i$ satisfy the invariant $l_\tau \leq l + n_\tau$; we refer to cells satisfying this condition as *active*. Suppose that we have inserted all the regions in $\mathcal{R}_{i-1}$ and that we have computed the $(i-1)$-st version of the above structures. We insert the region R_i and update the structures as follows.

(i) We find all the cells of Ξ_{i-1} whose interiors are intersected by ∂R_i, using the respective inverse conflict list. We split each such cell τ into the connected components of $\tau \setminus \partial R_i$, re-decompose each component into primitive pseudo-trapezoids and merge some of the components into a single pseudo-trapezoid as necessary, construct the conflict lists of the new cells (from those of the containing cells τ, just being destroyed), and compute the level information for each new cell.

(ii) We then test each of the newly created cells τ to see whether it is active. If τ is not active, we discard it along with its conflict list.

Note that the cells of Ξ_{i-1} that are fully contained in R_i are not touched at the ith step. We refer the reader to the original procedure, described in [3], for further details.

The time spent at each step is proportional to the number of newly created and destroyed cells τ in that step, plus the overall size of their conflict lists. A cell destroyed at step i must have been created at an earlier step, so it suffices to consider only cells created at each step i. Following the analysis in [3], the expected time spent in step i is

$$O\left(\frac{1}{i}\cdot\frac{n}{i}\cdot\frac{i^2}{n^2}n\varphi(n/l)\left(l+\frac{n}{i}\log i\right)\right)=O\left(\varphi(n/l)\left(l+\frac{n}{i}\log i\right)\right). \tag{1}$$

The above bound can be justified by the following informal argument. At step i, the maximum level of any point in an active cell τ of $\mathcal{A}^{[l]}(\mathcal{R}_i)$ is at most $l+2n_\tau$, and the analysis of Clarkson and Shor [16] implies that, with high probability, $n_\tau = O((n/i)\log i)$ for all τ, so all active cells are contained in level at most $h = l + O((n/i)\log i)$ of $\mathcal{A}(\mathcal{R})$. The complexity of $\mathcal{A}_{\le h}(\mathcal{R})$ is $O(nh\varphi(n/l))$. The probability of a vertex of $\mathcal{A}_{\le h}(\mathcal{R})$ to appear in $\mathcal{A}(\mathcal{R}_i)$ is roughly $O(i^2/n^2)$, so the expected number of active cells in Ξ_i is $O\left((i^2/n^2)\cdot nh\varphi(n/l)\right)$, and the expected size of the conflict list of each of these cells is $O(n/i)$. The probability that a cell of Ξ_i was created at the i-th step itself (conditioned on appearing in Ξ_i) is roughly $O(1/i)$. Putting this together, one can deduce that the expected overall size of the conflict lists of the newly created cells at step i plus the expected number of these cells is bounded by (1). The same argument also implies that the number of active cells τ for which $n_\tau \le n/i$ is

$$O\left(\frac{i^2}{n}\left(l+\frac{n}{i}\right)\varphi(n/l)\right). \tag{2}$$

Summing (1) over all r steps, the overall expected running time of the first stage of the algorithm is

$$O((n\log^2 r + rl)\varphi(n/l)). \tag{3}$$

We now take the resulting partition $\Xi' := \Xi_r$ and refine each of its cells τ into a collection Ψ_τ of subcells, as follows. If $n_\tau \le n/r$ (where $n_\tau = |\mathcal{R}_\tau|$), Ψ_τ consists only of τ itself. Otherwise, let $t \ge 1$ be the integer satisfying $tn/r < n_\tau \le (t+1)n/r$; we refer to t as the *excess* of τ. We apply a second sampling step to τ by constructing a $(1/t)$-cutting Ψ_τ for $\mathcal{R}_\tau$ that covers $\mathcal{A}_{\le l}(\mathcal{R})$ within τ. More precisely, we choose a random permutation of $\mathcal{R}_\tau$ and add the first $c't\ln t$ regions in this permutation, for some appropriate constant $c' > 0$, using the algorithm of the first stage, with two major differences: (i) each cell is clipped to within τ, and (ii) for simplicity, the

cells of Ψ_τ cover the entire arrangement $\mathcal{A}(\mathcal{R}_\tau)$ (clipped to within τ); as our analysis shows, this does not increase the bound on the overall expected running time, incurred at the first sampling step (see below for details). By construction, the cells of Ψ_τ cover $\mathcal{A}_{\leq l}(\mathcal{R})$ within τ. Instead of using the complex argument of Agarwal *et al.* [3], we can use the simpler argument by Clarkson and Shor [16] to prove that Ψ_τ is a $(1/t)$-cutting of $\mathcal{R}_\tau$ with probability at least $1/2$, that the expected number of cells in Ψ_τ is $O(t^2 \log^2 t)$, and that the expected time spent in constructing Ψ_τ is $O(n_\tau t \log t) = O((nt^2/r) \log t)$. If Ψ_τ is not a $(1/t)$-cutting, we reconstruct Ψ_τ from scratch, and keep doing so until we obtain a valid $(1/t)$-cutting. We set the final cutting Ξ to be $\bigcup_{\tau \in \Xi'} \Psi_\tau$.

For $t \geq 0$, let $\eta(r, t, l)$ denote an upper bound on the expected number of cells in Ξ' with excess at least t. Substituting $i = r$ in (2), we note that

$$\eta(r, 0, l) = O((r + lr^2/n)\varphi(n/l)) = O(qr\varphi(r/q)), \tag{4}$$

where $q = l(r/n) + 1$. Using an argument similar to those in [4, 6, 34], we can argue that for $t \geq 1$,

$$\eta(r, t, l) = O\big(2^{-t}\eta(r/t, 0, l)\big).$$

Summing the expected running time of the second stage over all cells τ, we thus obtain the bound

$$\begin{aligned}
\sum_{t \geq 1} \eta(r, t, l) \cdot O\left(\frac{n}{r} t^2 \log t\right) &= \sum_{t \geq 1} 2^{-t} \eta(r/t, 0, l) \cdot O\left(\frac{n}{r} t^2 \log t\right) \\
&= O\left(\frac{n}{r} \cdot \eta(r, 0, l)\right) \\
&= O((n + lr)\varphi(n/l)).
\end{aligned}$$

Adding the expected time of the first stage, the expected running time of the overall algorithm is

$$O((n \log^2 r + lr)\varphi(n/l)). \tag{5}$$

Similarly, the expected size of Ξ is

$$\begin{aligned}
\sum_{\tau \in \Xi'} \mathsf{E}\{|\Psi_\tau|\} &= \eta(r, 0, l) + \sum_{t \geq 1} \eta(r, t, l) O(t^2 \log^2 t) \\
&= O(qr\varphi(r/q)) = O((r + lr^2/n)\varphi(n/l)).
\end{aligned}$$

If $|\Xi| > A(r + lr^2/n)\varphi(n/l)$, for an appropriate constant $A \geq 1$, we repeat the above construction from scratch, and keep doing so until a cutting of the desired size is obtained. With an appropriate choice of the constant A, the probability of having to re-run the algorithm will be at most $1/2$. Hence we obtain the following:

Lemma 2.1. *Let $\mathcal{R}$ be a set of n well-behaved planar regions, and let $l \geq 0, r \geq 1$ be two given parameters. A $(1/r)$-cutting for $\mathcal{A}_{\leq l}(\mathcal{R})$ of size $O((r + lr^2/n)\varphi(n/l))$, along with the conflict list of each cell, can be computed in expected time $O((n \log^2 r + lr)\varphi(n/l))$ (which is also an upper bound on the expected overall size of the conflict lists).*

Remarks: (i) As indicated by the analysis, the bound on the running time is dominated by the bound for the first stage of the algorithm.

(ii) The algorithm in [3] inserts *all* the elements $R_1, \ldots, R_n$ (and thus $r = n$ in this case) in order to obtain the complete structure $\mathcal{A}_{\leq l}(\mathcal{R})$, whereas ours stops after r steps to obtain a shallow cutting instead.

Next, we describe two extensions of this construction, each of which will be needed for our hitting-set algorithm.

A point location mechanism. We can construct a directed acyclic graph, called a *history dag* and denoted by $\mathbb{H}$, whose nodes are the pseudo-trapezoids constructed by the algorithm (in the first or the second stage). There is an edge (τ', τ) in $\mathbb{H}$ if τ is created at the step where τ' is being destroyed, and $\tau' \cap \tau \neq \emptyset$. It is known that the expected length of any path in $\mathbb{H}$ is $O(\log r)$ [18]. Moreover, by our iterative construction (at both the first and the second stages), the out-degree of each node is $O(1)$ since the regions have constant description complexity (see, e.g. [41, Chapter 6]).[2] Let p be a point in $\mathbb{R}^2$. By tracing an appropriate path through $\mathbb{H}$, we can compute the cell of Ξ containing p in $O(\log r)$ time (recall that the out-degree is only $O(1)$), if such a cell exists. We thus have:

Lemma 2.2. *Let $\mathcal{R}$ be a set of n well-behaved planar regions, $l \geq 0$ and $r > 1$ two parameters, and Ξ a $(1/r)$-cutting of $\mathcal{A}_{\leq l}(\mathcal{R})$. Then one can construct in $O(n \log^2 r + lr)\varphi(n/l))$ expected time a point-location data structure for Ξ with expected query time $O(\log r)$.*

Hence, for a set X of m points in $\mathbb{R}^2$, we can locate, using Lemma 2.2, all the points of X in Ξ in overall expected time $O(m \log r)$. Let $\Xi^* \subseteq \Xi$ be the set of *nonempty* cells, namely, those that contain at least one point of X. Again, using the history dag $\mathbb{H}$, we can also compute the regions of $\mathcal{R}$ that fully contain at least one cell of Ξ^*. We thus obtain the following:

Corollary 2.3. *Let $\mathcal{R}$ be a set of n well-behaved planar regions, X a set of m points in $\mathbb{R}^2$, and $l \geq 0$ and $r > 1$ two parameters. The following can be computed in $O(m \log r + (n \log^2 r + lr)\varphi(n/l))$ expected time: (i) a $(1/r)$-cutting Ξ of $\mathcal{A}_{\leq l}(\mathcal{R})$, (ii) the sets $X \cap \tau$ for each $\tau \in \Xi$, and (iii) the subset of regions of $\mathcal{R}$ that contain at least one nonempty cell of Ξ.*

Approximating the maximal depth. Put $\delta^* = \max_{p \in X} \Delta(p, \mathcal{R})$. We next show how to compute a value $\delta \in [\delta^*/2, \delta^*]$ by performing an exponential search on δ and using the shallow-cutting algorithm and Corollary 2.3 at each step. Initially, we set $\delta = n/2$. At each step of the search, we set $l = \delta$, $r = n/l$, construct a $(1/r)$-cutting Ξ of $\mathcal{A}_{\leq l}(\mathcal{R})$, and use the history dag to track the depth of each point in X. More specifically, for each nonempty cell $\tau \in \Xi^*$, we retrieve N_τ, the number of regions that contain τ (that is, its depth with respect to the regions in $\mathcal{R} \setminus \mathcal{R}_\tau$). Let $\tau^0 = \arg\max_{\tau \in \Xi} N_\tau$. If $N_{\tau^0} \geq n_{\tau^0}$, we stop and return N_{τ^0}. Otherwise, we set $\delta = \delta/2$ and repeat the above step. Let τ be a cell of Ξ that contains a point $p \in X$ of maximum depth δ^*. Then

$$N_\tau \leq \delta^* \leq N_\tau + n_\tau \leq N_\tau + n/r,$$

so $\delta^* - n/r \leq N_\tau \leq \delta^*$, which implies that the algorithm will stop as soon as n/r, that is, δ, becomes smaller than $\delta^*/2$, or at some earlier (larger) value of δ. In the former case we have

$$\delta^*/2 \leq \delta^* - \frac{n}{r} \leq N_{\tau^0} \leq \delta^*.$$

[2] In contrast, the merging of sub-trapezoids may cause the in-degree of trapezoids to be arbitrarily large.

In the latter case we have $N_{\tau^0} \geq \delta \geq \delta^*/2$. This establishes the correctness of the algorithm.

A closer inspection of the algorithm presented above shows that Ξ need not be reconstructed from scratch at each step. Indeed, note that the value of r at the i-th step is 2^i. Since the algorithm for the first sampling stage is incremental, we can compute Ξ_{2^i} from $\Xi_{2^{i-1}}$ by inserting $R_{2^{i-1}+1}, \ldots, R_{2^i}$ as described above, and removing the cells that are no longer active (recall that the value of l is halved at each step). A careful analysis shows that the total expected time spent in the first stage, over the entire exponential search, is within a constant factor of the time spent in constructing Ξ_{r^*}, where r^* is the final value of r when the algorithm terminates. By (3) and Corollary 2.3, the total expected time spent in the first stage is thus

$$O(m \log (n/\delta^*) + n\varphi(n/\delta^*) \log^2(n/\delta^*)).$$

The second sampling stage is applied from scratch at each step. The running time of each application of that stage (including the cost of locating the points of P in the cutting) is only

$$O(m \log r^* + (n + lr^*)\varphi(n/l)) = O(m \log (n/\delta^*) + n\varphi(n/\delta^*)),$$

and the number of steps in the exponential search is $O(\log (n/\delta^*))$. Hence, the overall cost of both stages is $O((m + n\varphi(n/\delta^*)) \log^2(n/\delta^*))$. Putting all the pieces together, we conclude the following:

Corollary 2.4. *Let $\mathcal{R}$ be a set of n well-behaved planar regions, let X be a set of m points in $\mathbb{R}^2$, and let $\delta^* = \max_{p \in X} \Delta(p, \mathcal{R})$. We can compute, in expected time $O((m + n\varphi(n/\delta^*)) \log^2(n/\delta^*))$, a value $\delta \in [\delta^*/2, \delta^*]$.*

3 Approximate Hitting Sets for Well-Behaved Planar Regions

This section describes our hitting-set algorithm for range spaces $(X, \mathcal{R})$, where $\mathcal{R} = \{R_1, \ldots, R_n\}$ is a set of n well behaved regions in $\mathbb{R}^2$, and where X is either $\mathbb{R}^2$ (the continuous model) or a finite set of m points in $\mathbb{R}^2$ (the discrete model). We first consider the discrete model and then extend the machinery to tackle the continuous model.

The discrete model. The algorithm works in stages. At the beginning of the ith stage, we have subsets $\mathcal{R}_i \subseteq \mathcal{R}$, $X_i \subseteq X$, and a hitting set $H_{i-1} = X \setminus X_i$ for $(X, \mathcal{R} \setminus \mathcal{R}_i)$. Set $n_i = |\mathcal{R}_i|$ and $m_i = |X_i|$. Initially, $\mathcal{R}_1 = \mathcal{R}$, $X_1 = X$, and $H_0 = \emptyset$. The algorithm terminates when $\mathcal{R}_i = \emptyset$ or $X_i \cap R = \emptyset$ for every $R \in \mathcal{R}_i$ (i.e., $X_i \subset \mathbb{R}^2 \setminus \bigcup \mathcal{R}_i$). If $\mathcal{R}_i \neq \emptyset$ at the time of termination, then X does not hit the entire $\mathcal{R}$. Nevertheless, the algorithm returns a valid hitting set—see below. The ith stage consists of the following steps:

Set $l_i = \max_{p \in X_i} \Delta(p, \mathcal{R}_i)$. Using Corollary 2.4, we can compute, in expected time

$$O((m_i + n_i\varphi(n_i/l_i)) \log^2(n_i/l_i)),$$

an integer $h_i \in [l_i/2, l_i]$. If $h_i = 0$, then $l_i = 0$, implying that $X_i \cap R = \emptyset$ for every $R \in \mathcal{R}_i$, so we terminate the algorithm and return H_{i-1}. Otherwise, we proceed as follows. Put $r_i :=$

$\min\{n_i, cn_i/h_i\}$, for some constant $c > 4$. Using Lemma 2.1, we construct a $(1/r_i)$-cutting Ξ_i of $\mathcal{A}_{\leq 2h_i}(\mathcal{R}_i)$ of size $O(r_i\varphi(r_i))$. If $h_i \leq c$ (i.e., $r_i = n_i$), then Ξ_i is the vertical decomposition of the appropriate portion of $\mathcal{A}(\mathcal{R}_i)$; for simplicity, we take Ξ_i to be the entire vertical decomposition $\mathcal{A}^{\shortparallel}(\mathcal{R}_i)$. In either case, since each point of X_i has depth at most $l_i \leq 2h_i$ and Ξ_i covers $\mathcal{A}_{\leq 2h_i}(\mathcal{R}_i)$, each of these points lies in some cell of Ξ_i. For each cell $\tau \in \Xi_i$, put $X_i^{(\tau)} = X_i \cap \tau$, and let $\Xi_i^* \subseteq \Xi_i$ denote the subset of cells τ for which $X_i^{(\tau)} \neq \emptyset$. For each $\tau \in \Xi_i^*$, we choose an arbitrary point p_τ of $X_i^{(\tau)}$. Let $\bar{H}_i$ denote the set of chosen points; we have (see (3))

$$|\bar{H}_i| = O(r_i\varphi(r_i)) = O((n_i/l_i)\varphi(n_i/l_i)).$$

By Corollary 2.3, $\bar{H}_i$ can be computed in

$$O(m_i \log(n_i/l_i) + n_i\varphi(n_i/l_i)\log^2(n_i/l_i))$$

expected time. Set $H_i = H_{i-1} \cup \bar{H}_i$ and $X_{i+1} = X_i \setminus \bar{H}_i$.

Using Corollary 2.3, we also compute the set $\mathcal{G}_i \subseteq \mathcal{R}_i$ of regions that fully contain at least one cell of Ξ_i^*. Clearly, each region in $\mathcal{G}_i$ is hit by at least one point of $\bar{H}_i$. These points potentially hit additional regions of $\mathcal{R}_i$, which we find by scanning the conflict list $\mathcal{R}_\tau$ (or, rather, its restriction $\mathcal{R}_{i,\tau}$ to $\mathcal{R}_i$), for each cell $\tau \in \Xi_i^*$ (recall that each such list consists of those regions whose boundaries cross τ), and by selecting those regions $R \in \mathcal{R}_{i,\tau}$ for which $p_\tau \in R$. We add all these regions to $\mathcal{G}_i$, and set $\mathcal{R}_{i+1} = \mathcal{R}_i \setminus \mathcal{G}_i$, observing that no region of $\mathcal{R}_{i+1}$ contains a point of $\bar{H}_i$ (or of H_i for that matter). Putting everything together, the expected running time of the ith stage is

$$O((m_i + n_i\varphi(n_i/l_i))\log^2(n_i/l_i)). \tag{6}$$

We proceed in this manner, until the algorithm terminates, which happens either when $\mathcal{R}_{i+1} = \emptyset$ or when $X_{i+1} \subseteq \mathbb{R}^2 \setminus \bigcup \mathcal{R}_{i+1}$. Upon termination, we output the final set H_i. Suppose the algorithm terminates after completing k stages. If $\mathcal{R}_{k+1} = \emptyset$, then obviously H_k is a hitting set of $(X, \mathcal{R})$. If $\mathcal{R}_{k+1} \neq \emptyset$, then $X_{k+1} \cap R = \emptyset$ for each $R \in \mathcal{R}_{k+1}$. Since H_k does not intersect any region in $\mathcal{R}_{k+1}$, we conclude that $X \cap R = \emptyset$ for each $R \in \mathcal{R}_{k+1}$. Hence, H_k is a (valid) hitting set of the range space $(X, \{R \cap X \mid R \in \mathcal{R}\})$. Alternatively, we can report in this case that X does not hit the entire $\mathcal{R}$. The next two lemmas bound the value of k and the size of $H = H_k$.

Lemma 3.1. *The algorithm stops within $\lceil \log_4 n \rceil$ stages.*

Proof. We claim that for any $j > 1$, $l_j \leq l_{j-1}/4$, which will prove the lemma, since $l_1 \leq n$. Indeed, let $p \in X_j$ be a point of the maximum depth l_j with respect to $\mathcal{R}_j$. By construction, p lies in a cell $\tau \in \Xi_{j-1}$, which must contain another point $q \neq p$ that was chosen for $\bar{H}_{j-1}$ (one such point was chosen, and it could not have been p, for then p would have belonged to $\bar{H}_{j-1}$ and not included in X_j). At the $(j-1)$st stage, we eliminate all the regions of $\mathcal{R}_{j-1}$ that contain q. Hence, if p lies in a region $R \in \mathcal{R}_j$, then $q \notin R$. Since both p, q lie in τ, the boundary of R has to intersect τ and thus $R \in \mathcal{R}_{j-1,\tau}$. Since Ξ_{j-1} is a $(1/r_{j-1})$-cutting, there are at most $n_{j-1}/r_{j-1} \leq h_{j-1}/c \leq l_{j-1}/4$ such regions, therefore $l_j = \Delta(p, \mathcal{R}_j) \leq l_{j-1}/4$, as claimed. □

Lemma 3.2. $|H| = O(\kappa\varphi(\kappa)\log n)$.

Proof. Without loss of generality, we assume that X intersects every region in $\mathcal{R}$. (As discussed above, if this does not hold then no subset of X can hit $\mathcal{R}$, and we need to consider only those regions that are hit by X.) Let κ_j be the size of the smallest subset of X_j that intersects every region of $\mathcal{R}_j$. Observe that $\kappa_j \geq n_j/l_j \geq n_j/(2h_j) \geq r_j/(2c)$, for each point of X_j can lie in at most $l_j \leq 2h_j$ elements of $\mathcal{R}_j$. Since $|\bar{H}_j| \leq |\Xi_j| = O(r_j\varphi(r_j))$ and $\varphi(\cdot)$ is a slowly monotonically-nondecreasing sublinear function, we have $|\bar{H}_j| = O(\kappa_j\varphi(\kappa_j))$. Next, we claim that $\kappa_j \leq \kappa$. This follows from the observation that if H^* is an optimal hitting set for $\mathcal{R}$ (of size κ), then $H^* \setminus H_{j-1}$ is a hitting set for $\mathcal{R}_j$, for the simple reason that, by construction, no point in H_{j-1} intersects any region of $\mathcal{R}_j$. Hence, $|\bar{H}_j| = O(\kappa\varphi(\kappa))$, and Lemma 3.1 implies that $|H| = O(\kappa\varphi(\kappa)\log n)$. □

Putting everything together, we obtain the following result, in which we use (6) to bound the cost of a single step, and recall that $n_i/l_i \leq \kappa_i \leq \kappa$ for each i.

Theorem 3.3. *Let $\mathcal{R}$ be a set of n regions of constant description complexity in $\mathbb{R}^2$ such that the union complexity of any r of them is $O(r\varphi(r))$, with φ as above, and let X be a set of m points in $\mathbb{R}^2$. A hitting set for $(X, \mathcal{R})$ of size $O(\kappa\varphi(\kappa)\log n)$, where $\kappa = \kappa(X, \mathcal{R})$, can be computed in randomized expected time $O((m + n\varphi(\kappa))\log^2 \kappa \log n)$.*

The continuous model. In the continuous model, the points in the hitting set can be chosen anywhere in the plane, i.e., $X = \mathbb{R}^2$. The following simplified version of the previous algorithm and its analysis apply in this case.

The ith stage now proceeds as follows. We have a set $\mathcal{R}_i$ of regions, and a hitting set H_{i-1} for $\mathcal{R}\setminus\mathcal{R}_i$. We set $l_i = \min_{p\in\mathbb{R}^2} \Delta(p, \mathcal{R}_i)$, compute a value $h_i \in [l_i/2, l_i]$, and set $r_i := \max\{n_i, cn_i/h_i\}$, with the same choice of c as above. We then construct a $(1/r_i)$-cutting Ξ_i of the entire $\mathcal{A}(\mathcal{R}_i)$, which is clearly equal to $\mathcal{A}_{\leq 2h_i}(\mathcal{R}_i)$. The size of Ξ_i is, as above, $O(r_i\varphi(r_i))$. We choose a point in each cell of Ξ_i, and let $\bar{H}_i$ be the set of chosen points. We remove the regions that contain one of the points in $\bar{H}_i$, as above, put $H_i = H_{i-1} \cup \bar{H}_i$, and set $\mathcal{R}_{i+1}$ to be the set of regions of $\mathcal{R}_i$ that have not yet been hit. Following the same analysis as above, we can argue that $|\bar{H}_i| = O(\kappa\varphi(\kappa))$ for each i, and that the algorithm terminates in $O(\log n)$ stages. Hence, we obtain the following:

Theorem 3.4. *Let $\mathcal{R}$ be a set of n regions of constant description complexity in $\mathbb{R}^2$ such that the union complexity of any r of them is $O(r\varphi(r))$, with φ as above. A hitting set for $(\mathbb{R}^2, \mathcal{R})$ of size $O(\kappa\varphi(\kappa)\log n)$, where $\kappa = \kappa(\mathbb{R}^2, \mathcal{R})$, can be computed in $O(n\varphi(\kappa)\log^2 \kappa \log n)$ randomized expected time.*

Remark. We note that this algorithm is significantly simpler than that for the discrete case, since in the latter we need to compute a shallow cutting, locate the points of X_i in Ξ_i, and distinguish between empty and nonempty cells of Ξ_i, all of which are not needed now. Observe that the maximal depth of a point in X_i (in the discrete case) can be considerably different from $\max_{p\in\mathbb{R}^2} \Delta(p, \mathcal{R}_i)$, which is the major reason for constructing, for the discrete case, a shallow cutting, instead of just a "standard" one (that is, with respect to the entire arrangement $\mathcal{A}(\mathcal{R})$).

4 Axis-Parallel Boxes

We now present a near-linear algorithm for computing a hitting set for the range space $(X, \mathcal{R})$ where X is a set of m points in $\mathbb{R}^d$ and $\mathcal{R} = \{R_1, \ldots, R_n\}$ is a collection of n axis-parallel d-dimensional boxes in $\mathbb{R}^d$ (as in the introduction, we refer to them as *d-boxes*, or just *boxes*). That is, we consider here only the discrete model; see below for a remark concerning the continuous model. We follow the same iterative approach as in Section 3, which works in $\lceil \log_2 n \rceil$ stages, but the implementation of these stages is different. At the beginning of the ith stage, we have a subset $\mathcal{R}_i \subseteq \mathcal{R}$, a subset $X_i \subseteq X$, and a hitting set $H_{i-1} = X \setminus X_i$ for $\mathcal{R} \setminus \mathcal{R}_i$; we set $n_i = |\mathcal{R}_i|$ and $m_i = |X_i|$. Initially, $\mathcal{R}_1 = \mathcal{R}$, $X_1 = X$, and $H_0 = \emptyset$. The algorithm terminates when $\mathcal{R}_i = \emptyset$ or $X_i \cap R = \emptyset$ for each $R \in \mathcal{R}_i$. Let $l_i = \max_{p \in X_i} \Delta(p, \mathcal{R}_i)$. For simplicity, we first describe the algorithm for $d = 2$ and then extend it to higher dimensions.

The ith stage (for $d = 2$) proceeds as follows. We first compute l_i (exactly), in $O((m_i + n_i) \log n_i)$ time, using a sweep-line technique [18]. If $l_i = 0$, i.e., $X_i \cap R = \emptyset$ for each $R \in \mathcal{R}_i$, we stop and return H_{i-1}. Otherwise, we sweep $\mathcal{R}_i$ and X_i in the $(+y)$-direction by a horizontal line L. We maintain the intersections of L with the rectangles of $\mathcal{R}_i$, which is a set of intervals, in a dynamic segment tree. When L reaches a point $p \in X_i$, we compute, in $O(\log n_i)$ time, the depth δ_p, namely, the number of rectangles of $\mathcal{R}_i$ that contain p. If $\delta_p \geq l_i/2$, we add p to the ith hitting subset $\bar{H}_i$, delete from $\mathcal{R}_i$ the set $\mathcal{G}_p$ of the rectangles of $\mathcal{R}_i$ that contain p, and update the segment tree accordingly. Note that once a rectangle of $\mathcal{R}_i$ is deleted, it does not contribute to the depth of subsequent points of X_i. Let $\mathcal{G}_i = \bigcup_{p \in \bar{H}_i} \mathcal{G}_p$, and set $\mathcal{R}_{i+1} = \mathcal{R}_i \setminus \mathcal{G}_i$ (the set of the remaining unhit rectangles), $X_{i+1} = X_i \setminus \bar{H}_i$, and $H_i = H_{i-1} \cup \bar{H}_i$. By construction, $\bar{H}_i$ is a hitting set for $\mathcal{G}_i$. Moreover, $\mathcal{G}_p \cap \mathcal{G}_q = \emptyset$, for any pair of points $p \neq q \in \bar{H}_i$, and $|\mathcal{G}_p|, |\mathcal{G}_q| \geq l_i/2$. Following the same argument as in Section 3, we have $\kappa \geq n_i/l_i$ (since each point can hit at most l_i rectangles of $\mathcal{R}_i$), and $|\bar{H}_i| \leq 2n_i/l_i$, so we have $|\bar{H}_i| \leq 2\kappa$. Moreover, $l_{i+1} \leq l_i/2$, because, by construction, no point of $X_i \setminus \bar{H}_i$ can lie in more than $l_i/2$ rectangles of $\mathcal{R}_{i+1}$.

The ith stage for $d > 2$ works exactly as above except that we now sweep a $(d-1)$-dimensional hyperplane L normal to the x_d-axis, in the positive x_d-direction, and maintain the intersection of L with the boxes in $\mathcal{R}_i$, a set of $(d-1)$-boxes, in a dynamic range-searching data structure so that the number of boxes containing a query point can be computed quickly. As shown in [35], the structure can be implemented so that the update and query time are $O(\log^{d-1} n)$. Now the algorithm proceeds as above, to compute $\bar{H}_i$, X_{i+1}, and $\mathcal{R}_{i+1}$. A similar hyperplane-sweep algorithm can be used to compute the value of l_i at the beginning of the ith stage. The total running time of the ith stage is $O((|\mathcal{R}_i| + |X_i|) \log^{d-1} n)$.

If $X_{i+1} = \emptyset$ or $\mathcal{R}_{i+1} = \emptyset$, we return H_i as the hitting set of $(X, \mathcal{R})$. If $\mathcal{R}_{i+1} \neq \emptyset$, then we conclude as in Section 3 that H_i is a hitting set of the range space $(X, \{R \cap X \mid R \in \mathcal{R}\})$ and return H_i. Arguing as above, the algorithm terminates after $O(\log n)$ steps, so we obtain the following:

Theorem 4.1. *Given a set $\mathcal{R}$ of n axis-parallel d-boxes in $\mathbb{R}^d$ and a set $X \subset \mathbb{R}^d$ of m points, a hitting set for $(X, \mathcal{R})$ of size $O(\kappa \log n)$ can be computed in $O((n + m) \log^d n)$ time, where $\kappa = \kappa(X, \mathcal{R})$.*

Remark. The preceding analysis exploits in a strong way the discrete nature of X, and we do not know how to extend the above algorithm to the continuous model without increasing the running

time significantly. As far as we know, the result by Nielsen [37], mentioned in the introduction, is still the best-known near-linear-time approximation algorithm for the continuous model when $\mathcal{R}$ is a set of d-boxes.

5 Iterative Reweighted Sampling Scheme

This section describes a faster implementation of the iterative reweighted sampling scheme of Brönnimann and Goodrich [13], which generalizes an earlier technique of Clarkson [15], to compute a hitting set for $(X, \mathcal{R})$, where X is a set of m points and $\mathcal{R}$ is a set of n axis-parallel d-boxes in $\mathbb{R}^d$. The size of the hitting set is $O(\kappa \log \kappa)$, and the running time is[3] $O^*(m + n + \kappa^{d+1})$ (see the introduction for a discussion on the running time in a naive implementation). The size can be improved to $O(\kappa \log\log \kappa)$, for $d = 2, 3$, using the recent result of Aronov *et al.* [11]. For simplicity, we only describe the algorithms for $d = 2$.

We first briefly recall this general technique. Given a range space $(X, \mathcal{R})$, the goal is to find a small subset $H \subseteq X$ that intersects every range of $\mathcal{R}$. Suppose we know (or guess) the value of κ up to a factor of 2. The algorithm works in phases. It assigns weights to the elements of X, which change during the course of execution. Initially, each $p \in X$ is assigned weight $\omega(p) = 1$. In each phase, the algorithm performs two main steps. It first constructs a $(1/2\kappa)$-net $N \subseteq X$ of the weighted range space $(X, \mathcal{R})$. Next, it determines whether N is a hitting set for $\mathcal{R}$. If the answer is YES, the algorithms returns N and stops. Otherwise, the algorithm finds a "witness" range R that does not intersect N; $\omega(R) \leq \omega(X)/2\kappa$ because N is a $(1/2\kappa)$-net. The algorithm then doubles the weight of each point in $X \cap R$, and repeats the above two steps. The analysis of [13, 15] shows that the algorithm terminates within $g(\kappa) := O(\kappa \log (n/\kappa))$ rounds. Since we do not know the value of κ, we conduct an exponential search for κ. If the algorithm does not stop within $g(\kappa)$ phases for the current guess of κ, we double the value of κ and restart the process.

Instead of constructing a $(1/2\kappa)$-net in each phase, if we construct a set $N \subseteq X$ that is a $(1/2\kappa)$-net with probability at least $1/2$, then the weight of the witness range R may exceed $\omega(X)/2\kappa$. We call a phase *successful* if the weight of R is at most $\omega(X)/2\kappa$. Before proceeding to the next phase, we double the weights of points in $X \cap R$ only if the phase is successful. The same argument shows that the algorithm terminates after $g(\kappa)$ successful phases, and, as argued in [15], the algorithm terminates in an expected number of $g(\kappa)$ phases. We will use this randomized version of the algorithm.

Let $\mathcal{W}$ be the set of "witness" ranges identified by the algorithm until the current phase. The technique needs the following two procedures:

VERIFIER. Given a subset $H \subseteq X$, the procedure returns YES if H is indeed a hitting set for $\mathcal{R}$, or else returns a witness range $R \in \mathcal{R}$ that does not intersect H.

ε-NET GENERATOR. This procedure maintains a subset $\mathcal{W} \subseteq \mathcal{R}$ of "witness" ranges. The weight $\omega(p)$ of a point p in X is 2^{w_p}, where w_p is the number of ranges in $\mathcal{W}$ that contain p. The procedure supports two operations: (i) insert a range into $\mathcal{W}$, and (ii) given a parameter $\varepsilon > 0$,

[3] Here a bound of the form $O^*(X)$ means that the actual bound is $O(X \cdot \mathrm{polylog}(X))$.

return an ε-net N for the *weighted* range space $(X, \mathcal{R}, \omega)$, i.e., a subset N of X such that $R \cap N \neq \emptyset$ for any range $R \in \mathcal{R}$ with $\omega(R) \geq \varepsilon\omega(X)$,

If we implement the verifier and the ε-net generator in a brute-force manner, each phase of the algorithm can be implemented in $O((m+n)\kappa \log \kappa)$ time, since $|\mathcal{W}| = O(\kappa \log n)$ and $|N| = O(\kappa \log \kappa)$. We show that in our setting, where X is a set of m points and $\mathcal{R}$ is a set of n axis-parallel rectangles in $\mathbb{R}^2$, X and $\mathcal{R}$ can be preprocessed in $O^*(m+n)$ time so that each phase can be implemented in $O^*(\kappa^2)$ time, which is considerably faster than the naive implementation for small values of κ.

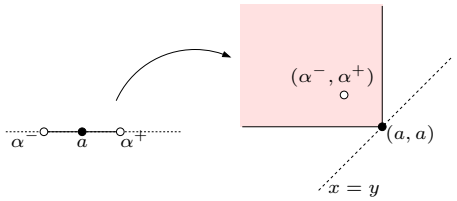

Figure 2. The transformation for the one-dimensional problem. Mapping an interval $[\alpha^-, \alpha^+]$ to the point (α^-, α^+), and a point $a \in \mathbb{R}$ to the quadrant $[-\infty, a] \times [a, \infty]$.

The verifier. We wish to preprocess a set $\mathcal{R} = \{R_1, \ldots, R_n\}$ of n axis-parallel rectangles in $\mathbb{R}^2$ into a data structure, so that we can determine whether a set H of points is a hitting set for $\mathcal{R}$ (or else produce a witness range that misses H). Suppose $R_i = [\alpha_i^-, \alpha_i^+] \times [\beta_i^-, \beta_i^+]$. We map R_i to a point $\pi_i = (\alpha_i^-, \alpha_i^+, \beta_i^-, \beta_i^+) \in \mathbb{R}^4$, and put $\Pi = \{\pi_i \mid 1 \leq i \leq n\}$. We preprocess Π in $O(n \log^4 n)$ time into a 4-dimensional range-searching data structure (see [2, 1]) so that for a query axis-parallel box $B \subseteq \mathbb{R}^4$, we can determine in $O(\log^2 n)$ time whether $B \cap \Pi \neq \emptyset$. If so, then we can also return a point of $B \cap \Pi$.

A point $p = (a, b) \in \mathbb{R}^2$ lies in a rectangle $R_i \in \mathcal{R}$ if $\alpha_i^- \leq a \leq \alpha_i^+$ and $\beta_i^- \leq b \leq \beta_i^+$, i.e., the point π_i lies in the orthant $O_p = [-\infty, a] \times [a, \infty] \times [-\infty, b] \times [b, \infty]$; see Figure 2 for a 1-dimensional illustration. Let $K_H = \mathbb{R}^4 \setminus \bigcup_{p \in H} O_p$. H is a hitting set for $\mathcal{R}$ if and only if $\Pi \subset \bigcup_{p \in H} O_p$, i.e., $K_H \cap \Pi = \emptyset$. Put $h := |H|$. Using a simple variant of the algorithm by Kaplan *et al.* [30], we decompose K_H in time $O(h^2 \log^2 h)$ into a family $\mathcal{B}$ of $O(h^2)$ pairwise-disjoint boxes. We query the range-searching data structure on Π with each box $B \in \mathcal{B}$ and determine, in $O(\log^2 n)$ time, whether $B \cap \Pi = \emptyset$. If the answer is YES for all boxes, we conclude that H is a hitting set for $\mathcal{R}$. Otherwise, there is a box $B \in \mathcal{B}$ for which the query procedure returns a point $\pi_j \in B$. We return the rectangle R_j as the witness rectangle that is not hit by H. We have thus shown:

Lemma 5.1. *$\mathcal{R}$ can be preprocessed in time $O(n \log^4 n)$ into a data structure so that one can determine in $O(h^2 \log^2 n)$ time whether a set H of h points is a hitting set of $(X, \mathcal{R})$.*

The ε-net generator. Let $\mathcal{W}$ be the set of "witness" rectangles maintained by the procedure, and let $s = |\mathcal{W}|$; recall that $s = O(\kappa \log(n/\kappa))$. Let Γ be the set of $4s$ lines that support the edges of the rectangles in $\mathcal{W}$. We maintain the arrangement $\mathcal{A}(\Gamma)$. For each cell $\tau \in \mathcal{A}(\Gamma)$, which is a

rectangle, let $X_\tau = X \cap \tau$, and let ν_τ be the number of rectangles in $\mathcal{W}$ that contain τ. The weight $\omega(p)$ of each point $p \in X_\tau$ is 2^{ν_τ}. Set $\omega(\tau) = \omega(X_\tau) = 2^{\nu_\tau}|X_\tau|$, and $\Omega = \omega(X) = \sum_\tau \omega(\tau)$. For each cell $\tau \in \mathcal{A}(\Gamma)$, we store ν_τ and $\omega(\tau)$. We also maintain two sorted copies of X_τ: $\mathsf{Lx}(X_\tau)$, where the points are sorted by their x-coordinates; and $\mathsf{Ly}(X_\tau)$, where the points are sorted by their y-coordinates. For each point of X_τ, we store cross-pointers between its copies in $\mathsf{Lx}(X_\tau)$ and $\mathsf{Ly}(X_\tau)$, to facilitate the deletion of an item from both lists.

When a new rectangle R is inserted into $\mathcal{W}$, we add the set Γ_R of the four lines supporting the edges of R to Γ one by one, and split each of the cells of $\mathcal{A}(\Gamma)$ that intersect a line of Γ_R, to obtain $\mathcal{A}(\Gamma \cup \Gamma_R)$. If a cell τ is split into two subcells τ^- and τ^+, we first set $\nu_{\tau^-} = \nu_{\tau^+} = \nu_\tau$ and $\omega_{\tau^-} = \omega_{\tau^+} = \omega_\tau$. Next, we split X_τ into the respective subsets X_{τ^-} and X_{τ^+}. This can be accomplished in $O(\min\{|X_{\tau^-}|, |X_{\tau^+}|\})$ time, as follows: Suppose τ is split into τ^- and τ^+ by a vertical line $x = x_0$. We scan the list $\mathsf{Lx}(X_\tau)$ from both ends in lock-step manner until we meet two successive points p_i, p_{i+1} such that $x_0 \in [x(p_i), x(p_{i+1})]$—we split $\mathsf{Lx}(X_\tau)$ at p_i; the two sublists are $\mathsf{Lx}(X_{\tau^-})$ and $\mathsf{Lx}(X_{\tau^+})$. We then construct $\mathsf{Ly}(X_{\tau^-})$ and $\mathsf{Ly}(X_{\tau^+})$ by deleting the elements of X_{τ^-} or X_{τ^+}, whichever is smaller, and inserting them into a new list. A cell may be split into more than just two subcells, a situation which we handle in much the same way. Finally, we increment the value of ν_τ for all those cells that lie inside R and update the values of $\omega(\tau)$ for each cell. See Figure 3. Updating the arrangement and the weights of all the cells takes $O(s^2)$ time. Due to our lock-step traversal, each element of $\mathsf{Lx}(X_\tau)$, $\mathsf{Ly}(X_\tau)$ is processed at most $O(\log m)$ times, and thus the total time spent in splitting the point sets over all insertions is $O(m \log m)$, which we can charge to preprocessing. Hence, the time spent in inserting a rectangle R is $O(s^2 + c_R)$, where c_R represents the amount of time required for splitting at the insertion of R, and thus $\sum_R c_R$ over the entire sequence of insertions is $O(m \log m)$.

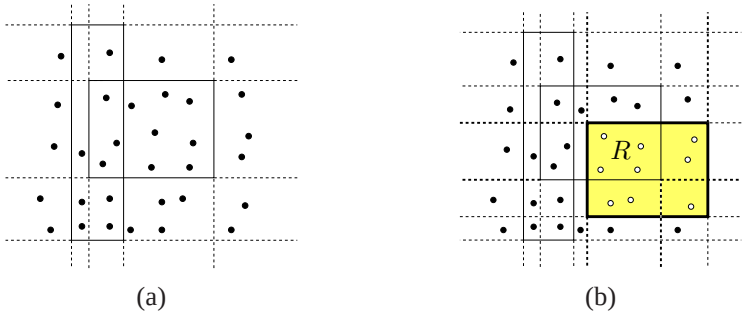

Figure 3. (a) $\mathcal{A}(\Gamma)$ and the sets X_τ. (b) Insertion of the shaded rectangle R into $\mathcal{W}$. The weight of each point inside R (denoted as hollow circles) is doubled.

Given a parameter ε, we construct an ε-net for $(X, \mathcal{R})$ by choosing a random subset $N \subseteq X$ of size $O((1/\varepsilon)\log(1/\varepsilon))$, where each point of X is chosen with probability proportional to its weight. By the ε-net theorem [27], a random sample of this kind is an ε-net with probability at least $1/2$, and thus on average we need to pick at most two samples to guarantee this property. In order to choose a point of X at random, we first randomly choose a cell $\tau \in \mathcal{A}(\Gamma)$ with probability $\omega(\tau)/\Omega$, and then choose a point of X_τ uniformly. Since a cell of $\mathcal{A}(\Gamma)$ is chosen from a non-uniform distribution, we store the cells of $\mathcal{A}(\Gamma)$ at the leaves of a height-balanced tree, where the probability weight of each leaf is the probability weight of its corresponding cell, and the probability weight of each internal node is the corresponding sum of these measures over its children. We then

choose a random cell by traversing a path of this tree (whose nodes are chosen according to their probability weights); see, e.g., [33]. After spending $O(s^2)$ time in processing $\mathcal{A}(\Gamma)$, a random cell of $\mathcal{A}(\Gamma)$ can be chosen in $O(\log s)$ time. Hence, we can compute N in randomized expected time $O(s^2 + (1/\varepsilon)\log(1/\varepsilon)\log s)$. We have thus shown:

Lemma 5.2. *$\mathcal{W}$ and X can be dynamically maintained in a data structure so that, after $O(m \log m)$ preprocessing,*

(i) a rectangle R can be inserted into $\mathcal{W}$ in $O(s^2 + c_R)$ time, where $\sum_R c_R = O(m \log m)$, and

(ii) a subset of X of size $O((1/\varepsilon)\log(1/\varepsilon))$ can be constructed in randomized expected time $O(s^2 + (1/\varepsilon)\log(1/\varepsilon)\log s)$, which is guaranteed to be an ε-net of $(X, \mathcal{R}, \omega)$ with probability at least $1/2$.

An improved ε-net generator in two dimensions. The recent construction by Aronov *et al.* [11] can be adapted to construct an ε-net of $(X, \mathcal{R}, \omega)$ of size $O((1/\varepsilon)\log\log(1/\varepsilon))$ for $d = 2$. This requires building a more sophisticated data structure on $\mathcal{W}$ and X. Specifically, we need a data structure that supports the following four operations:

(i) INSERT(R): Insert a new rectangle R into $\mathcal{W}$.

(ii) WT(R): Return $\omega(R \cap X)$.

(iii) RANK(w): The *rank* of a point $p \in X$ is the sum of the weights of the points lying to the left of p (that is, points with smaller x-coordinates). Return the leftmost point whose rank is at least w.

(iv) RANDOM(t, R): Return t random points from $R \cap X$; each point p is chosen with probability $\omega(p)/\omega(R \cap X)$.

As we will show, this can be done so that INSERT(R) takes $O^*(s^2 + c_R)$ time, where $\sum_R c_R = O(m \log m)$. WT and RANK take $O^*(s)$ time, and RANDOM takes $O^*(s + t)$ time. Before describing the data structure and verifying these statements, we show how the construction of Aronov *et al.* [11] can be implemented efficiently using this structure. Their construction follows a two-level sampling scheme similar to the one described in Section 2 for computing a shallow cutting. Set $r := 2/\varepsilon$, and $\tilde{r} := cr \log\log r$, where $c \geq 1$ is a sufficiently large constant. The construction then proceeds as follows.

I. Choose a random subset $S \subseteq X$, where each point $p \in X$ is chosen with probability $\min\{\tilde{r}\omega(p)/\omega(X), 1\}$. The expected size of S is at most $\tilde{r}$.

II. Partition the plane into a set Σ of r slabs by drawing $r - 1$ vertical lines so that the weight of the points lying in (the interior of) each slab is at most $\lceil \omega(X)/r \rceil$.

III. Build a binary tree $\mathcal{T}$ with r leaves. Each node $v \in \mathcal{T}$ is associated with a slab σ_v. The ith leftmost leaf of $\mathcal{T}$ is associated with the ith leftmost slab of Σ. For an interior node v with children w and z, $\sigma_v = \sigma_w \cup \sigma_z$. Set $S_v = S \cap \sigma_v$. For a non-root node $v \in \mathcal{T}$,

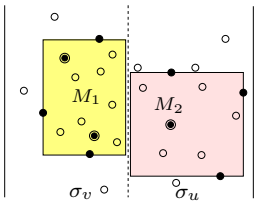

Figure 4. Two siblings nodes u and v; black circles are points in S, the first random sample; M_1 (resp. M_2) is a maximal anchored empty rectangle at v (resp. u); double circles are random samples chosen at the second stage; hollow circles are the remaining points in X.

we call a rectangle M *anchored* at v if $M \subseteq \sigma_v$ and one of its edges lies on the common boundary line $\sigma_v \cap \sigma_u$, where u is the sibling of v (in this case M should lie inside σ_v). We construct the set $\mathcal{M}_v$ of maximal anchored rectangles that do not contain any point of S_v in their interior, i.e., an anchored rectangle $M \in \mathcal{M}_v$ if $\text{int}(M) \cap S_v = \emptyset$ and there is no anchored rectangle M' properly containing M such that $\text{int}(M') \cap S_v = \emptyset$. See Figure 4 for an illustration. Aronov *et al.* [11] show that $|\mathcal{M}_v| = O(|S_v|)$ and that $\mathcal{M}_v$ can be computed in $O(|S_v| \log |S_v|)$ time. Set $\mathcal{M} = \bigcup_{v \in \mathcal{T}} \mathcal{M}_v$; $|\mathcal{M}| = O(|S| \log r)$. The expected size of $\mathcal{M}$ is $O(\tilde{r} \log r)$.

IV. For each $M \in \mathcal{M}$, compute $\omega(M \cap X)$. Let t_M be the integer satisfying $t_M \cdot n/r \leq \omega(M \cap X) < (1+t_M) \cdot n/r$. If $t_M \geq c \log\log(1/\varepsilon)$, then we choose a random subset $S_M \subseteq X \cap M$ of $\mu_M = O(t_M \log t_M)$ points; each point is chosen with probability proportional to its weight. Otherwise $S_M = \emptyset$.

V. Return $N = S \cup \left(\bigcup_{M \in \mathcal{M}} S_M\right)$.

Aronov *et al.* [11] prove that N is an ε-net of $(X, \mathcal{R}, \omega)$ with probability at least $1/2$ and that the expected size of N is at most $(A/\varepsilon) \log\log(1/\varepsilon)$ for an appropriate constant $A > 0$. If $|N| > (2A/\varepsilon) \log\log(1/\varepsilon)$, then we repeat Steps I–IV. So, upon termination of this process, the procedure returns a set N of size $O((1/\varepsilon) \log\log(1/\varepsilon))$ that is guaranteed to be an ε-net of $(X, \mathcal{R}, \omega)$ with probability at least $1/2$. As for the running time, recalling the notation $s = |\mathcal{W}| = O(\kappa \log(n/\kappa))$, Step I can be implemented in $O^*(\tilde{r}+s) = O^*(\kappa+1/\varepsilon)$ time by calling RANDOM$(\tilde{r}, \mathbb{R}^2)$. Step II can be implemented in $O^*(sr) = O^*(\kappa/\varepsilon)$ time by calling the procedure RANK $r-1$ times, and Step III takes $O^*(r) = O^*(1/\varepsilon)$ time. Finally, for each rectangle $M \in \mathcal{M}$, we spend $O^*(s) = O^*(\kappa)$ time to compute $\omega(X \cap M)$, using WT(M), and another $O^*(s + |S_M|) = O^*(\kappa + |S_M|)$ time to construct S_M using RANDOM. Hence, step IV takes

$$O\left(\sum_{M \in \mathcal{M}} (s + |S_M|)\right) = O\left(s|\mathcal{M}| + \sum_{M \in \mathcal{M}} |S_M|\right) = O^*\left(|N| + s \cdot r\right) = O^*(\kappa/\varepsilon)$$

expected time. The total time spent in Steps I–IV is therefore $O^*(\kappa/\varepsilon)$. Since the probability of having to repeat these steps is at most $1/2$, the expected running time of the procedure is $O^*(\kappa/\varepsilon)$.

We now describe the data structure for storing $\mathcal{W}$ and X, which supports efficient implementation of operations (i)–(iv). The structure is an extension of the one for the previous ε-net generator,

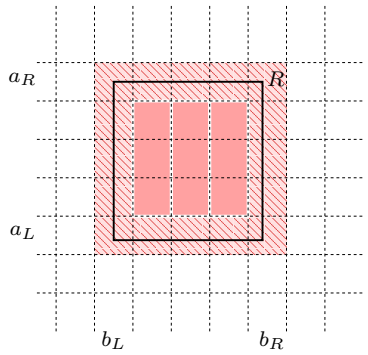

Figure 5. Computing WT(R). Hatched cells intersect ∂R; $\omega(X_\tau \cap R)$ for such a cell τ is computed using the range tree $\mathbb{T}(\tau)$. Darkly shaded cells lie inside R; the weight $\omega_b[a_L + 1, a_R - 1]$ for any "middle" column b is computed using $\mathbb{C}_b$.

and is implemented as follows. Let Γ be the set of the $4s$ lines supporting the edges of the rectangles in $\mathcal{W}$. We maintain $\mathcal{A}(\Gamma)$, which is a $(2s + 1) \times (2s + 1)$ grid; refer once again to Figure 3. When convenient, we will represent a cell of $\mathcal{A}(\Gamma)$ as $\tau_{i,j}$, for $0 \le i, j \le 2s$, where i and j denote, respectively, the row and column indices[4] of the grid containing $\tau_{i,j}$. For each cell $\tau \in \mathcal{A}(\Gamma)$, let X_τ, ν_τ, and $\omega(\tau)$ be as defined before. (When we use the notation $\tau_{i,j}$ to denote a cell, we will use $X_{i,j}$ to denote $X_{\tau_{i,j}}$, with a slight abuse of notation.) In addition to maintaining these quantities, we also preprocess X_τ into a dynamic range-tree data structure $\mathbb{T}(\tau)$ so that $\omega(X_\tau \cap R)$, for a query rectangle R, can be computed in $O(\log^2 m)$ time [19]. This data structure can also choose a random point of $X_\tau \cap R$ in $O(\log m)$ time. For each column j of $\mathcal{A}(\Gamma)$, we maintain a height-balanced binary tree $\mathbb{C}_j$ so that, for an interval $[a^-, a^+]$, $\omega_j[a^- : a^+] = \sum_{i=a^-}^{a^+} \omega(\tau_{i,j})$ can be computed in $O(\log n)$ time. $\mathbb{C}_j$ is also used to choose a random cell among $\tau_{a^-,j}, \ldots, \tau_{a^+,j}$, where each cell is chosen with probability proportional to its weight. With this data structure at our disposal, each of the four desired operations can be performed as follows.

INSERT(R): We first update $\mathcal{A}(\Gamma)$ as earlier. Suppose a cell τ of $\mathcal{A}(\Gamma)$ is split into two cells τ^- and τ^+ so that, say, $|X_{\tau^-}| \le |X_{\tau^+}|$. We obtain $\mathbb{T}(\tau^+)$ from $\mathbb{T}(\tau)$ by deleting the points of X_{τ^-} from $\mathbb{T}(\tau)$, and we construct $\mathbb{T}(\tau^-)$ by building afresh the dynamic range tree on X_{τ^-}. This step takes $O(|X_{\tau^-}| \log^2 |X_{\tau^-}|)$ time. Since $|X_{\tau^-}| \le |X_{\tau^+}|$, each point actively participates in only $O(\log m)$ updates, so the overall cost of these updates is $O(m \log^3 m)$. Next, for each column j of the arrangement, we reconstruct the binary tree $\mathbb{C}_j$ in $O(s)$ time. A rectangle R thus can be inserted in $O^*(s^2 + c_R)$ time, where c_R is defined similarly as in the initial description of the ε-net generator, and $\sum_R c_R = O(m \log m)$.

WT(R): Let τ_{a_L,b_L} (resp., τ_{a_R,b_R}) be the cell of $\mathcal{A}(\Gamma)$ containing the lower left (resp., upper right) vertex of R, i.e., $\tau_{a,b} \cap R \ne \emptyset$ for $a_L \le a \le a_R$ and $b_L \le b \le b_R$. For simplicity, assume

[4]Because of the matrix notation, the row index, which represents the y-order of the cells, precede the column index, which represents their x-order.

that the edges of R do not lie on the lines of Γ. Then

$$\begin{aligned}
\textsc{Wt}(R) &= \sum_{\tau:\tau\cap\partial R\neq\emptyset} \omega(X_\tau \cap R) + \sum_{\tau:\tau\subseteq R} \omega(X_\tau \cap R) \\
&= \sum_{\tau:\tau\cap\partial R\neq\emptyset} \omega(X_\tau \cap R) + \sum_{b_L<b<b_R} \sum_{a_L<a<a_R} \omega(\tau_{a,b}) \\
&= \sum_{\tau:\tau\cap\partial R\neq\emptyset} \omega(X_\tau \cap R) + \sum_{b_L<b<b_R} \omega_b[a_L+1 : a_R-1].
\end{aligned}$$

The weight $\omega(X_\tau \cap R)$ for each τ in the first sum is computed using $\mathbb{T}(\tau)$. The second term is nontrivial only if $a_R > a_L + 1$ and $b_R > b_L + 1$ (i.e., there exists a cell $\tau \in \mathcal{A}(\Gamma)$ such that $\tau \subset R$). For each b, we compute $\omega_b[a_L + 1 : a_R - 1]$ using $\mathbb{C}_b$. See Figure 5. The total time spent is $O^*(s)$.

RANDOM(t,R): We choose a random point of $X \cap R$ in three stages: first choose the column j from which the point is to be chosen, next choose the cell τ within column j from which the point is to be chosen, and finally choose a random point of $X_\tau \cap R$. After $O^*(s)$ preprocessing, each point can be chosen in $O(\log^2 s)$ time. In more detail, let a_L, b_L, a_R, and b_R be the same as in the description of $\textsc{Wt}(R)$. Let

$$\mathsf{W} = \{\omega(X_\tau \cap R) \mid \tau \cap \partial R \neq \emptyset\} \bigcup \{w_b = \omega_b[a_L + 1 : a_R - 1] \mid b_L < b < b_R\}.$$

We store W in a tree so that a random element of W can be chosen in $O(\log s)$ time with probability proportional to its value. If w_b is chosen, then we choose a random cell $\tau_{a,b}$, $a_L < a < a_R$, with respective probability $\omega(\tau_{a,b})/w_b$, using $\mathbb{C}_b$. Finally, we choose a random point of $X_{\tau_{a,b}} \cap R$ using the range tree $\mathbb{T}(X_{\tau_{a,b}})$. If an item $\omega(X_\tau \cap R)$ is chosen from W, we choose a random point of $X_\tau \cap R$ using $\mathbb{T}(X_\tau)$. By repeating this procedure t times, we choose t random points of $X \cap R$. The total time spent is $O^*(s + t)$.

RANK(w): Let $p^* \in X$ be the point of rank w. We find in $O(s)$ time the column j of $\mathcal{A}(\Gamma)$ that contains p^*. Note that $\mathbb{T}(\tau)$ stores the points of X_τ sorted by their x-coordinates. Let $\bar{w}$ be the rank of p^* within column j, i.e., $\bar{w} = w - \sum_{b<j} w(\tau_{a,b})$. Using the monotone matrix searching technique of Fredrickson and Johnson [25], we can choose, in $O^*(s)$ time, a point of rank $\bar{w}$ from the set $\bigcup_i X_{i,j}$. Hence, RANK(w) takes $O^*(s)$ time.

Lemma 5.3. *$\mathcal{W}$ and X can be dynamically maintained in a data structure so that, after $O^*(m)$ preprocessing,*

(i) a rectangle R can be inserted into $\mathcal{W}$ in $O^*(s^2 + c_R)$ time, where $\sum_R c_R = O(m \log^3 m)$, and

(ii) a subset of X of size $O((1/\varepsilon) \log\log(1/\varepsilon))$ can be constructed in randomized expected time $O^*(s/\varepsilon)$, which is guaranteed to be an ε-net of $(X, \mathcal{R}, \omega)$ with probability at least $1/2$.

Putting the pieces together. Returning to the problem of computing a hitting set for $\mathcal{R}$ in our setting, we plug the above procedures into the general machinery of [13, 15]. Observing that $|H| = O(\kappa \log \kappa)$, $|\mathcal{W}| = O(\kappa \log n)$, and $\varepsilon = 1/(2\kappa)$, we conclude that, after $O^*(m + n)$ preprocessing, each phase of the algorithm can be implemented in $O^*(\kappa^2)$ time. Hence, the expected running time

of the overall algorithm is $O^*(m+n+\kappa^3)$. The verifier can be extended to any higher dimension. The ε-net construction by Aronov *et al.* [11] extends to $\mathbb{R}^3$, so Lemma 5.3 can also be extended to $\mathbb{R}^3$. Lemmas 5.1 and 5.2 extend to $d \geq 4$. The expected running time in $\mathbb{R}^d$ is $O^*(m+n+\kappa^{d+1})$. We thus conclude the following:

Theorem 5.4. *Let X be a set of m points in $\mathbb{R}^d$, and let $\mathcal{R}$ be a set of n d-boxes in $\mathbb{R}^d$. A hitting set for $(X, \mathcal{R})$ of size $O(\kappa \log \kappa)$ can be computed in randomized expected time $O^*(m+n+\kappa^{d+1})$, where $\kappa = \kappa(X, \mathcal{R})$. For $d = 2, 3$, the size of the hitting set is $O(\kappa \log\log \kappa)$.*

Remark: We note that with some effort we can speed up the overall running time for the ε-net generator (for any dimension d) to be nearly $O\left(n + \kappa^d\right)$, over all k steps of the algorithm, however, we are not aware of any technique that achieves a similarly improved time bound for the verifier.

6 Conclusions

We presented near-linear algorithms for computing hitting sets of small size, in both discrete and continuous models, for geometric range spaces in which ranges were induced by rectangles or "well-behaved" planar regions. We also described a faster implementation of the iterative weighted sampling scheme by Brönnimann-Goodrich [13] for the discrete case when $\mathcal{R}$ is a set of rectangles. We conclude by mentioning two open problems:

- Is there a near-linear algorithm to compute a hitting set of a set of triangles in $\mathbb{R}^2$ whose size is within a polylogarithmic factor of the optimal size?
- Is there an efficient dynamic data structure for maintaining a small-size hitting set under insertions and deletions, even when ranges are induced by simple regions such as squares or disks? Agarwal *et al.* [9] described such a data structure for the one-dimensional case.

Acknowledgments

We thank Sariel Har-Peled for several useful discussions, which formed the basis of the algorithm described in Section 4. We also thank two anonymous referees and the editor for their helpful comments.

References

[1] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3d, and higher dimensional improvements. *Proc. 26th Annu. Sympos. Comput. Geom.*, 2010, 240–246.

[2] P. K. Agarwal. Range searching. *Handbook of Discrete and Computational Geometry*, 2nd edition, (J. Goodman and J. O'Rourke, eds.), CRC Press, New York, 2004, 809–838.

[3] P. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM J. Comput.*, 27 (1998), 654–667.

[4] P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29 (2000), 912–953.

[5] P. K. Agarwal, E. Ezra, and S. Ganjugunte. Efficient sensor placement for surveillance problems. *Proc. 5th IEEE Internat. Conf. Distrib. Comput. Sensor Sys.*, 2009, 301–314.

[6] P. K. Agarwal, J. Matoušek, and O. Schwarzkopf. Computing many faces in arrangements of lines and segments. *SIAM J. Comput.*, 27 (1998), 491–505.

[7] P. K. Agarwal, J. Pach and M. Sharir. State of the union (of geometric objects). *Surveys on Discrete and Computational Geometry*, (J. Goodman, J. Pach and R. Pollack, eds.), American Mathematical Society, Providence, RI, 2008, 9–48.

[8] P. K. Agarwal and M. Sharir. Arrangements and their applications. *Handbook of Computational Geometry*, (J. Sack and J. Urrutia, eds.), Elsevier, Amsterdam, 2000, 49–119.

[9] P. K. Agarwal, J. Xie, J. Yang, and H. Yu. Input-sensitive scalable continuous join query processing. *ACM Trans. Database Syst.*, 34 (2009), 1–41.

[10] B. Aronov, M. de Berg, E. Ezra, and M. Sharir. Improved bound for the union of locally fat objects in the plane. unpublished manuscript, 2011.

[11] B. Aronov, E. Ezra, and M. Sharir, Small-size ε-nets for axis-parallel rectangles and boxes. *SIAM J. Comput.*, 39 (2010), 3248–3882.

[12] B. Aronov and S. Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38 (2008), 899–921.

[13] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC dimensions. *Discrete Comput. Geom.*, 14 (1995), 463–479.

[14] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46 (2003), 178–189.

[15] K. L. Clarkson. Algorithms for polytope covering and approximation. *Proc. 3rd Workshop on Algorithms and Data Structures*, 1993, 246–252.

[16] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4 (1989), 387–421.

[17] K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete Comput. Geom.*, 37 (2007), 43–58.

[18] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. 3rd ed., Springer-Verlag, Berlin, 2008.

[19] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.* 17 (1988), 427–462.

[20] H. Edelsbrunner, L. Guibas, J. Hershberger, J. Pach, R. Pollack, R. Seidel, M. Sharir, and J. Snoeyink. On arrangements of Jordan arcs with three intersections per pair. *Discrete Comput. Geom.*, 4 (1989), 523–539.

[21] G. Even, D. Rawitz, and S. Shahar. Hitting sets when the VC-dimension is small. *Inform. Process. Letts.*, 95 (2005), 358–362.

[22] E. Ezra, B. Aronov and M. Sharir. Improved bound for the union of fat triangles. *Proc. 22nd Annu. ACM–SIAM Sympos. Discrete. Algo.*, 2011, 1778–1785.

[23] U. Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45 (1998), 634–652.

[24] R. Fowler, M. Paterson, and S. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Letts.*, 12 (1981), 133–137.

[25] G. N. Frederickson and D. B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM J. Comput.*, 13 (1984), 14–30.

[26] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[27] D. Haussler and E. Welzl. ε-nets and simplex range queries. *Discrete Comput. Geom.*, 2 (1987), 127–151.

[28] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32 (1985), 130–136.

[29] S. Laue. Geometric set cover and hitting sets for polytopes in $\mathbb{R}^3$. *Proc. 25th Int. Sympos. Theoret. Aspects Comput. Sci.*, 2008, 479–490.

[30] H. Kaplan, N. Rubin, M. Sharir, and E. Verbin. Efficient colored orthogonal range counting. *SIAM J. Comput.*, 38 (2008), 982–1011.

[31] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1 (1986), 59–71.

[32] J. King and D. Kirkpatrick. Improved approximation for guarding simple galleries from the perimeter. In arXiv:1001.4231.

[33] D. E. Knuth and A. C-C. Yao. The complexity of nonuniform random number generation. in *Algorithms and Complexity: New Directions and Recent Results* (J. F. Traub, ed.), 357–428, Academic Press, New York, NY, 1976.

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

[35] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica*, 5 (1990), 215–241.

[36] N. H. Mustafa and S. Ray. PTAS for geometric hitting set problems via local search. *Proc. 25th Annu. ACM Sympos. Comput. Geom.*, 2009, 17–22.

[37] F. Nielsen. Fast stabbing of boxes in high dimensions. *Theoret. Comput. Sci.*, 246 (2000), 53–72.

[38] J. Pach and G. Tardos. On the boundary complexity of the union of fat triangles. *SIAM J. Comput.*, 31 (2002), 1745–1760.

[39] J. Pach and G. Tardos. Tight lower bounds for the size of ε-nets. *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, 2011, to appear.

[40] E. Pyrga and S. Ray. New existence proofs for ε-nets. *Proc. 24th Annu. Sympos. Comput. Geom.*, 2008, 199–207.

[41] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

[42] K. Varadarajan. Epsilon nets and union complexity. *Proc. 25th Annu. ACM Sympos. Comput. Geom.*, 2009, 11–16.

[43] V. V. Vazirani, *Approximation Algorithms*. Springer-Verlag, New York, 2001.