Near-Linear Approximation Algorithms for Geometric Hitting Sets*

Pankaj K. Agarwal
Duke University, Durham
NC 27708-0129, USA

Esther Ezra
Duke University, Durham
NC 27708-0129, USA

Micha Sharir
Tel Aviv University, Tel Aviv
69978 Israel, and
Courant Institute, New York
New York, NY 10012, USA

Abstract

Given a set system $(X, \mathcal{R})$, the *hitting set* problem is to find a smallest-cardinality subset $H \subseteq X$, with the property that each range $R \in \mathcal{R}$ has a non-empty intersection with H. We present near-linear time approximation algorithms for the hitting set problem, under 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-rectangles in $\mathbb{R}^d$. In both cases X is either the entire d-dimensional space or a finite set of points in d-space. The approximation factors yielded by the algorithm are small; they are either the same as or within an $O(\log n)$ factor of the best factors known to be computable in polynomial time.

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Analysis of Algorithm and Problem Complexity Nonnumerical Algorithms and Problems [Computations on discrete structures, geometrical problems and computations]

General Terms

Algorithms, Theory.

Keywords

Geometric range spaces, Hitting sets, Cuttings, Approximation algorithms.

*Work on this paper by Pankaj K. Agarwal and Micha Sharir has been supported by a joint grant from the US-Israel Binational Science Foundation. Work on this paper by Pankaj K. Agarwal and Esther Ezra has been supported by NSF under grants CNS-05-40347, CFF-06-35000, and DEB-04-25465, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, and by an NIH grant 1P50-GM-08183-01. Work by Micha Sharir has also been supported by NSF Grants CCF-05-14079 and CCF-08-30272, by Grant 155/05 from the Israel Science Fund, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

SCG'09, June 8–10, 2009, Aarhus, Denmark.

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 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 $\mathbb{R}^d$. Given a range space $(\mathbb{R}^d, \mathcal{R})$ of the above kind, the geometric *hitting-set* problem is to find a smallest set $H \subset \mathbb{R}^d$ of points that intersects all regions of $\mathcal{R}$; we call H a *hitting set* for $\mathcal{R}$. Several applications 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 this range space 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*. Let $\kappa := \kappa(X, \mathcal{R})$ denote the size of the smallest subset of X that is a hitting set of $(X, \mathcal{R})$.

The hitting-set problem is NP-Complete even for very simple geometric settings in $\mathbb{R}^2$, e.g., when $\mathcal{R}$ is a set of unit disks or squares [26, 27]. 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, 41]. In such situations, it is desirable 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.

Previous results. The well-known greedy algorithm gives a polynomial-time $(\log n)$-approximation for computing a hitting set [42], 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 [25]. However, by exploiting the underlying geometry, one can obtain polynomial-time algorithms with better approximation ratio 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 refer the reader to [11, 12, 13, 14, 29] and the references therein. We only mention a few results that are directly relevant to our results.

A polynomial-time $(1+\varepsilon)$-approximation algorithm, for any $\varepsilon > 0$, is known under the continuous model if $\mathcal{R}$ is a set of "fat" objects [13]; see also [29]. However these techniques do not extend to the discrete model. Given a parameter $0 < \varepsilon < 1$, an *ε-net* for a (finite) range space $(X, \mathcal{R})$ 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 [28], a range space of finite VC-dimension (see [28] for the definition) has an ε-net of size $O((1/\varepsilon)\log(1/\varepsilon))$. 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 and three dimensions, pseudo-disks in the plane, or translates of a fixed convex polytope in 3-space; see [17, 30, 33, 34, 39]. Recently, Aronov *et al.* [10] 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. Generalizing a technique by Clarkson [15], Brönnimann and Goodrich [12] 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 $O((1/\varepsilon)\varphi(1/\varepsilon))$ 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 [12] 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 linear-size ε-net. Several other instances with improved bounds are listed in [10].

Both the greedy and the Brönnimann–Goodrich 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 $O((|X|+|\mathcal{R}|)\text{polylog}(|\mathcal{R}|))$ (expected) time per stage if the complexity of the union of the regions in $\mathcal{R}$ is near linear [11]. Similarly, a straightforward implementation of the Brönnimann–Goodrich algorithm takes $\Omega(|X|+|\mathcal{R}|)$ time (per stage). Thus the bounds on the running time of these approaches is $\Omega((|X|+|\mathcal{R}|)\kappa\log n)$. It is not clear whether the running of these algorithms can be improved to $O((|X|+|\mathcal{R}|)\text{polylog}(|\mathcal{R}|))$ or even to $O((|X|+|\mathcal{R}|+\kappa^{O(1)})\text{polylog}(|\mathcal{R}|))$. Although near-linear algorithms are known for computing a hitting set under the continuous model in several special cases, such as an $O(1)$-approximation algorithm when the objects are fat [13], no such algorithms are known under the discrete model even for very simple cases, e.g., the case of disks in the plane.

Our results. In this paper we present near-linear algorithms for computing a hitting set of approximately optimal size, under both the discrete and continuous models. However, 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, which will be used by our hitting-set algorithms. Although there are known algorithms to construct shallow cuttings [4, 33], we do not know how to adapt them to support the additional procedures required in our scenario. We thus had to resort to a different approach to construct shallow cuttings, which is randomized incremental and is based on a variant of of the algorithm of Agarwal *et al.* [3] for constructing levels in arrangements. As a by-product result, 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 and X is a set of m points in the plane.[1] For simplicity, we assume throughout the paper that X hits all the regions in $\mathcal{R}$. This assumption can easily be relaxed because our algorithm can detect when X does not hit all the regions. We further assume that the union complexity of any subset of $\mathcal{R}$ of size r is $O(r\varphi(r))$, where $\varphi(r)$ is a slowly growing function (ideally, $o(\log r)$ or at worst $O(\log r)$). We describe in Section 3 a randomized algorithm that computes a hitting set $H \subseteq X$ for $\mathcal{R}$ of size $O(\kappa\varphi(\kappa)\log n)$, in expected time $O((m\log\kappa + n\varphi(\kappa)\log^2\kappa)\log n)$. 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 cause the algorithm to terminate within $\log n$ stages. In contrast, the greedy algorithm terminates in $O(\kappa\log n)$ stages and chooses one point at each stage. We obtain a faster algorithm by reducing the number of stages, but have to pay an extra $\varphi(\kappa)$ factor in the output size. There are various classes of planar regions with small union complexity; see [7] for a comprehensive survey. Here we just mention that we obtain near-linear hitting-set algorithms, with small approximation factors of the above kind, for (i) pseudo-disks [32], with approximation factor $O(\log n)$, (ii) fat wedges and fat triangles [38], with respective approximation factors $O(\log n)$ and $O(\log n\log\log\kappa)$, (iii) locally γ-fat objects [19], with approximation factor $O(\log^2(\kappa)\beta_{s+2}(\kappa)\log n)$, where s is a constant that depends on the description complexity of the objects, and $\beta_t(q) = \lambda_t(q)/q$, where $\lambda_t(q)$ is the maximal length of Davenport-Schinzel sequences of order t on q symbols (see [40]), (iv) regions bounded by Jordan arcs having three intersections per pair [22], with approximation factor $O(\alpha(\kappa)\log n)$, where $\alpha(\cdot)$ is the (extremely slowly growing) inverse Ackermann function.

We next consider in Section 4 the case in which $\mathcal{R}$ is a set of n axis-parallel rectangles and X a set of m points in the plane. 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 $O((m+n)\log^2 n)$. It turns out that the continuous problem is considerably simpler in this case: a hitting set of size $O(\kappa\log^{d-1}\kappa)$ for a set of n orthogonal rectangles in $\mathbb{R}^d$ can be computed in $O(n\log^{d-1} n)$ time by a much simpler algorithm (the one-dimensional problem has an exact solution, computable in $O(n\log n)$ time).

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-rectangles in $\mathbb{R}^d$.

[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.

Given a set X of m points and a set $\mathcal{R}$ of n axis-parallel d-rectangles in $\mathbb{R}^d$, we can compute a hitting set $H \subseteq X$ for $\mathcal{R}$ of $O(\kappa \log \kappa)$ size in $O((m+n+\kappa^{d+1})\text{polylog}(mn))$ randomized expected time. The main ingredient of our algorithm is a data structure that, after $O((m+n)\text{polylog}(mn))$ preprocessing time, can implement each stage of the Brönnimann–Goodrich algorithm in $O(\kappa^d \text{polylog}(mn))$ time. We improve the approximation factor to $O(\log \log \kappa)$, for $d = 2, 3$, using the recent result by Aronov *et al.* [10], while keeping the expected running time $O((m+n+\kappa^{d+1})\text{polylog}(mn))$.

2. ARRANGEMENTS AND 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 $\partial\mathcal{U}(\mathcal{R})$. For $r \leq n$, let $f(r)$ denote the maximum complexity of the union of a subset of r regions in $\mathcal{R}$, measured as above. We assume that $f(r)$ is nearly linear, i.e., $f(r) \leq r\varphi(r)$, where $\varphi(\cdot)$ is a slowly growing function (e.g., $\varphi(r) = \ln r$). We refer to such a set of regions as *well behaved* — they have constant description complexity and the complexity of the union of any of their subset is near linear.

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; thus, 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 and edges of $\mathcal{A}(\mathcal{R})$ that lie in $\mathcal{A}_{\leq l}(\mathcal{R})$. A well-known result by Clarkson and Shor [16] implies that the complexity of $\mathcal{A}_{\leq l}(\mathcal{R})$ is $O(l^2 f(n/l)) = O(nl\varphi(n/l))$. 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.

Vertical decomposition. The *vertical decomposition* of a cell C in $\mathcal{A}(\mathcal{R})$, denoted by $C^{\shortparallel}$, is obtained by erecting, within C, a vertical line through each vertex and x-extremal point on ∂C until it meets ∂C again (or else extends to ∞). $C^{\shortparallel}$ 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 may have fewer than four edges. The number of pseudo-trapezoids in $C^{\shortparallel}$ is proportional to the number of vertices in C. The vertical decomposition of $\mathcal{A}(\mathcal{R})$ or $\mathcal{A}_{\leq l}(\mathcal{R})$, denoted by $\mathcal{A}^{\shortparallel}(\mathcal{R})$ and $\mathcal{A}^{\shortparallel}_{\leq l}(\mathcal{R})$, respectively, is obtained by computing the vertical decomposition of each of its cells. The number of pseudo-trapezoids in $\mathcal{A}^{\shortparallel}_{\leq l}(\mathcal{R})$ is $O\left(nl\varphi(n/l)\right)$; see Figure 1.

A pseudo-trapezoidal cell 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 $\mathcal{A}^{\shortparallel}(\mathcal{D}(\tau))$. We call a pseudo-trapezoid $\tau \subseteq \mathbb{R}^2$ *primitive* if there exists a set $D = \mathcal{D}(\tau)$ of at most four regions of $\mathcal{R}$ such that $\tau \in \mathcal{A}^{\shortparallel}(D)$.

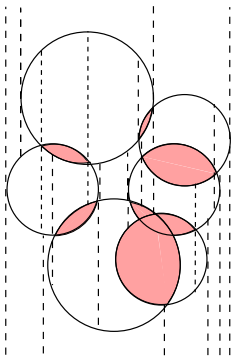

Figure 1. The decomposition $\mathcal{A}^{\shortparallel}_{\leq 1}$ of an arrangement of disks. The level of the shaded regions ≥ 2.

Shallow cuttings. Let $l \geq 0$ and $1 \leq r \leq n$ be two parameters. A collection Ξ of (possibly unbounded) pairwise-disjoint primitive cells 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 at most n/r boundary curves of $\mathcal{R}$ cross each cell $\tau \in \Xi$. The *size* of Ξ is the number of its cells. The set of boundary curves that cross a cell $\tau \in \Xi$, denoted by $\mathcal{R}_\tau$, is called the *conflict list* of τ.

The notion of shallow cutting was originally introduced by Matoušek [33] for a set of halfspaces in $\mathbb{R}^d$, and this notion was subsequently extended to other regions (see e.g. [4]). It is shown in [33] (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 $q = l(r/n) + 1$.

The proof of the above property is constructive and based on a two-level random sampling. Namely, 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, let $\mathcal{R}_\tau$ be its conflict list. 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 $t > 1$, 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, and add the clipped cell to Ξ if it intersects $\mathcal{A}_{\leq l}(\mathcal{R})$. It is shown that Ξ is a $(1/r)$-cutting of $\mathcal{A}_{\leq l}(\mathcal{R})$ and that its expected size is $O(qr\varphi(r/q))$.

Our hitting-set algorithm will construct a shallow cutting and perform some additional steps on the cutting. We therefore describe an efficient algorithm to construct a shallow cutting, based on the two-stage sampling scheme described in [33, 4], which uses a variant of the randomized incremental construction of Agarwal *et al.* [3] for constructing $\mathcal{A}_{\leq l}(\mathcal{R})$. This approach is more suitable for our purposes than the ones described in [33, 4], since it aids us to track additional information that we need to maintain—see below.

Let r and l be two 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. The algorithms maintains the following structure after the first i steps:

(i) A set Ξ_i of cells of $\mathcal{A}^{\shortparallel}(\mathcal{R}_i)$ that potentially intersect $\mathcal{A}_{\leq l}(\mathcal{R})$.

(ii) The *conflict lists*: For every (primitive) cell $\tau \in \Xi_i$, we

store the set $\mathcal{R}_\tau \subseteq \mathcal{R} \setminus \mathcal{R}_i$ whose boundary curves cross τ, and, for every region $R \in \mathcal{R} \setminus \mathcal{R}_i$, we attach 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 maintain the level l_τ of one of its (arbitrary chosen) interior points p with respect to $\mathcal{A}(\mathcal{R})$.

The algorithm maintains the invariant that $l_\tau \leq l + n_\tau$. We refer to a cell satisfying this condition as *active*.

Suppose we have inserted the regions in $\mathcal{R}_{i-1}$ and we have the above structure. We insert the region R_i and update the structure as follows.

(i) We find all cells of Ξ_{i-1} intersected by R, using the conflict lists. We split these cells, re-decompose them into primitive cells if necessary, update the conflict lists, and compute the level information for the new cells.

(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.

We refer the reader to the original paper [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 size of their respective conflict lists. A cell destroyed in step i must have been created in an earlier step, so it suffices to bound the number of cells created in step i and the total size of their conflict lists. Following the analysis in [3], the expected time spent in step i is

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

$$= O\left(n/i^2 + \varphi(n/l)(l + (n/i)\log i)\right).$$

The above bound is obtained by the following intuitive (and imprecise) argument. At the ith step, the set of active cells of $A(\mathcal{R}_i)$ lie at level at most $h = l + n_\tau$, and, due to the analysis of Clarkson-Shor [16], $n_\tau = O((n/i)\log i)$. The complexity of $\mathcal{A}_{\leq h}(\mathcal{R})$ is $O(nh\varphi(n/l))$. The probability of a vertex of $\mathcal{A}_{\leq h}(\mathcal{R})$ 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 ith-step is roughly $O(1/i)$. Putting this together, one can deduce that the size of the conflict lists of the newly created cells at step i is bounded by (1). Summing over all r steps, the total expected running time of the first stage of the algorithm is $O((n\log^2 r + rl)\varphi(n/l))$.

Put $\Xi' := \Xi_r$. For each cell $\tau \in \Xi'$, let $n_\tau = |\mathcal{R}_\tau|$. Next, we refine each cell of Ξ' as follows. Suppose $(t-1)n/r < n_\tau \leq tn/r$. We refer to t as the *excess* in τ. If $t = 1$, we add τ to the final cutting Ξ. Otherwise, we apply the second sampling step on τ by constructing a $(1/t)$-cutting (here this is just the "standard" cutting) for $\mathcal{R}_\tau$. This is done by choosing a random sample S_τ of $c't\log t$ regions of $\mathcal{R}_\tau$, for some sufficiently large constant $c' > 0$. We maintain the primitive cells induced by S_τ (confined within τ) that meet $\mathcal{A}_{\leq l}(\mathcal{R})$ along with their conflict lists. The primitive cells (clipped within τ) computed by this second stage are then added to the final cutting Ξ. A standard Clarkson-Shor analysis [16] shows that the expected time spent on τ in the second stage is $O(t\log tn_\tau) = O(t^2\log tn/r)$, and the number of cells created is $O(t^2\log^2 t)$.

For $t \geq 0$, let $\eta(r,t,l)$ denote the expected number of cells in Ξ' with excess at least t. The following two bounds on $\eta(r,t,l)$, proved in [4, 33, 6][2], bound the expected size of the final cutting and the expected running time of the algorithm:

$$\eta(r,0,l) = O\left(qr\varphi(r/q)\right) \quad \text{and} \quad \eta(r,t,l) = O\big(2^{-t}\eta(r/t,0,l)\big),$$

where $q = 1 + lr/n$. Summing the expected running time of the second stage over all cells τ and using the above inequalities, we obtain that the total expected time of the second stage is:

$$\sum_{t\geq 1} O\left(\frac{t^2\log tn}{r}\cdot\eta(r,t,l)\right) = \sum_{t\geq 1} O\left(\frac{t^2\log tn}{2^t r}\cdot\eta(r/t,0,l)\right)$$

$$= O\left(\frac{n}{r}\cdot\eta(r,0,l)\right) = O((n+lr)\varphi(n/l)).$$

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

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

Similarly, one can show that the total number of cells in Ξ is $O(qr\varphi(r/q))$. Hence we obtain the following:

LEMMA 2.1. *Let $\mathcal{R}$ be a set of n well-behaved planar regions, let $l \geq 0, r \geq 1$ be two parameters, and let $q = 1+lr/n$. A $(1/r)$-cutting for $\mathcal{A}_{\leq l}(\mathcal{R})$ of size $O(qr\varphi(r/q))$, along with the conflict list of each cell, can be computed in expected time $O((n\log^2 r + lr)\varphi(n/l))$.*

Remarks: (1) As indicated by the analysis, the overwhelming majority of the running time is dominated by the first stage of the algorithm.
(2) The algorithm in [3] inserts all elements $R_1, \ldots, R_n$ (and thus $r = n$ in this case) in order to obtain the final structure $\mathcal{A}_{\leq l}(\mathcal{R})$, whereas ours stops after a prespecified number of steps to obtain the shallow cutting.

Next, we describe two extensions of this algorithm, each of which will be crucial for our hitting-set algorithm. We can construct a directed acyclic graph, called a *history dag* and denoted by $\mathbb{H}$, whose nodes are the 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 by destroying τ'. It is well known that the out-degree of each node is $O(1)$ and that the expected length of any path is $O(\log r)$ [37]. Let X be a set of m points in $\mathbb{R}^2$. By tracing a path through $\mathbb{H}$, we can compute the cell of Ξ containing a point $p \in X$ if such a cell exists. We can thus locate all points of X in Ξ in expected time $O(m\log r)$. Esther: It was $O(m\log n)$ by mistake (?) Let $\Xi^* \subseteq \Xi$ be the set of *nonempty* cells, which contain at least one point of X. Again, by using the history dag $\mathbb{H}$, we can compute the regions of $\mathcal{R}$ that contain at least one cell of Ξ^*. We thus obtain the following:

COROLLARY 2.2. *Let R and Ξ be as defined above, and let X be a set of m points in $\mathbb{R}^2$. We can compute in $O(m\log r+$*

[2]In the first two citations this lemma is proved under the repetition model. The reader is referred to the analysis given in [6, 20] for a proof of this lemma (or some of its variants) under our model.

$(n \log^2 r + lr)\varphi(n/l))$ *expected time: (i)* $X \cap \tau$ *for each* $\tau \in \Xi$*, and (ii) the subset of regions of* $\mathcal{R}$ *that contain at least one nonempty cell of* Ξ*.*

Esther: The bound in the corollary was $\varphi(r)$ instead of $\varphi(n/l)$. I fixed that.

Esther: Pankaj, please check the revised par below.

Finally, by using the above shallow-cutting algorithm and Corollary 2.2, we can compute a value δ such that the maximum depth of any point in X is at most 2δ. This is done by applying a decreasing exponential search on δ, while beginning with $\delta = n/2$, and choosing $l = \delta$, $r = n/l$. At each step of the search, we construct a shallow cutting Ξ, with the above parameters and use the history dag to track the depth of each point in X. More specifically, for each nonempty cell $\tau \in \Xi^*$, we maintain the number of regions N_τ that contain τ (that is, its depth with respect to the regions in $\mathcal{R} \setminus \mathcal{R}_\tau$). We terminate the search when $N_{\tau^0} \geq n_{\tau^0}$ (recall that n_τ is the size of the conflict list of τ), where τ^0 is the cell of Ξ^* with the largest value N_τ, over all cells in Ξ^*. Clearly, this implies that the actual largest depth is approximated within a factor of 2. A closer inspection of the algorithm presented above shows that we need not reconstruct Ξ at each step from scratch. Indeed, since the algorithm for the first sampling stage is incremental, we can update Ξ by inserting (in a random order) the remaining regions and updating Ξ using the conflict lists in a similar manner as above. It is only the second sampling stage which has to be applied from scratch at each step of the search. Nevertheless, since the running time of that stage is only $O((n + lr)\varphi(n/l))$, which is bounded by $O(n\varphi(n/l))$ for our choice of l, r, and the overall number of steps of the search is $O(\log(n/\delta))$, this is subsummed by the time bound for the first sampling stage, as is easily verified (see once again the bound in (2)). In addition, when we begin a new step of the search, we need to eliminate all cells (generated at previous steps) that are no longer active. This is because the level l decreases by a half. Recall that the overall number of cells generated at each step is $O(qr\varphi(r/q))$, which is bounded by $O(r\varphi(r)) = O(n/l\varphi(n/l))$ for our choice of r, and since we double r at each step, the overall running time to test these cells is proportional to the number of these cells obtained at the last iteration of the algorithm. We thus conclude the following:

COROLLARY 2.3. *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 a value* $\delta \in [\delta^*/2, \delta^*]$*, in expected time* $O\left(m \log(n/\delta^*) + n\varphi(n/\delta^*) \log^2(n/\delta^*)\right)$*.*

3. WELL-BEHAVED REGIONS

In this section we describe our hitting-set algorithm for a set $\mathcal{R} = \{R_1, \ldots, R_n\}$ of n well behaved regions in $\mathbb{R}^2$. We first consider the discrete model in which the hitting points are to be taken from a finite set $X \subset \mathbb{R}^2$ of size m, and then consider the continuous model in which $X = \mathbb{R}^2$.

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 $\mathcal{R} \setminus \mathcal{R}_i$. 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 $X_i = \emptyset$. The ith stage consists of the following steps:

Set $l_i = \max_{p \in X_i} \Delta(p, \mathcal{R}_i)$. By Corollary 2.3, we can compute, in $O(m_i \log(n_i/l_i) + n_i\varphi(n_i/l_i) \log^2(n_i/l_i))$ expected time, an integer $h_i \in [l_i/2, l_i]$. Put $r_i := \max\{n_i, cn_i/h_i\}$, 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 $O(r_i\varphi(r_i))$. Note that if $h_i \leq c$, we can simply take Ξ_i to be the vertical decomposition $\mathcal{A}^{\shortparallel}(\mathcal{R}_i)$ of $\mathcal{A}(\mathcal{R}_i)$. 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 $\tau \in \Xi_i$, let $X_i^{(\tau)} = X_i \cap \tau$, 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; $|\bar{H}_i| = O(r_i\varphi(r_i)) = O((n_i/l_i)\varphi(n_i/l_i))$. By Corollary 2.2, $\bar{H}_i$ can be computed (along with Ξ_i^*) in $O(m_i \log(n_i/l_i) + n_i\varphi(n_i/l_i) \log^2(n_i/l_i))$ time. Set $H_i = H_{i-1} \cup \bar{H}_i$ and $X_{i+1} = X_i \setminus \bar{H}_i$.

Using Corollary 2.2, we also compute the set $\mathcal{G}_i \subseteq \mathcal{R}_i$ of regions that fully contain a cell of Ξ_i^*, and then augment it by scanning the conflict lists $\mathcal{R}_\tau$, for each cell $\tau \in \Xi_i^*$ (recall that each such list consists of those regions whose boundaries cross τ), and by adding a region $R \in \mathcal{R}_\tau$ to $\mathcal{G}_i$ if $p_\tau \in R$. It is clear that $\bar{H}_i$ is a hitting set for the augmented $\mathcal{G}_i$, so we set $\mathcal{R}_{i+1} = \mathcal{R}_i \setminus \mathcal{G}_i$. Note that no region of $\mathcal{R}_{i+1}$ contains a point of $\bar{H}_i$. Putting everything together, the expected running time of the ith stage is $O(m_i \log(n_i/l_i) + n_i\varphi(n_i/l_i) \log^2(n_i/l_i))$. As mentioned above, the algorithm terminates when $X_i = \emptyset$ or $\mathcal{R}_i = \emptyset$. Suppose the algorithm terminates after k steps. If $\mathcal{R}_{k+1} = \emptyset$, we return $H = H_k$ as the hitting set for $\mathcal{R}$. If $X_{k+1} = \emptyset$ and $\mathcal{R}_{k+1} \neq \emptyset$, then we conclude that no point of X lies inside any region of $\mathcal{R}_{k+1}$, and thus a hitting set for $(X, \mathcal{R})$ does not exist. The next two lemmas bound the value of k and the size of H.

LEMMA 3.1. *The algorithm terminates within at most* $\lceil \log_2 n \rceil$ *stages.*

PROOF. We claim that for any $j > 1$, $l_j \leq l_{j-1}/2$, 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 $(j-1)$th 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 of p, q lie in τ, the boundary of R intersects τ and thus $R \in \mathcal{R}_\tau$. Since Ξ_{j-1} is a $(1/r_{j-1})$-cutting, there are at most $n_{j-1}/r_{j-1} \leq h_{j-1}/c \leq l_{j-1}/2$ such regions, therefore $\Delta(p, \mathcal{R}_j) \leq l_{j-1}/2$, as claimed. □

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

PROOF. Let κ_j be the size of the smallest subset of X_j that meets all regions of $\mathcal{R}_j$. Observe that $\kappa_j \geq n_j/l_j \geq r_j/(2c)$, since each point of X_j can intersect at most $l_j \leq 2h_j$ elements of $\mathcal{R}_j$. Since $|\bar{H}_j| \leq |\Xi_j| = O(r_j\varphi(r_j))$, we obtain $|\bar{H}_j| = O(\kappa_j\varphi(\kappa_j))$. 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, $|\bar{H}_j| = O(\kappa\varphi(\kappa))$, implying that $|H| = O(\kappa\varphi(\kappa) \log n)$. □

Putting everything together, we obtain the following:

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 $O(r\varphi(r))$, 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 $O((m\log\kappa + n\varphi(\kappa)\log^2\kappa)\log n)$.*

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 to 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 $l_i = \max_{p\in\mathbb{R}^2} \Delta(p, \mathcal{R}_i)$, compute a value $h_i \in [l_i/2, l_i]$, and set $r_i = cn_i/h_i$, 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 simpler to do than to construct a shallow cutting). 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. We note that this algorithm is significantly simpler than that of 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, which is the major reason for constructing a shallow cutting, instead of just a "standard" one (that is, with respect to the entire arrangement $\mathcal{A}(\mathcal{R})$).

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 $O(r\varphi(r))$. 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.*

4. NEAR-LINEAR ALGORITHM FOR AXIS-PARALLEL RECTANGLES

We now present a near-linear algorithm for computing a hitting set for a collection of axis-parallel rectangles. As in Section 3, we first describe the algorithm for the discrete model and then for the continuous model.

The discrete model. Let $\mathcal{R} = \{R_1, \ldots, R_n\}$ be a set of n axis-parallel rectangles in $\mathbb{R}^2$, and let X be a set of m points in $\mathbb{R}^2$. We follow the same iterative approach as in Section 3, which works in $\lceil \log_2 n \rceil$ stages, but each stage is implemented differently. 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 $n_i = |\mathcal{R}_i|$, $m_i = |X_i|$. Initially, $\mathcal{R}_1 = \mathcal{R}$, $X_1 = X$, and $H_0 = \emptyset$. The algorithm terminates when $X_i = \emptyset$ or $\mathcal{R}_i = \emptyset$. The ith stage proceeds as follow.

Let $l_i = \max_{p\in X_i} \Delta(p, \mathcal{R}_i)$. By a sweep-line algorithm, l_i can be computed in $O((m_i + n_i)\log n_i)$ time [21]. Next, we sweep $\mathcal{R}_i$ and X_i from left to right by a vertical line L. We maintain the intersection of L with the rectangles of $\mathcal{R}_i$, a set of intervals, in a segment tree. When L reaches a point $p \in X_i$, we compute in $O(\log n_i)$ time, δ_p, the number of rectangles (currently) in $\mathcal{R}_i$ that contain p. If $\delta_p \geq l_i/2$, we add p to $\bar{H}_i$, delete the set G_p of rectangles of $\mathcal{R}_i$ that contain p from $\mathcal{R}_i$, and update the segment tree. 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 $G_i = \cup_{p\in\bar{H}_i} G_p$, $\mathcal{R}_{i+1} = \mathcal{R}_i \setminus G_i$ (the set of remaining 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 G_i. Moreover, $G_p \cap G_q = \emptyset$, for $p \neq q \in \bar{H}_i$, and $|G_p|, |G_q| \geq l_i/2$. Following the same argument as in Section 3, we can conclude that $\bar{H}_i \leq 2\kappa$, and $l_{i+1} \leq l_i/2$.

If $\mathcal{R}_{i+1} = \emptyset$, we return H_i as the hitting set of $(X, \mathcal{R})$. If $\mathcal{R}_{i+1} \neq \emptyset$ and $X_{i+1} = \emptyset$, we conclude that a hitting set does not exist for $(X, \mathcal{R})$ and we stop. Since the ith stage takes $O((m_i + n_i)\log n_i)$ time, we conclude the following:

THEOREM 4.1. *Given a set $\mathcal{R}$ of n axis-parallel rectangles in $\mathbb{R}^2$ and a set $X \subset \mathbb{R}^2$ of m points, a hitting set for $(X, \mathcal{R})$ of size $O(\kappa\log n)$ can be computed in $O((m+n)\log^2 n)$ time, where $\kappa = \kappa(X, R)$.*

The continuous model. The algorithm for the discrete case can be extended to the continuous case as well, yielding an $O(\log n)$ approximation factor within a similar time bound. However, we describe a different algorithm that not only yields an approximation factor of $O(\log\kappa)$ but also extends to higher dimensions.

We construct an interval tree $\mathcal{T}$ over $\mathcal{R}$, as follows. Let $\mathcal{J}$ be the set of the x-projections of the rectangles in $\mathcal{R}$; $\mathcal{J}$ is a set of n intervals in $\mathbb{R}$. Using a greedy algorithm [18], we compute in $O(n\log n)$ time an optimal hitting set $Q \subset \mathbb{R}$ for $\mathcal{J}$. Let L be the set of vertical lines $x = q$, for $q \in Q$. The lines of L intersect all rectangles in $\mathcal{R}$. Since $\mathcal{R}$ has a hitting set of size κ, we have $k := |L| = |Q| \leq \kappa$.

The lines in L partition the plane into $k+1$ "atomic" slabs. We construct a balanced binary tree $\mathcal{T}$ over these slabs. Each node v of $\mathcal{T}$ is associated with a slab σ_v. Specifically, the ith leaf of $\mathcal{T}$ is associated with the ith leftmost atomic slab. If v is an interior node of $\mathcal{T}$ with children w and z, then $\sigma_v = \sigma_w \cup \sigma_z$. Let $\ell_v \in L$ be the common boundary of σ_w and σ_z. A rectangle $R \in \mathcal{R}$ is stored at the highest node v of $\mathcal{T}$ for which ℓ_v intersects R. Since each rectangle in $\mathcal{R}$ intersects at least one line in L, it is associated with a unique interior node of $\mathcal{T}$. Let $\mathcal{R}_v \subseteq \mathcal{R}$ be the set of rectangles stored at v, and put $n_v := |\mathcal{R}_v|$. For $i \leq \log_2(k+1)$, let $V^{(i)}$ be the set of nodes of $\mathcal{T}$ whose level is i. Set $\mathcal{R}^{(i)} = \bigcup_{v\in V^{(i)}} \mathcal{R}_v$. The time to construct these subsets is clearly $O(n\log\kappa)$. See Figure 2 for an illustration.

For a node v, let $\mathcal{J}_v = \{R \cap \ell_v \mid R \in \mathcal{R}_v\}$. Since we can use any point in $\mathbb{R}^2$ to hit the rectangles of $\mathcal{R}_v$, we observe that there exists an optimal hitting set H_v for $\mathcal{R}_v$ in which every point lies on the line ℓ_v. H_v is also a hitting set for $\mathcal{J}_v$, so it suffices to compute a hitting set for $\mathcal{J}_v$. As above, we can construct an optimal hitting set H_v for $\mathcal{J}_v$ (i.e., for $\mathcal{R}_v$) in $O(n_v\log n_v)$ time, using a greedy algorithm [18]; set $\kappa_v = |H_v|$. By construction, for any two nodes v and w at the same level of $\mathcal{T}$, $\mathcal{R}_v$ and $\mathcal{R}_w$ are pairwise disjoint, so their respective hitting sets are also disjoint. Consequently, $\sum_{v\in V^{(i)}} \kappa_v \leq \kappa$. Hence, $H = \bigcup_{v\in\mathcal{T}} H_v$ is a hitting set for $\mathcal{R}$ of size $O(\kappa\log\kappa)$.

This algorithm can be extended to d-rectangles in any dimension d, using induction on d, while paying a $\log\kappa$ factor in the size of the hitting set and a $\log n$ factor in the running time at each inductive step. Omitting the straightforward details, we obtain:

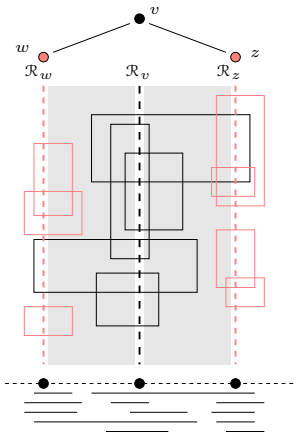

Figure 2. The binary tree construction with two atomic slabs associated with the nodes w, z, whose parent node is v. The lightly-shaded rectangles are stored at the leaf nodes w, z, whereas the black rectangles are stored at v.

THEOREM 4.2. *Given a set $\mathcal{R}$ of n axis-parallel d-rectangles in $\mathbb{R}^d$, a hitting set for $(\mathbb{R}^d, \mathcal{R})$ of size $O(\kappa \log^{d-1} \kappa)$ can be computed in $O(n \log^{d-1} n)$ time, where $\kappa = \kappa(\mathbb{R}^d, \mathcal{R})$.*

5. ITERATIVE REWEIGHTED SAMPLING SCHEME

We now describe a faster implementation of the iterative reweighted scheme of Brönnimann and Goodrich and of Clarkson [12, 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-rectangles in $\mathbb{R}^d$. The size of the hitting set is $O(\kappa \log \kappa)$, and the running time is $O((m+n+\kappa^{d+1})\text{polylog}(mn))$. The size can be improved to $O(\kappa \log\log \kappa)$, for $d = 2, 3$, using the recent result of Aronov *et al.* [10]. For simplicity, we only describe the algorithms for $d = 2$.

We first briefly recall this general technique. We are given a range space $(X, \mathcal{R})$, and 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. The algorithm works in phases, and it maintains a weight function $\omega : X \to \mathbb{Z}$. Initially, $\omega(p) = 1$ for each $p \in X$. 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; note that $\omega(R) \leq \omega(X)/2\kappa$. The algorithm then doubles the weight of each point in $X \cap R$, and repeats the above two steps. The analysis of [12, 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.

Let $\mathcal{W}$ be the set of "witness" ranges identified by the algorithm in 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.

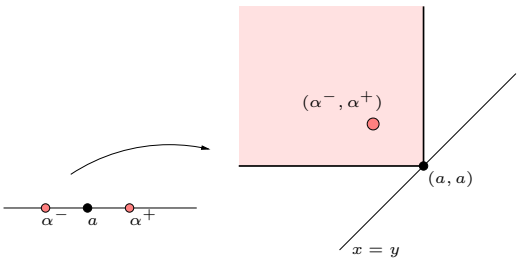

Figure 3. The transformation for the one-dimensional problem. Mapping an interval $[\alpha^-, \alpha^+]$ to the point (α^-, α^+), and a point $a \in \mathbb{R}$ to the quadrant $[-\infty, a] \times [a, \infty]$.

ε-Net Generator. This procedure maintains a subset $\mathcal{W} \subset \mathcal{R}$ 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 $\varepsilon > 0$, return an ε-net N for the weighted range space $(X, \mathcal{R}, \omega)$, i.e., a subset N of X such that, for any range $R \in \mathcal{R}$ with $\omega(R) \geq \varepsilon\omega(X)$, $R \cap N \neq \emptyset$.

If we implement the verifier and the ε-net generator in a brute-force manner, each phase of the algorithm can be implemented in $O((m+n)\kappa \log n)$ 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 $\mathbb{R}^2$, we can preprocess X and $\mathcal{R}$ in $O((m+n)\text{polylog}(mn))$ time so that each phase can be implemented in $O(\kappa^2 \text{polylog}(mn))$ time, which is considerably faster than a naive implementation for small values of κ.

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 $\mathcal{R}$. 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 $O(n \log^3 n)$ time into a 4-dimensional range-searching data structure [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 it also returns a point of $\Pi \cap B$.

A point $p = (a, b) \in \mathbb{R}^2$ intersects 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 3 for a 1D illustration. Let $K_H = \mathbb{R}^2 \setminus \bigcup_{p \in H} O_p$. Hence 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.* [31], we decompose K_H in time $O(h^2 \log^2 h)$ into a family $\mathcal{B}$ of $O(h^2)$ 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 $O(n \log^3 n)$ into a data structure so that we can determine in $O(h^2 \log^2 n)$ time whether a set H of h points is a hitting set of $\mathcal{R}$.*

The ε-net generator. Let $\mathcal{W}$ be the set of "witness" rect-

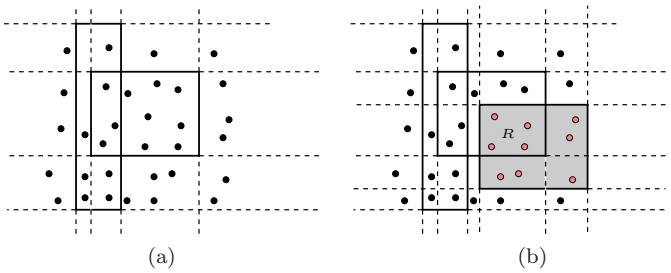

Figure 4. (a) An illustration of the arrangement of the lines passing through the rectangle edges. (b) An insertion of a new rectangle R (depicted in the figure by the lightly-shaded rectangle) into the arrangement. As a result the weight of each point inside R is doubled.

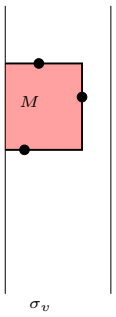

Figure 5. An anchored maximal S-empty rectangle.

angles 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 w_τ be the number of rectangles in W that contain τ. The weight $\omega(p)$ of each point $p \in X_\tau$ is 2^{w_τ}. Set $\omega(\tau) = \omega(X_\tau) = 2^{w_\tau}|X_\tau|$, and $\Omega = \omega(X) = \sum_\tau \omega(\tau)$. For each cell $\tau \in \mathcal{A}(\Gamma)$, we maintain two copies of X_τ, one sorted by the x-coordinates and the other by the y-coordinates, and the quantities w_τ, $\omega(\tau)$. 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, 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 $w_{\tau^-} = w_{\tau^+} = w_\tau$. 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, using standard techniques. (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 w_τ for all those cells that lie inside R and update the values of $\omega(\tau)$ for each cell. Updating the arrangement and the weights of all the cells take $O(s^2)$ time. 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 amortized time spent in inserting a rectangle is $O(s^2)$. See Figure 4.

Given a parameter ε, we compute an ε-net of $(X, \mathcal{R})$ 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. In order to choose a point of X randomly, 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. After $O(s^2)$ preprocessing, a random cell of $\mathcal{A}(\Gamma)$ can be chosen in $O(\log s)$ time; see e.g. [36]. 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. *We can maintain $\mathcal{W}$ and X into a data structure so that after $O(m \log m)$ preprocessing, a rectangle can be inserted into $\mathcal{W}$ in $O(s^2)$ time, and an ε-net of $(W, \mathcal{R}, \omega)$ of size $O(1/\varepsilon \log(1/\varepsilon))$ can be constructed in randomized expected time $O(s^2 + (1/\varepsilon)\log(1/\varepsilon)\log s)$.*

An improved ε-net generator for $d = 2, 3$.

We now describe how we can adapt the recent construction by Aronov *et al.* [10] to construct an ε-net of $(X, \mathcal{R}, \omega)$ of size $O(1/\varepsilon \log\log(1/\varepsilon))$ for $d = 2, 3$. This requires building a more sophisticated data structure on $\mathcal{W}$ and X. In particular, we build 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. Return the leftmost point whose rank is at least w.

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

Each of the first three operations takes $O(s \operatorname{polylog}(ms))$ time, and the last operation takes $O((s+t)\operatorname{polylog}(ms))$ time. Before describing the data structure we describe how the construction by Aronov *et al.* [10] can be implemented quickly. Set $r = (c/\varepsilon)\log\log(1/\varepsilon)$, where $c \geq 1$ is a sufficiently large constant.

I. Choose a random subset $S \subseteq X$, where each point p of X is chosen with probability $\omega(p)/\omega(X)$. The expected size of S is r. Esther: Isn't it $r\omega(p)/\omega(X)$?

II. Divide the plane into a set Σ of r slabs by drawing $r-1$ vertical lines so that the weight of the points lying in 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 most 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 $S_v = S \cap \sigma_v$. We call a rectangle M *anchored* (with respect to v) if one of the edges of M lies on the common boundary line $\sigma_v \cap \sigma_u$, where u is the sibling of 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 $\operatorname{int}(M) \cap S_v = \emptyset$ and there is no anchored rectangle $M' \supset M$ such that $\operatorname{int}(M') \cap S_v = \emptyset$. See Figure 5 for an illustration. Aronov *et al.* [10] showed that $|\mathcal{M}_v| = O(|S_v|)$ and that $\mathcal{M}_v$ can be computed in $O(|S_v| \log |S_v|)$ time. Set $\mathcal{M} = \bigcup_{v \in \mathcal{T}} \mathcal{M}_v$. The expected size of $\mathcal{M}$ is $O(r \log r)$.

IV. For each $M \in \mathcal{M}$, compute $\omega(M \cap X)$. If there exists an integer $t_M \geq c \log\log(1/\varepsilon)$, s.t. $t_M \cdot n/r \leq \omega(M \cap X) \leq (1 + t_M) \cdot n/r$, 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 $S_M = \emptyset$. Esther: Notice that the inequality should be applied with n/r and not with εn.

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

Aronov *et al.* [10] proved that N is an ε-net of $(X, \mathcal{R}, \omega)$ and that the expected size of N is $O(r) = O(1/\varepsilon \log\log(1/\varepsilon))$. As for the running time, step I can be implemented in $O^*(r+s)$ time[3] by calling Random$(r, \mathbb{R}^2)$. Step II can be implemented in $O^*(sr)$ time by calling the procedure Rank$(\cdot)$ $r-1$ times. Finally, we spend $O^*(s)$ time to compute $\omega(X \cap M)$ and another $O^*(s+|S_M|)$ time to construct S_M. Hence, step IV takes

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

expected time.

We now describe the data structure, an extension of the one for the previous ε-generator, for storing $\mathcal{W}$ and X that supports operations (i)–(iv). More presicely, 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. When convenient, we will represent a cell of $\mathcal{A}(\Gamma')$ as $\tau_{i,j}$, for $0 \le i, j \le 2s$. For each cell $\tau \in \mathcal{A}(\Gamma')$, let X_τ, w_τ, and $\omega(\tau)$ be as defined above. In addition, to maintaining these quantities, we also preprocess X_τ into a dynamic range-tree data structure $\psi(\tau)$ so that $\omega(X_\tau \cap R)$, for a query rectangle R, can be computed in $O(\log^2 m)$ time. 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 C_j so that, for an interval $[a, b]$, $\sum_{i=a}^{b} \omega(\tau_{i,j})$ can be computed in $O(\log n)$ time. Using C_j, we can also choose a random cell among $\tau_{a,j}, \ldots, \tau_{b,j}$, where each cell is chosen with probability proportional to its weight.

Esther: If C_j is a column, shouldn't we flip i and j? as i ranges over the x-coord and not the y-coord.

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^+ and $|X_{\tau^-}| \le |X_{\tau^+}|$. We obtain $\psi(\tau^+)$ from $\psi(\tau)$ by deleting the points of X_{τ^-} from $\psi(\tau)$, and we construct $\psi(\tau^-)$ by building the range tree on X_{τ^-}. This step takes $O(|X_{\tau^-}| \log |X_{\tau^-}|)$ time and we charge it to preprocessing—each point of X is charged a total of $O(\log^2 m)$ over the entire sequence of operations. Next, for each column j of the arrangement, we reconstruct the binary tree C_j. The total amortized time spent in the insertion is $O^*(s^2)$.

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., $\tau_{a,b} \cap R \neq \emptyset$ for $a_L \le a \le a_R$ and $b_L \le b \le b_R$. For a cell $\tau \in \mathcal{A}(\Gamma')$ that intersects ∂R, we compute $\omega(X_\tau \cap R)$ using $\psi(\tau)$. If $a_R > a_{L+1}$ and $b_R > b_{L+1}$, i.e., a cell $\tau \subset R$, for each $b_L < b < b_R$, we compute $W_b = \sum_{a_L < a < a_R} \omega(\tau_{a,b})$ using C_b. We then add these quantities to obtain $\omega(X \cap R)$. The time spent is $O^*(s)$.

Random(t,R): We choose a random point of $A \cap R$ in three stages: first choose the column j from which the point is chosen, next choose the cell τ of column j from which the point is chosen, and finally choose a random point of $X_\tau \cap R$. After $O^*(s)$ preprocessing, each point is chosen in $O(\log^2 s)$ time. In more details, let a_L, b_L, a_R, b_R, W_b be the same as in Wt(R). Let $Y = \{\omega(X_\tau \cap R) \mid \tau \cap \partial R \neq \emptyset\} \bigcup \{W_b \mid b_L < b < b_R\}$. We store Y in a tree so that a random item of Y can be chosen in $O(\log s)$ time with probablilty proportional to its value. If we choose W_b, then we choose a random cell $\tau_{a,b}$, $a_L < a < a_R$, with probability $\omega(\tau_{a,b})/W_b$, using C_b. Finally, we choose a random point of $X_{\tau_{a,b}} \cap R$ using $\psi(X_{\tau_{a,b}})$. If an item $\omega(X_\tau \cap R)$ is chosen from Y, we choose a random point of $X_\tau \cap R$ using $\psi(X_\tau)$. 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 $\psi(\tau)$ stores the points of X_τ sorted by their x-coordinates. Hence, using a technique by Fredrickson and Johnson [], we can choose a point of a given rank from the set $X_{\tau_{0,j}}, X_{\tau_{1,j}}, \ldots$, in $O^*(s)$ time. Hence, Rank(w) takes $O^*(s)$ time.

Esther: Below, I removed the polylog$(\cdot)$ factors, and used O^* instead.

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. 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 $O^*((m+n+\kappa^3))$. Both the verifier and the ε-net generator procedures can be extended to any dimension $d \ge 2$ (the improved ε-net generator with respect to the bound of Aronov *et al.* [10] is extended only to $d = 3$), therefore the hitting-set algorithm can be extended to higher dimensions. The expected running time is then $O^*((m+n+\kappa^{d+1}))$. We thus conclude the following:

Theorem 5.3. *Let X be a set of m points in $\mathbb{R}^d$, and let $\mathcal{R}$ be a set of n d-rectangles 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 (presented for any dimension d) to be nearly $O\left(n + \kappa^d\right)$, over all steps k of the algorithm, however, we are not aware of any technique that achieves a similar time bound for the verifier.

Acknowledgments. The authors thank Sariel Har-Peled for the algorithm presented in Section 4 under the discrete case, which improves the approximation factor of the original algorithm to this problem, presented in a preliminary version by the authors.

References

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

[2] P. K. Agarwal, D. Z. Chen, S. K. Ganjugunte, E. Misiołek and M. Sharir Stabbing convex polygons with a segment or a polygon. In *Proc. 16th European Sympos. Algorithms*, 52–63, 2008.

[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

[3] Under this context, a bound of the form $O^*(f(q))$ means that the actual bound is $O(f(q) \cdot \text{polylog}(nm))$.

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. Submitted for publicaiton.

[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: A review. In *Surveys on Discrete and Computational Geometry*, (I. Goodman, J. Pach and R. Pollack, eds.), AMS, Providence, RI, pp. 9–48, 2008.

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

[9] P. K. Agarwal, J. Xie, J. Yang, and H. Yu. Scalable continuous query processing by tracking hotspots. In *Proc. Intl. Conf. Very Large Database Systems*, 31–42, 2006.

[10] B. Aronov, E. Ezra, and M. Sharir, Small-size ε-nets for axis-parallel rectangles and boxes. In *Proc. 41th Annu. ACM Symp. Theory Comput.*, to appear.

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

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

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

[14] C. Chekuri, K. Clarkson, and S. Har-Peled. On the multi-cover problem in geometric settings. In *Proc. 25th Annu. ACM Sympos. Comput. Geom.*, to appear.

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

[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] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001.

[19] M. de Berg. Improved bounds on the union complexity of fat objects. *Discrete Comput. Geom.*, 40(2008):127–140.

[20] M. de Berg, K. Dobrindt, and O. Schwarzkopf. On lazy randomized incremental construction. *Discrete Comput. Geom.*, 14(1995):261–286.

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

[22] 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.

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

[24] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Symp. Theory Comput.*, 434–444, 1988.

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

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

[27] 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.

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

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

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

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

[32] 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.

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

[34] J. Matoušek, R. Seidel, and E. Welzl. How to net a lot with little: Small ε-nets for disks and halfspaces. In *Proc. 6th Annu. Sympos. Comput. Geom.*, 16–22, 1990.

[35] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. *SIAM J. Comput.*, 23(1994):154–169.

[36] R. Motwani and P. Raghavan. *Randomized Algorithms*, Cambridge University Press, New York, 1995.

[37] K. Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.

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

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

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

[41] J. Urrutia. Art Gallery and illumination problems. In *Handbook of Computational Geometry*, (J. Sack and J. Urrutia, eds), Elsevier, Amsterdam, pages 973–1027, 2000.

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