Range Minima Queries with Respect to a Random Permutation, and Approximate Range Counting*

Haim Kaplan† Edgar Ramos‡ Micha Sharir§

September 30, 2007

Abstract

In approximate halfspace range counting, one is given a set P of n points in R^d, and an $\varepsilon > 0$, and the goal is to preprocess P into a data structure which can answer efficiently queries of the form: Given a halfspace h, compute an estimate N such that $(1-\varepsilon)|P \cap h| \leq N \leq (1+\varepsilon)|P \cap h|$.

Several recent papers have addressed this problem, including a study by the authors [18], which is based, as is the present paper, on Cohen's technique for approximate range counting [9]. In this approach, one chooses a small number of random permutations of P, and then constructs, for each permutation π, a data structure that answers efficiently minimum range queries: Given a query halfspace h, find the minimum-rank element (according to π) in $P \cap h$. By repeating this process for all chosen permutations, the approximate count can be obtained, with high probability, using a certain averaging process over the minimum-rank outputs.

In the previous study, the authors have constructed such a data structure in $\mathbb{R}^3$, using a combinatorial result about the overlay of minimization diagrams in a randomized incremental construction of lower envelopes.

In the present work, we propose an alternative approach to the range-minimum problem, based on cuttings, which achieves better performance. Specifically, it uses, for each permutation, $O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$ expected storage and preprocessing time, and answers a range-minimum query in $O(\log n)$ expected time. We also present a different approach, based on so-called "antennas", which is very simple to implement, although the bounds on its expected storage, preprocessing, and query costs are worse by polylogarithmic factors.

*Work by Haim Kaplan was partially supported by Grant 975/06 from the Israel Science Foundation (ISF). Work by Micha Sharir was partially supported by NSF Grant CCR-05-14079 by a grant from the U.S.-Israeli Binational Science Foundation, by grant 155/05 from the Israel Science Fund, Israeli Academy of Sciences, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

†School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: haimk@post.tau.ac.il

‡Department of computer science, University of Illinois at Urbana-Champaign E-mail: edgar@cs.uiuc.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. E-mail: michas@post.tau.ac.il

1 Introduction

Approximate range counting. Let P be a finite set of points in $\mathbb{R}^d$, and $\mathcal{R}$ a set of ranges (certain subsets of $\mathbb{R}^d$, e.g., halfspaces, balls, etc.). The *range counting problem* for $(P, \mathcal{R})$ is to preprocess P into a data structure that supports efficient queries of the form: Given a range $r \in \mathcal{R}$, count the number of points in $r \cap P$. We focus here on the case where $\mathcal{R}$ is the set of halfspaces in $\mathbb{R}^d$. (Our results then apply also to balls in $\mathbb{R}^{d-1}$, using a standard lifting transformation.)

Unfortunately, the best algorithms for solving the *exact* range counting problem are not very efficient. For halfspaces in $\mathbb{R}^d$, if we wish to answer queries in logarithmic or polylogarithmic time, the best solution requires $\Omega(n^d)$ storage, and if we allow only linear or near-linear storage, the best known query time is $O(n^{1-1/d})$ [22]. For example, when $d = 3$ we need $\Omega(n^{2/3})$ time for an exact counting query, with near-linear storage.

It is therefore desirable to find improved algorithms that can answer *approximate range counting* queries, in which we specify the maximum *relative* error $\varepsilon > 0$ that we allow, and, for any range $r \in \mathcal{R}$, we want to quickly estimate $|P \cap r|$, so that the answer n' that we produce satisfies

$$(1 - \varepsilon)|P \cap r| \leq n' \leq (1 + \varepsilon)|P \cap r|. \tag{1}$$

In particular, if $|P \cap r| < \frac{1}{\varepsilon}$, it has to be counted *exactly* by the algorithm. Specializing this still further, the case where $|P \cap r| = 0$ (*range emptiness*) has to be detected exactly by the algorithm.

Using ε-approximations. There is a simple well-known method that almost achieves this goal. That is, choose a random sample E of $\frac{c}{\varepsilon^2} \log \frac{1}{\varepsilon}$ points of P, for some sufficiently large absolute constant c (that depends on the so-called *VC-dimension* of the problem [7]). Then, with high probability, E is an *ε-approximation* for P (see, e.g., [7]), in the sense that, with high probability, we have, for each $r \in \mathcal{R}$,

$$\left| \frac{|E \cap r|}{|E|} - \frac{|P \cap r|}{|P|} \right| \leq \varepsilon.$$

This allows us to approximate $|P \cap r|$ by $|E \cap r| \cdot \frac{|P|}{|E|}$, where $|E \cap r|$ is obtained by brute force, in $O(|E|)$ time. However, the *additive* error bound is $\varepsilon|P|$, rather than $\varepsilon|P \cap r|$. If $|P \cap r|$ is proportional to $|P|$, then an appropriate re-scaling of ε turns this absolute error into the desired relative error. However, if $|P \cap r|$ is small, the corresponding re-scaling of ε, namely, using $\varepsilon' = \varepsilon|P \cap r|/|P|$, will require $|E|$ to grow significantly (roughly by a factor of $O(|P|^2/|P \cap r|^2)$) to ensure relative error of ε, and the approach will become inefficient. In particular, range emptiness cannot be detected exactly by this method, unless we take $E = P$. Some saving in the size of the ε-approximation can be made by using so-called *relative* ε-approximations; see [19, 15]. However, the technique continues to be applicable only to sufficiently large ranges (larger than some pre-specified threshold, which affects the size of the ε-approximation).

Cohen's technique. In this paper we present a different approach to approximate range counting. Our technique is an adaptation of a general method, introduced by Cohen [9], which estimates the number of data objects in a given range r as follows. One assigns to each data object, independently, a random *weight*, drawn from an exponential distribution with density function e^{-x}, and finds the *minimum rank* of an object in the query range r. Then we repeat this experiment $O\left(\frac{1}{\varepsilon^2} \log n\right)$ times, compute the average μ of the *weights* of the minimum elements, and approximate $|P \cap r|$ by $1/\mu$. (Cohen [9] also proposes several other estimators that have similar properties.) As shown in [9], this approximate count lies, with high probability, within relative error ε of $|P \cap r|$. If

only $\frac{1}{\varepsilon^2}$ experiments are conducted, the *expected* relative error remains at most ε. See [9] for more details.

To apply this machinery for approximate halfspace range counting in $\mathbb{R}^d$, for $d \geq 3$, we need to solve the following problem: Let P be a set of n points in $\mathbb{R}^d$, in general position,[1] and let π be a random permutation of P. (It is easily verified that the sorted order of the points of P according to their randomly drawn weights is indeed a random permutation; see [9].) We want to construct a data structure that can answer efficiently *halfspace-minimum range queries* of the form: Given a query halfspace h, find the point of $p \in P \cap h$ of minimum rank in π (i.e., minimum value of $\pi(p)$).

Our results. We present efficient algorithms that perform these minimum-rank range searching tasks. We focus on the dual setup where we are given a set H of n hyperplanes in $\mathbb{R}^d$. Each hyperplane in H draws a random weight from the same exponential distribution, as above, and we let $\pi = (h_1, \ldots, h_n)$ denote the random permutation of the hyperplanes ordered by increasing weight. Given a query point p, we want to find the first hyperplane in π (the one with smallest weight) which passes below[2] p. More generally, let q be the projection of p onto $\mathbb{R}^{d-1}$. We compute the prefix minima of π with respect to the heights of the hyperplanes of H at q. These prefix minima are those hyperplanes h_j whose height at q is smaller than the height at q of every h_i with $i < j$. Putting this differently, assume that we insert the hyperplanes of H one at a time, in the order of increasing rank in π, while maintaining their lower envelope. The sequence of prefix minima at q consists exactly of the hyperplanes that attain the lower envelope at q when they are inserted. Since π is a random permutation, the expected number of prefix minima is $O(\log n)$. The hyperplane with minimum weight below p is the first hyperplane in the sequence of prefix minima that passes below p. If there is no such hyperplane in the sequence of prefix minima then no hyperplane of H passes below p.

We present two algorithms for this range searching problem, which work in any dimension. We obtain our algorithms by carefully adapting known randomized range searching techniques to answer our prefix minima query. The main idea is to make the randomized range searching technique at stake use the given random permutation of the hyperplanes as its source of randomness, rather than tossing additional coins.

Both algorithms are based on *cuttings*. The first algorithm computes a triangulation of the lower envelope of the first r hyperplanes of π. It then continues recursively with the *conflict list* of each simplicial vertical prism Δ below a simplex of the triangulation, where this list consists of all the hyperplanes that cross the prism. The random permutation for the hyperplanes that cross Δ is obtained by restricting π to this set. We answer a prefix minima query by first searching for the prefix minima within the first r hyperplanes of π. We then continue the search recursively within the conflict list of the (unique) prism intersecting the vertical line through the query point q. We first use this construction with a constant value of r, but to obtain the best storage and preprocessing bounds, we bootstrap our construction and use larger values of r that depend on n.

The expected query time of the resulting data structure is $O(\log n)$, for any dimension $d \geq 3$. In $\mathbb{R}^3$ the data structure requires $O(n)$ expected storage, and the expected preprocessing time is $O(n \log n)$. For $d > 3$, our data structure requires $O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$ expected storage and preprocessing time. Note that this is smaller by a polylogarithmic factor than the worst-case combinatorial complexity of the lower envelope of the hyperplanes. Our bootstrapping technique

[1] To simplify the presentation, we assume throughout the paper that the data objects (points or planes) are in general position, in the sense discussed, e.g., in [10].

[2] The relations "above" and "below" are with respect to the x_d-direction. A vertical line is a line parallel to the x_d-direction.

resembles the ray shooting data structure of Matoušek and Schwarzkopf [23].

Our second data structure uses a simple variant of the technique of Mulmuley [25, Chapter 6] and Schwarzkopf [26, Section 4], based on so-called "antennas". Let H_i be the set of the first $n/2^i$ hyperplanes in π. These subsets form a so-called "gradation" $H_0 = H \supseteq H_1 \supseteq H_2 \supseteq \cdots \supseteq H_k$ of H, where $k = \lfloor \log n \rfloor$. For each vertex v of the lower envelope of H_i, we construct the *conflict list* of v with respect to H_{i-1}; this is the list of hyperplanes of H_{i-1} that pass below v. To answer a range-minimum query, we associate with each query point p, and for each i, a structure known as the *antenna* of p, which maps p to a particular set of up to 2^d vertices of H_i. We start with the (easily computable) antenna of p with respect to H_k, and then proceed backwards through the gradation, constructing the antenna of p with respect to H_{i-1} from the antenna with respect to H_i, using the conflict lists of the vertices of this antenna (see below for details). While constructing these antennas, we can easily find the minimum-rank hyperplane below p.

The expected query time of this data structure in dimension d is $O(\log^{d-2} n)$, and its expected storage is $O(n^{\lfloor d/2 \rfloor})$. The expected preprocessing time is $O(n^{\lfloor d/2 \rfloor})$ for $d \geq 4$, and $O(n \log n)$ for $d = 3$. These asymptotic bounds are inferior to the ones of the first data structure, but this data structure is considerably simpler and easier to implement. In fact, it is much simpler than the previous variants considered in [25, 26].

Moreover, the resulting structure can be adapted to answer several other kinds of queries, all with the same resource bounds. First, it can find, as does the first structure, the entire sequence of prefix minima of π at any query point $q \in \mathbb{R}^{d-1}$. Moreover, it can perform point location of, and arbitrary ray shooting from, any query point below the lower envelope of H.

Comparing our antenna-based techniques with the previous ones, we make the following observations:

The data structure of Mulmuley [25] handles point location and ray shooting queries for *any* point in $\mathbb{R}^d$. Consequently, its expected storage and preprocessing are $O(n^d)$. To obtain the significantly smaller costs, stated above, when restricting the structure to queries below the lower envelope (of for range-minimum queries), we had to modify the structure considerably; as a pleasant surprise, this modification has resulted in a much simplified structure.

The data structure of Schwarzkopf [26, Section 4], is designed for ray shooting queries from points below the lower envelope of H. The structure uses data collected during a randomized insertion, and does not use a gradation. As a result, it can can easily be adapted to answer prefix minima queries too. For any d, the structure requires $O(n^{\lfloor d/2 \rfloor}(\log n)^{O(1)})$ expected storage and preprocessing time, and answers a query in $O(\log^2 n)$ expected time. Thus, it does better than our solution in terms of query time, when d is large, but worse in terms of storage and preprocessing. The data structure is fairly complicated, since it builds (in order to complete truncated antennas) a complicated substructure for each face that shows up on the lower envelope during the randomized incremental construction. Our solution uses a different approach, and is much simpler.

Returning to our original goal, we next plug the first algorithm into the general approximate range counting framework of Cohen [9]. In $\mathbb{R}^3$, we obtain an algorithm that uses $O\left(\frac{1}{\varepsilon^2} n \log n\right)$ expected storage and preprocessing time, and answers a query in $O\left(\frac{1}{\varepsilon^2} \log^2 n\right)$ expected time. The count produced by the algorithm is guaranteed to satisfy (1) with high probability (over the choice of the random permutations π). (This should be compared to the $O(n^{2/3})$ cost of exact range counting queries with near-linear storage.) As mentioned before, if we only want the expectation of the output count to be within $1 \pm \varepsilon$ of the true count, then we can improve each of these bounds by a factor of $\log n$. Moreover, a simple modification of the algorithm, which we detail below, brings the expected storage down to $O\left(\frac{1}{\varepsilon^2} n + n \log \log n\right)$, without affecting the bound on the query time. In $\mathbb{R}^d$, for $d \geq 4$, the expected query time remains $O\left(\frac{1}{\varepsilon^2} \log^2 n\right)$, and the expected storage and

preprocessing time is $O\left(\frac{1}{\varepsilon^2} n^{\lfloor d/2 \rfloor} \log^{2-\lfloor d/2 \rfloor} n\right)$.

Related work. Except for the alternative approach that uses ε-approximations, as discussed earlier, there are several recent results that present other alternative solutions to the approximate range counting problem.

The application of Cohen's technique to approximate range counting was first proposed by the authors in [18]. In that paper we constructed a data structure for prefix minima queries in $\mathbb{R}^3$, using a combinatorial result about the *overlay of minimization diagrams* in a randomized incremental construction of lower envelopes of planes. This data structure requires $O(\frac{1}{\varepsilon^2} n \log n)$ expected storage and preprocessing time, and answers a query in $O(\frac{1}{\varepsilon^2} \log^2 n)$ expected time. The combinatorial bound itself on the complexity of the overlay can be extended to higher dimensions; this extension is presented in a companion paper [17].

Even earlier, Aronov and Har-Peled [2, 3] showed how to reduce the approximate range counting problem to *range emptiness*. In the conference version of their paper [2], they obtain a data structure in $\mathbb{R}^3$ that uses $O\left(\frac{1}{\varepsilon^3} n \log^2 n\right)$ storage, and answers a query in $O\left(\frac{1}{\varepsilon^2} \log^2 n \log\left(\frac{1}{\varepsilon} \log n\right)\right)$ time: it performs $O\left(\log\left(\frac{1}{\varepsilon} \log n\right)\right)$ binary search steps, each involving $O\left(\frac{1}{\varepsilon^2} \log n\right)$ range emptiness queries, each of which takes $O(\log n)$ time.[3] After we have pointed out (in [18]) that their result can be improved, they indeed refined their analysis, and in the full version of their paper [3] they obtain a data structure which requires $O(\frac{1}{\varepsilon^2} n \log n)$ storage and preprocessing time, and answers a query in $O(\frac{1}{\varepsilon^2} \log^2 n)$ time. These bounds coincide with those that we have obtained in [18].

The technique of Aronov and Har-Peled [2] is applicable whenever range emptiness can be solved efficiently. Indeed, assume that we have a data structure for range emptiness which requires $S(n)$ storage, can be constructed in $T(n)$ time, and answers a query in $Q(n)$ time. Furthermore, assume that $S(n)$ and $T(n)$ satisfy $S(n/i) = O(S(n)/i^\lambda)$ and $T(n/i) = O(T(n)/i^\lambda)$, for some constant parameter λ and any $i \geq 1$. Aronov and Har-Peled then show how to construct a data structure for approximate range counting, which requires $O((\varepsilon^{\lambda-3} + H_{\lambda-2}(\frac{1}{\varepsilon}))S(n) \log n)$ expected storage, $O((\varepsilon^{\lambda-3} + H_{\lambda-2}(\frac{1}{\varepsilon}))T(n) \log n)$ expected preprocessing time, and answers a query in $O(\frac{1}{\varepsilon^2} Q(n) \log n)$ expected time, where $H_\mu(k) = \sum_{i=1}^{k} 1/i^\mu$.

In a more recent work, Afshani and Chan [1] slightly improve the results which we give here in $\mathbb{R}^3$. They present a data structure of expected linear size, which can be constructed in $O(n \log n)$ expected time, and which answers a query in $O\left(\log \frac{n}{k}\right)$ expected time, where k is the exact count of the points in the query halfspace. (Afshani and Chan do not make the dependence of their constant of proportionality on ε explicit.) In contrast with the previous algorithms, the query procedure is Las Vegas, meaning that it always produces a correct approximate count (one that satisfies (1).) The previous algorithms are Monte Carlo, and may return (with very small probability, though) an answer which does not satisfy (1).

Another recent result is due to Aronov et al. [5, 4]. In this approach, they take the partition-tree data structure of Matoušek [20], which facilitates efficient range emptiness queries, or range reporting queries for *shallow* ranges,[4] and modify the structure by adding to each node of the tree a (relative) ε-approximation subset of the set that it stores. The query range is then fed into the structure, and visits some of its nodes. As long as it is shallow, with respect to the subset stored at the current node, the query proceeds in the standard recursive (and efficient)

[3] All time bounds of Aronov and Har-Peled [2, 3] hold with high probability, and the answer is correct with high probability.

[4] That is, more efficient than the partition-tree structure for general (halfspace or simplex) range counting, which handles *arbitrary* ranges [21].

manner. When the query range is detected not to be shallow, the algorithm counts it approximately, using the ε-approximation stored at the current node. (Recall from the earlier discussion that relative ε-approximations produce good relative error when the ranges are large, and the size of the approximation set is relatively small.) By fine-tuning the relevant parameters, one obtains performance bounds that are comparable with those of the corresponding range-emptiness or range-reporting algorithms, and is significantly faster than those for exact range counting. We note that, in dimension $d \geq 4$, this approach produces a data structure that has near-linear size, and the cost of a query is some fractional power of n. In contrast, in this paper we aim to construct data structures which support logarithmic or polylogarithmic query time, using more storage. As noted in [5, 4], a variant of the approach used there can compute the minimum-rank element in a query halfspace, using resources which are comparable with those used in their general approach. Thus their approach allows one to apply Cohen's machinery with a data structure that uses near-linear storage (unlike the ones used in the present paper).

The remainder of this paper is organized as follows. Section 2 describes our first data structure for prefix minima queries with respect to a random permutation of a set of hyperplanes. Section 3 describes the simpler (albeit slightly less efficient) data structure that uses antennas. Section 4 applies the first data structure for prefix minima queries to obtain efficient algorithms for the approximate halfspace range counting problem.

2 The Prefix Minima of a Random Permutation of Hyperplanes Crossing a Vertical Line

Let H be a set of n hyperplanes in $\mathbb{R}^d$, and let $\pi = (h_1, h_2, \ldots, h_n)$ be a random permutation of H. We want to construct a data structure such that, for a given query point $q \in \mathbb{R}^{d-1}$, we can efficiently report the entire sequence of prefix minima, i.e., the sequence of hyperplanes $(h_{i_1}, h_{i_2}, \ldots)$, such that h_{i_k} is the lowest hyperplane at q among those in the prefix $(h_1, \ldots, h_{i_k})$ of π (the expected size of this subsequence is $O(\log n)$). We call the elements of this sequence the *prefix minima* of π at q. Our goal is to achieve logarithmic query time, making the storage (and preprocessing time) as low as possible, where the final target is to make these parameters $o(n^{\lfloor d/2 \rfloor})$, as done by Matoušek and Schwarzkopf [23] for a related problem, namely, ray shooting into the lower envelope of H. We construct three versions of the structure, all answering a query in $O(\log n)$ expected time, so that each of them uses the previous one as an auxiliary bootstrapping substructure, and consumes less storage in expectation. The expected performance of the structure depends on the fact that π is a random permutation. In fact, the only source of randomness in the algorithm is π itself, and expectation is with respect to the choice of π. We assume that $d \geq 4$ in Sections 2.1–2.3, and discuss the case $d = 3$ in Section 2.4.

2.1 The first simple structure

Our data structure is recursive. Each recursive subproblem is specified by a subset H_0 of H, and the sub-permutation π_0 obtained by restricting π to H_0.

We fix some sufficiently large integer constant r. For each subproblem, with input set H_0, we take the set ${H_0}^r$ of the first r hyperplanes in π_0, and construct their lower envelope $LE({H_0}^r)$, whose complexity, by the upper bound theorem [24], is $O(r^{\lfloor d/2 \rfloor})$. We decompose $LE({H_0}^r)$ into $O(r^{\lfloor d/2 \rfloor})$ simplices, using the bottom-vertex triangulation method. Each simplex is extended vertically downwards, yielding a decomposition of the region $\overline{LE}({H_0}^r)$ below $LE({H_0}^r)$ into $O(r^{\lfloor d/2 \rfloor})$ vertical simplicial prisms. For each prism Δ in this decomposition, we compute the set of hyperplanes of H_0

that cross Δ, in a brute force manner. We call this set the *conflict list* of Δ, and denote it by $C(\Delta)$. Then, for each prism Δ, we recursively solve the problem defined by $C(\Delta)$ and the restriction of π to $C(\Delta)$. We keep recursing in this manner until we get to a subproblem where $|C(\Delta)| < r$. In this case we simply store $C(\Delta)$ and stop.

A query with a point $q \in \mathbb{R}^{d-1}$ is answered as follows. We first scan H^r in the π-order, and report, in $O(r)$ time, the prefix minima at q that belong to this prefix. Then we find the prism Δ in the partition of $\overline{LE}(H^r)$ that is stabbed by the vertical line through q, using a brute-force search that takes $O(r^{\lfloor d/2 \rfloor})$ time, and recursively continue the search at the substructure corresponding to $C(\Delta)$ (if it exists), appending the output from the recursive call to that obtained for H^r.

Let $Q(n)$ (resp., $S(n)$, $T(n)$) denote the maximum expected query time (resp., storage, preprocessing time), where the maximum is taken over all problem instances involving n hyperplanes, and expectation is with respect to the choice of π. To bound these parameters, we first argue that, for any prism Δ that arises during the construction, at any recursive level, the permutation π, restricted to $C(\Delta)$, is a random permutation of $C(\Delta)$. For simplicity, we give the argument for the first-level prisms, but it easily extends to any level. Indeed, conditioned on a fixed choice of the set H^r of the first r elements of π, the suffix of π beyond its first r elements is a random permutation of $H \setminus H^r$. Moreover, our decomposition of $\overline{LE}(H^r)$ is uniquely determined once H^r is fixed. Thus the set of its prisms, and the conflict list $C(\Delta)$ of each prism Δ, are uniquely determined and are independent of the choice of the remainder of π. Since the suffix of π is a random permutation of $H \setminus H^r$, its restriction to the fixed set $C(\Delta)$ is also a random permutation of that set.

The maximum expected query time $Q(n)$ satisfies the recurrence

$$Q(n) \leq \mathbf{E}\{Q(n_\Delta)\} + O(r^{\lfloor d/2 \rfloor}),$$

where n_Δ is a random variable, equal to the size of the conflict list $C(\Delta)$ of the simplicial prism $\Delta \in \overline{LE}(H^r)$ intersecting the vertical line through any (fixed) query point q. We then use the fact that, for any fixed $q \in \mathbb{R}^{d-1}$, the expected value of n_Δ is $O(n/r)$. This is a special instance of the general theory of Clarkson and Shor [8], which we review and elaborate upon below. This implies that $Q(n) = O(\log n)$. Indeed, using induction, and the fact that $\log x$ is a concave function, Jensen's inequality implies that, for an appropriate constant c, we have

$$Q(n) \leq \mathbf{E}\{c \log n_\Delta\} + O(r^{\lfloor d/2 \rfloor}) \leq c \log(\mathbf{E}\{n_\Delta\}) + O(r^{\lfloor d/2 \rfloor}) \leq c \log(O(n/r)) + O(r^{\lfloor d/2 \rfloor}) \leq c \log n,$$

with an appropriate choice of c, as asserted.

To bound the expected storage $S(n)$, we go into the Clarkson-Shor theory [8], and, for the sake of completeness, derive and present an adapted version of it, tailored to our setup. In doing so, we also extend the analysis, to obtain related bounds that will be used in the second and third versions of the structure.

The Clarkson-Shor sampling theory. Let F be a finite set of n hyperplanes in $\mathbb{R}^d$. Let Δ be a simplicial prism *defined* by a subset $D \subseteq F$. That is, D is the unique minimal subset such that Δ appears in the triangulation of $\overline{LE}(D)$. Let $b(\Delta)$ denote the size $|D|$ of this defining set. It is easy to verify that $b(\Delta) \leq d(d+3)/2$, for any prism Δ of the above kind. Indeed, to determine the ceiling simplex of Δ in the bottom vertex triangulation of $LE(H^r)$, we have to specify d hyperplanes that define the bottom vertex, the hyperplane that contains the $(d-1)$-dimensional simplex obtained by removing the bottom vertex, and, recursively, the hyperplanes defining this $(d-1)$-dimensional simplex. Therefore, if we denote by $b(d)$ the maximum number of hyperplanes needed to define Δ in $\mathbb{R}^d$, then $b(2) = 3$, and, for $d > 1$, $b(d) = d + 1 + b(d-1)$. Hence $b(d) = (d-1)(d+4)/2$. We abbreviate $b(d)$ to b below.

The *level* of Δ with respect to F, denoted by $\ell(\Delta)$, is the number of hyperplanes in $F \setminus D$ that intersect Δ. We define $\mathbf{\Delta}(F)$ to be the set of all simplicial prisms defined by subsets of F. We also define $\mathbf{\Delta}^c(F)$, for any integer $c \geq 0$, to be the set of prisms $\Delta \in \mathbf{\Delta}(F)$ such that $\ell(\Delta) = c$.

The following simple lemma is taken from Clarkson and Shor [8].

Lemma 2.1 ([8, Lemma 2.1])**.** *Let $R \subseteq F$ be a random subset of r hyperplanes. Then, for each $c \geq 0$, we have*

$$\sum_{\Delta \in \mathbf{\Delta}(F)} \frac{\binom{\ell(\Delta)}{c}\binom{n-b(\Delta)-\ell(\Delta)}{r-b(\Delta)-c}}{\binom{n}{r}} = \mathbf{E}\left\{|\mathbf{\Delta}^c(R)|\right\} \ .$$

Proof. For each $\Delta \in \mathbf{\Delta}(F)$, $\binom{\ell(\Delta)}{c}\binom{n-b(\Delta)-\ell(\Delta)}{r-b(\Delta)-c}/\binom{n}{r}$ is the probability that $\Delta \in \mathbf{\Delta}^c(R)$; we have to choose the $b(\Delta)$ hyperplanes defining Δ, and select exactly c of the $\ell(\Delta)$ hyperplanes crossing Δ. The lemma is now immediate from the definition of expectation. □

The following lemma is needed for the second version of the data structure, presented in Section 2.2.

Lemma 2.2. *Fix a parameter $0 \leq c \leq \frac{4b}{3}\log n$, and let $R \subseteq F$ be a random subset of $3(b+c) \leq r \leq n/(4b\log n)$ hyperplanes. Then, for each $s \geq 0$, we have, assuming n to be at least some sufficiently large constant,*

$$\sum_{\Delta \in \mathbf{\Delta}(F)} \frac{\binom{\ell(\Delta)}{c}\binom{n-b(\Delta)-\ell(\Delta)}{r-b(\Delta)-c}\log^s(\ell(\Delta)+1)}{\binom{n}{r}} = O\left(\log^s \frac{n}{r} \cdot \mathbf{E}\left\{|\mathbf{\Delta}^c(R)|\right\}\right) \ ,$$

where the constant of proportionality depends on d and s.

Proof. We may assume that $r \geq \sqrt{2n}$. For smaller values of r, $\log \frac{n}{r} = \Theta(\log n)$, and the lemma is then an immediate consequence of Lemma 2.1.) We show that the contribution of those Δ with $\ell(\Delta) \geq 2(n/r)^2$ to the sum in the lemma is negligible, and then use Lemma 2.1 to handle the remaining Δ's.

Let $n_{\ell,\beta}$ be the number of prisms Δ such that $\ell(\Delta) = \ell$ and $\beta(\Delta) = \beta$. Put

$$p_{\ell,\beta} = \frac{\binom{\ell}{c}\binom{n-\beta-\ell}{r-\beta-c}}{\binom{n}{r}} \ .$$

Then

$$\sum_{\Delta \in \mathbf{\Delta}(F)} \frac{\binom{\ell(\Delta)}{c}\binom{n-b(\Delta)-\ell(\Delta)}{r-b(\Delta)-c}}{\binom{n}{r}} \log^s(\ell(\Delta)+1) = \sum_{\ell}\sum_{\beta \leq b} n_{\ell,\beta} p_{\ell,\beta} \log^s(\ell+1) \ .$$

Now observe that, for $\ell \geq c$,

$$\frac{p_{\ell,\beta}}{p_{\ell-1,\beta}} = \frac{\ell}{\ell-c} \cdot \frac{n-\ell-r+c+1}{n-\ell-\beta+1} = \frac{1-y}{1-x},$$

where $x = c/\ell$ and $y = (r-\beta-c)/(n-\ell-\beta+1)$; note that $0 \leq x, y \leq 1$ (the inequality $y \leq 1$ holds only when $\ell \leq n+c-r-1$; larger values of ℓ can however be ignored, because then the corresponding terms in the sum vanish).

We also note that $x \le \frac{y}{2}$ for $\ell \ge (n/r)^2$ and $3(b+c) \le r \le n/(4b\log n)$. Indeed,

$$\frac{c}{\ell} \le \frac{r-\beta-c}{2(n-\ell-\beta+1)} \quad \text{holds when} \quad \ell \ge \frac{2c(n-\beta+1)}{r+c-\beta},$$

and one can easily verify that the latter inequality does hold, for $\ell \ge (n/r)^2$ and r in the assumed range (in fact $3(b+c) \le r \le n/(3c)$ would already imply the inequality, and the right inequality $r \le n/(3c)$ is implied by our assumed upper bound on r, as long as $c \le \frac{4b}{3}\log n$). We now use the inequality $\frac{1-y}{1-x} \le 1 - \frac{y}{2}$, which holds for $0 < x \le y/2$ and $y \le 1$, and obtain that

$$\frac{p_{\ell,\beta}}{p_{\ell-1,\beta}} \le 1 - \frac{r-\beta-c}{2(n-\ell-\beta+1)} \ .$$

Moreover, since $r \ge 3(b+c)$ and $1 \le \beta \le b$, one also has

$$\frac{r-\beta-c}{2(n-\ell-\beta+1)} \ge \frac{r}{3n} \ ,$$

so we have

$$\frac{p_{\ell,\beta}}{p_{\ell-1,\beta}} \le 1 - \frac{r}{3n} \ ,$$

for $\ell \ge (n/r)^2$ (and $\ell \le n+c-r+1$). Since $p_{\ell,\beta} \le 1$, for any ℓ, β, we have, for $\ell \ge 2(n/r)^2$,

$$p_{\ell,\beta} \le p_{(n/r)^2,\beta}\left(1 - \frac{r}{3n}\right)^{(n/r)^2} \le e^{-n/(3r)}.$$

Hence, the contribution of those Δ's with $\ell(\Delta) \ge 2(n/r)^2$ is at most

$$\sum_{\ell \ge 2(n/r)^2} \sum_{\beta \le b} n_{\ell,\beta} p_{\ell,\beta} \log^s(\ell+1) \le e^{-n/(3r)} \log^s n \cdot \sum_{\beta \le b} n_{\ell,\beta} = O\left(n^b e^{-n/(3r)} \log^s n\right).$$

Since $r \le n/(4b\log n)$, this bound tends to zero as n grows. For the remaining Δ's, we have $\log(\ell(\Delta)+1) \le 2\log\frac{n}{r} + \log 2$. This, combined with Lemma 2.1, completes the proof. □

The preceding lemmas imply the following theorem, which is a special case of Theorem 3.6 of [8], tailored to our needs.

Theorem 2.3. *Let $R \subseteq F$ be a random subset of r hyperplanes. Then, for any $t \ge 1$, $\gamma = 0, 1$, and $3(b + \lceil t \rceil) \le r \le n/(4b\log n)$, we have*

$$\mathbf{E}\left\{\sum_{\Delta \in \mathbf{\Delta}^0(R)} \ell(\Delta)^t \log^\gamma(\ell(\Delta)+1)\right\} \le A r^{\lfloor d/2 \rfloor}\left(\frac{n}{r}\right)^t \log^\gamma \frac{n}{r},$$

where A is a constant that depends on d, t, and γ. We need the upper bound on r only if $\gamma = 1$; for $\gamma = 0$ this inequality holds for any $3(b + \lceil t \rceil) \le r \le n$.

Proof. We only give the proof for the slightly more complicated case $\gamma = 1$; the proof for $\gamma = 0$ is similar, and simpler. Let

$$Z = \mathbf{E}\left\{\sum_{\Delta \in \mathbf{\Delta}^0(R)} \ell(\Delta)^t \log(\ell(\Delta)+1)\right\} = \sum_{\Delta \in \mathbf{\Delta}(F)} \Pr\left\{\Delta \in \mathbf{\Delta}^0(R)\right\} \ell(\Delta)^t \log(\ell(\Delta)+1). \quad (2)$$

Put $c = \lceil t \rceil$, $\alpha = t/c \leq 1$, $s = 1/\alpha$, and $W(x) = x^\alpha$.

There exist constants $a, e > 0$, which depend on t, such that

$$\ell(\Delta)^t \log(\ell(\Delta)+1) \leq a \binom{\ell(\Delta)}{c}^\alpha \log(\ell(\Delta)+1) + e,$$

for any $\ell(\Delta) \geq 0$. We can therefore rewrite Equation (2) as

$$\begin{aligned} Z &\leq a \sum_{\Delta \in \mathbf{\Delta}(F)} \Pr\left\{\Delta \in \mathbf{\Delta}^0(R)\right\} W\left(\binom{\ell(\Delta)}{c} \log^s(\ell(\Delta)+1)\right) + e \sum_{\Delta \in \mathbf{\Delta}(F)} \Pr\left\{\Delta \in \mathbf{\Delta}^0(R)\right\} \\ &= \left[a \sum_{\Delta \in \mathbf{\Delta}(F)} \frac{\Pr\left\{\Delta \in \mathbf{\Delta}^0(R)\right\}}{\mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\}} W\left(\binom{\ell(\Delta)}{c} \log^s(\ell(\Delta)+1)\right) + e\right] \cdot \mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\} . \end{aligned}$$

Since $W(\cdot)$ is concave, and $\sum_{\Delta \in \mathbf{\Delta}(F)} \frac{\Pr\left\{\Delta \in \mathbf{\Delta}^0(R)\right\}}{\mathbf{E}\{|\mathbf{\Delta}^0(R)|\}} = 1$, it follows that

$$Z \leq \left[aW\left(\sum_{\Delta \in \mathbf{\Delta}(F)} \frac{\Pr\left\{\Delta \in \mathbf{\Delta}^0(R)\right\}}{\mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\}} \binom{\ell(\Delta)}{c} \log^s(\ell(\Delta)+1)\right) + e\right] \cdot \mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\} .$$

Substituting $\Pr\left\{\Delta \in \mathbf{\Delta}^0(R)\right\} = \frac{\binom{n-b(\Delta)-\ell(\Delta)}{r-b(\Delta)}}{\binom{n}{r}}$, we obtain

$$Z \leq \left[aW\left(\sum_{\Delta \in \mathbf{\Delta}(F)} \frac{\binom{n-b(\Delta)-\ell(\Delta)}{r-b(\Delta)}}{\binom{n}{r}\mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\}} \binom{\ell(\Delta)}{c} \log^s(\ell(\Delta)+1)\right) + e\right] \cdot \mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\} . \tag{3}$$

Since $b(\Delta) \leq b$, we have, as is easily verified,

$$\binom{n-b(\Delta)-\ell(\Delta)}{r-b(\Delta)} \leq \frac{(n-r+c)(n-r+c-1)\cdots(n-r+1)}{(r-b)(r-b-1)\cdots(r-b-c+1)} \binom{n-b(\Delta)-\ell(\Delta)}{r-b(\Delta)-c},$$

and therefore, substituting into Equation (3), we get

$$\begin{aligned} Z \leq & \left[aW\left(\frac{(n-r+c)(n-r+c-1)\cdots(n-r+1)}{(r-b)(r-b-1)\cdots(r-b-c+1)} \sum_{\Delta \in \mathbf{\Delta}(F)} \frac{\binom{n-b(\Delta)-\ell(\Delta)}{r-b(\Delta)-c}}{\binom{n}{r}\mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\}} \binom{\ell(\Delta)}{c} \log^s(\ell(\Delta)+1)\right)\right. \\ & \left. + e\right] \cdot \mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\} . \end{aligned}$$

Since we assume that $3(b + \lceil t \rceil) \leq r \leq n/(4b \log n)$, Lemma 2.2 implies that

$$Z \leq \left[aW\left(f\left(\frac{n-r+1}{r-b-c+1}\right)^c \log^s \frac{n}{r} \cdot \frac{\mathbf{E}\left\{|\mathbf{\Delta}^c(R)|\right\}}{\mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\}}\right) + e\right] \cdot \mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\} ,$$

where f is the constant of proportionality in the bound of Lemma 2.2. Note that $\frac{n-r+1}{r-b-c+1} \leq \frac{2n}{r}$, for $r \geq 2(b+c-1)$. Hence we get, for $r \geq 3(c+b)$,

$$Z \leq A'\left[\left(\frac{n}{r}\right)^t \log \frac{n}{r} \cdot \left(\frac{\mathbf{E}\left\{|\mathbf{\Delta}^c(R)|\right\}}{\mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\}}\right)^\alpha + 1\right] \cdot \mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\} =$$

$$A'\left[\left(\frac{n}{r}\right)^t \log \frac{n}{r} \cdot \mathbf{E}\left\{|\mathbf{\Delta}^c(R)|\right\}^\alpha \mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\}^{1-\alpha} + \mathbf{E}\left\{|\mathbf{\Delta}^0(R)|\right\}\right], \tag{4}$$

where A' is some constant that depends on d, and t. Recall that $|\mathbf{\Delta}^0(R)| = O\left(r^{\lfloor d/2 \rfloor}\right)$. A standard application of another basic result of Clarkson and Shor [8, Corollary 3.3] implies that $|\mathbf{\Delta}^c(R)|$ is upper bounded by $O\left(c^{\lceil d/2 \rceil} r^{\lfloor d/2 \rfloor}\right)$. This is easily seen to complete the proof of the theorem (the second term in the final bound is dominated by the first when $r \ll n$). □

Remark: The proof of Theorem 2.3, up to the bound (4), and the proof of Lemmas 2.1 and 2.2, continue to hold if we modify the definition of $\mathbf{\Delta}(F)$ to be the set of all prisms that are intersected by the vertical line through some fixed point $q \in \mathbb{R}^{d-1}$. In this case, $\mathbf{\Delta}^0(R)$ consists of exactly one prism, and $\mathbf{\Delta}^c(R)$ is a constant depending on c. Specializing the analysis to $t = 1$, and $\gamma = 0$, this implies that the expected size $\ell(\Delta)$ of the conflict list $C(\Delta)$ of the prism $\Delta \in \mathbf{\Delta}^0(R)$ that intersects the vertical line through any given query point $q \in \mathbb{R}^{d-1}$ is $O(n/r)$. This is the property that we have cited above, in the analysis of the expected cost of a query.

Analysis: Storage and preprocessing. The expected storage required by the algorithm satisfies the recurrence

$$S(n) \leq \begin{cases} \mathbf{E}\left\{\sum_{\Delta \in \mathbf{\Delta}^0(H^r)} S(\ell(\Delta))\right\} + O(r^{\lfloor d/2 \rfloor}), & \text{for } n \geq r \\ O(r), & \text{for } n < r, \end{cases} \tag{5}$$

where the expectation is over the choice of the random permutation π. As we have argued, this is the same as first taking the expectation over a draw of a random prefix H^r of size r, and then taking the conditional expectation (conditioned on the choice of H^r) within each prism Δ (the prisms are now well defined, once H^r is fixed). This justifies (5).

We prove, by induction on n, that, for any $\delta > 0$ there is a constant $B = B(\delta)$, such that $S(n) \leq Bn^{\lfloor d/2 \rfloor + \delta}$. Indeed, this will be the case for $n \leq r$ if we choose B sufficiently large, as a function of r (and we will shortly choose r as a function of δ). For larger values of n, applying the induction hypothesis to Equation (5), we obtain

$$S(n) \leq B \cdot \mathbf{E}\left\{\sum_{\Delta \in \mathbf{\Delta}^0(H^r)} \ell(\Delta)^{\lfloor d/2 \rfloor + \delta}\right\} + O(r^{\lfloor d/2 \rfloor}) \ .$$

Using Theorem 2.3, with $t = \lfloor d/2 \rfloor + \delta$ and $\gamma = 0$, we obtain

$$S(n) \leq BA \cdot \frac{n^{\lfloor d/2 \rfloor + \delta}}{r^{\delta}} + O(r^{\lfloor d/2 \rfloor}) \ .$$

This establishes the induction step, provided that r is chosen to be a sufficiently large constant (as a function of d and δ), and B is also chosen to be sufficiently large.

The maximum expected preprocessing time $T(n)$ satisfies the recurrence

$$T(n) \leq \begin{cases} \mathbf{E}\left\{\sum_{\Delta \in \mathbf{\Delta}^0(H^r)} T(\ell(\Delta))\right\} + O(nr^{\lfloor d/2 \rfloor}), & \text{for } n \geq r, \\ O(r), & \text{for } n < r, \end{cases}$$

where the overhead non-recursive cost is dominated by the cost of constructing the conflict lists $C(\Delta)$, in brute force. Again, the solution of this recurrence is easily seen to be $T(n) = O(n^{\lfloor d/2 \rfloor + \delta})$, for any $\delta > 0$.

In summary, we have shown:

Theorem 2.4. *The data structure described in this subsection answers a prefix-minima query in $O(\log n)$ expected time, and requires $O(n^{\lfloor d/2 \rfloor + \delta})$ expected storage and preprocessing time, for any $\delta > 0$. The expectation is with respect to the random choice of the input permutation.*

2.2 Second structure

Our second structure is similar to the first one, except that at each recursive instance, involving a subset H_0 of H, it uses a large, non-constant value of r. Specifically, we fix some positive $\beta < 1$ (which will be very close to 1; its exact value will be determined later), and set $r = |H_0|^\beta$.

Similar to the first structure, we take, for each subproblem, the set ${H_0}^r$ of the first r hyperplanes in π_0, construct their lower envelope $LE({H_0}^r)$, whose complexity is $O(r^{\lfloor d/2 \rfloor})$, and decompose it into $O(r^{\lfloor d/2 \rfloor})$ simplices, using the bottom-vertex triangulation method. However, it is now too expensive to locate by brute force the prism containing a query point q, since there are too many prisms. We therefore augment the structure by the ray-shooting data structure of Matoušek and Schwarzkopf [23], which can find the hyperplane in $LE({H_0}^r)$ which is first hit by a query ray originating below $LE({H_0}^r)$.

We use this data structure to find the prism Δ containing q as follows. We assume that each face F, of any dimension $j \geq 3$ of $LE({H_0}^r)$, stores its bottom vertex (i.e., the one with the smallest x_d-coordinate; it is unique if one assumes general position), and denote it by $bot(F)$. We first find the hyperplane h^1 containing the simplex defining Δ, by a ray-shooting query with a vertical ray emanating from q. Let q^1 denote the intersection of this vertical ray with h^1, and let $F(h^1)$ denote the facet which is contained in h^1 and appears on $LE({H_0}^r)$. We next perform a second ray-shooting query, with a ray emanating from $bot(F(h^1))$ and passing through q^1. Let h^2 be the hyperplane, other than h^1, which is first hit by this ray, let q^2 be the intersection of this ray with h^2, and let $F(h^1, h^2)$ be the $(d-2)$-face which is contained in $h^1 \cap h^2$ and appears on $LE({H_0}^r)$. Our next query is with a ray emanating from $bot(F(h^1, h^2))$ and passing through q^2. We continue in the same way until our query ray is contained in an edge e of $LE({H_0}^r)$. The sequence of the bottom vertices that we have collected in these queries, including the endpoints of e, define Δ.

This data structure requires $o(r^{\lfloor d/2 \rfloor})$ storage and preprocessing time, and answers a query in $O(\log r)$ time (this is the cost of a single ray-shooting query in [23], and we perform $d-1$ of them). In addition, we build the simple data structure of Section 2.1 for ${H_0}^r$ (again, this set is too large to scan explicitly by brute force when processing a query).

To continue the recursion, we have to compute $C(\Delta)$ for each Δ in the decomposition of $\overline{LE}({H_0}^r)$. A simple way of implementing this step is to take the set V of all the vertices of the prisms Δ (that is, the set of all vertices of $LE({H_0}^r)$), and to preprocess it for halfspace range reporting, as in Matoušek [20]. Then, for each $h \in H_0$, we find the set V_h of vertices that lie above h, and for each vertex $v \in V_h$, we add h to the conflict list of all prisms that are incident to v. It is easy to verify that this correctly constructs the conflict lists (in other words, this reflects the trivial property that any hyperplane that crosses a vertical prism Δ must pass below at least one of its vertices). Using the algorithm in [20, Theorem 1.1], and putting $m = |V| = O(r^{\lfloor d/2 \rfloor})$, this can be done, in dimension $d \geq 4$, with $O(m \log m)$ preprocessing time and $O(m \log\log m)$ storage, so that a query with a halfspace h takes $O(m^{1-1/\lfloor d/2 \rfloor} \log^c m + k_h)$ time, where k_h is the number of points of V above h, and c is a constant that depends on d. Thus, substituting the bound $m = O(r^{\lfloor d/2 \rfloor})$, the overall cost of preprocessing and of executing $|H_0|$ queries is

$$O\left(m \log m + |H_0| m^{1-1/\lfloor d/2 \rfloor} \log^c m + \sum_{h \in H_0} k_h\right) = O\left(r^{\lfloor d/2 \rfloor} \log r + |H_0| r^{\lfloor d/2 \rfloor - 1} \log^c r + \sum_{\Delta} |C_\Delta|\right).$$

Note that in principle $\sum_\Delta |C_\Delta| = \sum_\Delta \ell(\Delta)$ may be significantly larger than $\sum_{h \in H} k_h$, because the number of prisms incident to a vertex $v \in V$ can be large. Nevertheless, by Theorem 2.3 with $t = 1$, $\gamma = 0$, it follows that the expected sum of the sizes of the conflict lists is

$$O\left(r^{\lfloor d/2 \rfloor} \cdot \frac{|H_0|}{r}\right) = O\left(|H_0| r^{\lfloor d/2 \rfloor - 1}\right).$$

Hence the overall cost of this step is

$$O\left(|H_0| r^{\lfloor d/2 \rfloor - 1} \log^c r\right).$$

(By the choice of r, the first term $O(r^{\lfloor d/2 \rfloor} \log r)$ is dominated by the second term, provided that $|H_0|$ is at least some sufficiently large constant.)

The storage required during preprocessing is $O(r^{\lfloor d/2 \rfloor} \log\log r)$. The overall size of the nonrecursive part of the data structure is thus $O(r^{\lfloor d/2 \rfloor + \delta})$, for any $\delta > 0$, as implied by Theorem 2.4.

Once we have computed $C(\Delta)$ for all Δ in the decomposition of $\overline{LE}(H_0{}^r)$, we continue to preprocess each subproblem recursively, as in the first structure. The recursion stops when $|H_0|$ is smaller than some fixed constant n_0, in which case we just store the list $C(\Delta)$ and stop.

We answer a query with a point $q \in \mathbb{R}^{d-1}$ as follows. We first query the simple data structure constructed for H^r, and find the prefix minima at q that belong to H^r; this takes $O(\log r) = O(\log n)$ time. Then we use the data structure of Matoušek and Schwarzkopf [23] to find the prism Δ in the partition of $\overline{LE}(H^r)$ that is stabbed by the vertical line through q, in additional $O(\log r)$ time. We then continue with the recursive subproblem corresponding to $C(\Delta)$, and append its output to that obtained for H^r.

The maximum expected query time $Q(n)$ satisfies the recurrence

$$Q(n) \leq \mathbf{E}\{Q(n_\Delta)\} + O(\log n),$$

where, as above, n_Δ is a random variable, equal to the size of the conflict list $C(\Delta)$ of the simplicial prism $\Delta \in \overline{LE}(H^r)$ intersecting the vertical line through any (fixed) query point q. We claim that $Q(n) = O(\log n)$, and prove it by induction, using similar arguments to those in the analysis of the first structure. That is, we get the inequality

$$Q(n) \leq \mathbf{E}\{c \log n_\Delta\} + O(\log n) \leq c \log(\mathbf{E}\{n_\Delta\}) + O(\log n)$$

$$\leq c \log(O(n/r)) + O(\log n) \leq c \log(O(n^{1-\beta})) + O(\log n) \leq c \log n,$$

with an appropriate choice of c, as asserted.

The expected storage required by the algorithm satisfies the recurrence

$$S(n) \leq \begin{cases} \mathbf{E}\left\{\sum_{\Delta \in \mathbf{\Delta}^0(H^r)} S(\ell(\Delta))\right\} + O(r^{\lfloor d/2 \rfloor + \delta}), & \text{for } n \geq n_0, \\ O(1), & \text{for } n < n_0. \end{cases} \tag{6}$$

We prove, using induction on n, that $S(n) \leq B n^{\lfloor d/2 \rfloor} \log(n+1)$, for an appropriate sufficiently large constant B that depends on d. Indeed, this holds for $n \leq n_0$, provided that B is sufficiently large. For larger values of n, we substitute the induction hypothesis into Equation (6), and obtain

$$S(n) \leq B\mathbf{E}\left\{\sum_{\Delta \in \mathbf{\Delta}^0(H^r)} \ell(\Delta)^{\lfloor d/2 \rfloor} \log(\ell(\Delta)+1)\right\} + O(r^{\lfloor d/2 \rfloor + \delta}).$$

We now apply Theorem 2.3, with $t = \lfloor d/2 \rfloor$ and $\gamma = 1$, to conclude that

$$\mathbf{E}\left\{\sum_{\Delta \in \mathbf{\Delta}^0(R)} \ell(\Delta)^{\lfloor d/2 \rfloor} \log(\ell(\Delta)+1)\right\} \leq A n^{\lfloor d/2 \rfloor} \log \frac{n}{r},$$

where A is a constant that depends on d. Recalling that $r = n^\beta$, we obtain

$$S(n) \le BAn^{\lfloor d/2 \rfloor} \log \frac{n}{r} + O(r^{\lfloor d/2 \rfloor + \delta}) \le B(1-\beta)An^{\lfloor d/2 \rfloor} \log n + Cn^{\beta(\lfloor d/2 \rfloor + \delta)} ,$$

for an appropriate constant C that depends on d and on δ. We now choose β so that $(1-\beta)A \le 1/2$, then choose δ so that $\beta(\lfloor d/2 \rfloor + \delta) \le \lfloor d/2 \rfloor$, and finally choose B to satisfy $B \ge 2C$. With these choices, the induction step carries through, and completes the proof of the bound for $S(n)$.

The maximum expected preprocessing time $T(n)$ satisfies the similar recurrence

$$T(n) \le \begin{cases} \mathbf{E}\left\{\sum_{\Delta \in \mathbf{\Delta}^0(H^r)} T(\ell(\Delta))\right\} + o(n^{\lfloor d/2 \rfloor}), & \text{for } n \ge n_0, \\ O(1), & \text{for } n < n_0. \end{cases}$$

where the overhead non-recursive cost is dominated by the cost of constructing the conflict lists $C(\Delta)$ using a halfspace range reporting data structure, as explained above. As in the case of $S(n)$, the solution of this recurrence is $T(n) = O(n^{\lfloor d/2 \rfloor} \log n)$, provided β is chosen large enough, to satisfy a similar constraint as above. We have thus shown:

Theorem 2.5. *The data structure described in this subsection answers a prefix-minima query in $O(\log n)$ expected time, and requires $O(n^{\lfloor d/2 \rfloor} \log n)$ expected storage and preprocessing time. Again, expectation is with respect to the random choice of the input permutation.*

2.3 The final structure

In this subsection we show how to reduce the storage and preprocessing time even further, so that the query time remains $O(\log n)$. We now take $r = n/\log n$, and build the data structure of Section 2.2 for the set H^r of the first r hyperplanes in π. This takes $O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$ expected storage and preprocessing time.

We construct $LE(H^r)$, triangulate it, and build the ray shooting data structure of Matoušek and Schwarzkopf [23], which supports the operation of locating the prism Δ in the decomposition of $\overline{LE}(H^r)$ that intersects a query vertical line.

For each prism in the decomposition, we compute $C(\Delta)$, using batched halfspace range reporting, as in Section 2.2. We store $C(\Delta)$ with each prism Δ. This completes the description of the data structure and of its construction. (Note that this structure does not use recursion.)

A query with a point $q \in \mathbb{R}^{d-1}$ is answered as follows. We first query the data structure of H^r and find the prefix minima at q that belong to H^r; this takes $O(\log r) = O(\log n)$ time. Then we use the data structure of Matoušek and Schwarzkopf [23] to find the prism Δ in the partition of $\overline{LE}(H^r)$ that is stabbed by the vertical line through q, in additional $O(\log r) = O(\log n)$ time. Finally we scan $C(\Delta)$, to find the prefix minima at q that belong to $C(\Delta)$.

Since, for any fixed q, the expected size of $C(\Delta)$ is $O(n/r) = O(\log n)$, the maximum expected query time is $O(\log n)$.

The maximum expected storage required to store the data structure of the preceding subsection, and the ray shooting data structure of Matoušek and Schwarzkopf for H^r, is $O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$. In addition, we store the conflict lists, whose overall expected size, as implied by Theorem 2.3 (with $t = 1$, $\gamma = 0$), is $O(\frac{n}{r} \cdot r^{\lfloor d/2 \rfloor}) = O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$. Hence, the total expected size of our data structure is $O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$. A similar argument shows that the expected preprocessing time is also $O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$. We thus obtain the main result of this section.

Theorem 2.6. *Let H be a set of n nonvertical hyperplanes in $\mathbb{R}^d$, and let π be a random permutation of H. The data structure described in this subsection answers a prefix-minima query in $\mathcal{A}(H)$ with*

respect to π in $O(\log n)$ expected time, and requires $O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$ expected storage and preprocessing time, where expectation is with respect to the random choice of π.

2.4 Prefix minima in $\mathbb{R}^3$

The same technique can also be applied in three dimensions, with few modifications. First, instead of the range reporting data structure of Matoušek [20], used in the second and third versions of the structure, we use the range reporting data structure of Chan [6], which requires $O(n \log\log n)$ storage, $O(n \log n)$ preprocessing time, and answers a query in $O(k_h + \log n)$ time, where k_h is the output size. Second, instead of the ray shooting data structure of Matoušek and Schwarzkopf [23], we use a planar point location data structure for the minimization diagram of the lower envelope (see, e.g., [10]). Such a structure requires linear storage and $O(n \log n)$ preprocessing time, and answers a query in $O(\log n)$ time. Using these alternatives in the algorithm presented above, we obtain the following theorem.

Theorem 2.7. *Let H be a set of n nonvertical planes in $\mathbb{R}^3$, and let π be a random permutation of H. The data structure described in this subsection answers a prefix-minima query in $\mathcal{A}(H)$ with respect to π in $O(\log n)$ expected time, and requires $O(n)$ expected storage and $O(n \log n)$ expected preprocessing time.*

Remark: This result is related to a data structure of Guibas et al. [14], for point location in incrementally constructed planar Voronoi diagrams. Specifically, they have considered a randomized incremental procedure for constructing the diagram, in which each Voronoi cell is maintained in triangulated form, and have shown that one can use the "history DAG" constructed by the algorithm, to step through the sequence of triangles containing a query point $q \in \mathbb{R}^2$. By modelling the Voronoi diagram as the minimization diagram of the lower envelope of a corresponding set of planes in $\mathbb{R}^3$, the output of the procedure of Guibas et al. yields the sequence of prefix minima of these planes at q, with respect to the random insertion order. The expected query time of the point location mechanism of Guibas et al. is $O(\log^2 n)$, and they have asked whether this could be improved to $O(\log n)$. Our algorithm provides these prefix minima in $O(\log n)$ expected time while maintaining the same expected storage and preprocessing costs as in [14].

2.5 Implications and corollaries

Going back to the problem in arbitrary dimension, we derive several straightforward implications of our results.
(1) Using a standard duality that maps points to hyperplanes and vice versa, while preserving the above / below relationships (see [12]), we obtain the following Corollary of Theorems 2.6 and 2.7.

Corollary 2.8. *Let $\pi = (p_1, \ldots, p_n)$ be a random permutation of a set P of n points in $\mathbb{R}^d$, for $d \geq 4$. Let $P_i := \{p_1, \ldots, p_i\}$, for $i = 1, \ldots, n$, and let $CH(P_i)$ denote the convex hull of P_i. We can preprocess P and π in $O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$ expected time, and build a data structure of expected size $O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$, such that, given a direction ω in $\mathbb{R}^d$, we can retrieve the sequence of all points $p \in P$ that are touched by the planes with outward direction ω that support the incremental hulls $CH(P_i)$, for $i = 1, \ldots, n$, in $O(\log n)$ expected time. For $d = 3$, we obtain a similar data structure, whose expected storage is $O(n)$, the expected preprocessing time is $O(n \log n)$, and the expected query time is $O(\log n)$.*

(2) Another Corollary of Theorem 2.6 and 2.7 is the following.

Corollary 2.9. *Let $\pi = (p_1, \ldots, p_n)$ be a random permutation of a set P of n points in $\mathbb{R}^{d-1}$, for $d \geq 4$. Let $P_i = \{p_1, \ldots, p_i\}$, for $i = 1, \ldots, n$, and let $Vor(P_i)$ denote the (Euclidean) Voronoi diagram of P_i. We can preprocess P and π in $O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$ expected time, and build a data structure of expected size $O(n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n)$, such that, given a query point $\kappa \in \mathbb{R}^{d-1}$, we can retrieve the sequence of all distinct points $p \in P$, whose Voronoi cells in the partial diagrams $Vor(P_i)$ contain κ, in $O(\log n)$ expected time. For $d = 3$, the same result holds, except that the expected storage is $O(n)$, the expected preprocessing time is $O(n \log n)$, and the expected query time is $O(\log n)$.*

Proof. Consider for simplicity the case $d = 2$; the case $d \geq 3$ is handled in an analogous manner. We use the standard lifting transformation [12] that maps each point $p_i = (a_i, b_i) \in P$ to the plane $h_i : z_i = -2a_i x - 2b_i y + a_i^2 + b_i^2$. Let $H_i := \{h_1, \ldots, h_i\}$, for $i = 1, \ldots, n$. It is well known that the xy-projection of $LE(H_i)$ is equal to $Vor(P_i)$ [10, 13]. We can therefore use the data structure of Theorem 2.6 for the set $H = \{h_1, \ldots, h_n\}$, and obtain the asserted performance bounds. □

Note that the algorithm of Corollary 2.9 effectively produces the sequence of the distinct nearest neighbors of κ in the prefix sets P_i.

3 An Alternative Algorithm Using Antennas

In this section we consider a different solution to the rank-minimum problem, which uses a variant of the technique of Mulmuley [25, Chapter 6] and of Schwarzkopf [26, Section 4]. This technique is based on structures known as *antennas*, a notion that we review and apply next. Mulmuley uses this structure to perform efficiently point location and ray shooting queries in an arrangement of hyperplanes in $\mathbb{R}^d$, while Schwarzkopf addresses the same problems for points (or ray origins) below the lower envelope of the given hyerplanes. Our solution uses some tools from these previous studies, but it is considerably simpler than either of them.

We begin by defining antennas and some related terminology. Let H be a collection of n hyperplanes in $\mathbb{R}^d$. Given a point $p \in \mathbb{R}^d$ below $LE(H)$, and an arbitrary direction ζ, we define the *antenna* of p in $\mathcal{A}(H)$, with respect to ζ (we sometimes call it the antenna of p and ζ), as follows. We shoot from p in the directions ζ and $-\zeta$, until we meet two respective points p^+, p^- on $LE(H)$ (or reach infinity—see below). We continue the construction recursively from p^+ and from p^-. Consider p^+, for example, and let $h \in H$ be the hyperplane containing it. We then shoot from p^+ along h, forward and backwards, in some fixed direction, which, for simplicity, we take to be the orthogonal projection of ζ onto h (assuming ζ is not orthogonal to h). These shootings hit two other respective hyperplanes, at points that lie on two respective $(d-2)$-dimensional faces of $LE(H)$. We continue the shootings recursively within each of these faces, and reach points on four respective $(d-3)$-faces of $LE(H)$, and keep doing so until we reach a total of at most 2^d vertices of the envelope. (We allow some of these shootings to reach infinity—this will be the case if, say, ζ is the x_d-direction (or sufficiently close to it); in this case the downward-directed shooting from p will not hit any hyperplane. We will record this unboundedness in an appropriate symbolic manner.) The collection of all the segments traced during the shootings constitute the *antenna of* p, denoted by $\mathsf{antenna}(p)$. It has (up to) 2^d vertices where the bottommost-level shootings have ended; all of them are vertices of $LE(H)$. In what follows, we refer to these terminal points as the *vertices* of $\mathsf{antenna}(p)$.

Here is a brief description of our data structure. We first describe it for the original problem of ray-shooting and point location below the lower envelope, and then show how to extend this to handle minimum-rank (or, more generally, prefix minima) queries.

The structure is recursive, and is based on "repeated halving" of H, resulting in a sequence of subsets (also called a "gradation") of H, $H_0 = H \supseteq H_1 \supseteq H_2 \supseteq \cdots \supseteq H_k$, where $k = \lfloor \log n \rfloor$. For each $i = 0, \ldots, k-1$, H_{i+1} is a random subset of exactly $|H_i|/2$ hyperplanes of H_i. For each $i = 1, \ldots, k$, we construct $LE(H_i)$, and compute, for each vertex v of $LE(H_i)$, its *conflict list*, which is the set of all hyperplanes in $H_{i-1} \setminus H_i$ which pass below v. A simple method to construct all these envelopes and conflict lists is as follows. Choose a random permutation π of H, and take each H_i to be the prefix of π consisting of the first $n/2^i$ elements of π. Clearly, each H_{i+1} is a random subset of exactly $|H_i|/2$ hyperplanes of H_i. Now run the randomized incremental algorithm of Clarkson and Shor [8] for constructing $LE(H)$, using π as the random insertion order. While running the algorithm, we maintain the conflict lists of the vertices currently on the lower envelope, where each list is ordered by increasing weight (rank in π). This is easy to do without affecting the asymptotic bounds on the expected behavior of the randomized incremental construction. We pause each time a full prefix H_i has been inserted (in the order $i = k, k-1, \ldots, 0$), and copy into our output the current envelope, and, for each of its vertices v, the prefix of the conflict list of v which contains hyperplanes from H_{i-1} (all these data are available and easily accessible at this point, since the full conflict lists are maintained in sorted rank-order). The expected running time of the algorithm (with respect to the random choice of π) is $O(n^{\lfloor d/2 \rfloor})$ for $d \geq 4$, and $O(n \log n)$ for $d = 3$.

Finally, for each i and for each 3-dimensional face f of $LE(H_i)$, we preprocess f, and each of its facets, for logarithmic-time ray-shooting queries, from points inside f towards its boundary, or from points inside some facet towards its boundary, using standard techniques that produce linear-size data structures and take linear preprocessing time. (In three dimensions we use the Dobkin-Kirkpatrick hierarchy [11], and for each planar face we simply store its cyclic boundary representation.)

Answering ray-shooting queries. Let ρ be a query ray, emanating from a point q that lies below $LE(H)$ in direction ζ. We answer the query by constructing $\mathsf{antenna}(q)$ in H with respect to ζ. Clearly, this antenna encodes the point where ρ hits $LE(H)$.

We construct $\mathsf{antenna}(q)$ by the following bottom-up iterative manner. For each i, let $\mathsf{antenna}_i(q)$ denote $\mathsf{antenna}(q)$ in H_i (with respect to ζ); thus our goal is to compute $\mathsf{antenna}_0(q)$. To do so, we first compute $\mathsf{antenna}_k(q)$, in brute force, and then, stepping back through the gradation of the H_i's, we compute $\mathsf{antenna}_i(q)$ from $\mathsf{antenna}_{i+1}(q)$, until we get the desired $\mathsf{antenna}_0(q)$. Each edge of any of these antennas stores the set of hyperplanes that contain it; the size of this set is 0 for the first two edges (those emanating from q, assuming a generic position of q), and increases by 1 at each recursive step of the construction.

To compute $\mathsf{antenna}_i(q)$ from $\mathsf{antenna}_{i+1}(q)$, we need the following easy lemma.

Lemma 3.1. *If a (non-vertical) hyperplane $h \in H_i \setminus H_{i+1}$ crosses an edge of $\mathsf{antenna}_{i+1}(q)$, then it must pass below some vertex of $\mathsf{antenna}_{i+1}(q)$, i.e., it must belong to the conflict list of such a vertex.*

Proof: Suppose to the contrary that all vertices of $\mathsf{antenna}_{i+1}(q)$ lie below h. By construction, all the edges of $\mathsf{antenna}_{i+1}(q)$ are contained in the convex hull of its vertices, and thus the antenna lies fully below h, so h cannot cross any of its edges, a contradiction that completes the proof. □

Hence, to construct $\mathsf{antenna}_i(q)$ from $\mathsf{antenna}_{i+1}(q)$, we scan the conflict lists of all the vertices of $\mathsf{antenna}_{i+1}(q)$, and check which of the hyperplanes appearing in these lists intersects the antenna, and where. From this we can obtain the *trimmed antenna*, namely, the maximal connected portion of it that avoids all the new hyperplanes and contains q. Each terminal point of the trimmed portion lies on one of the hyperplanes in the conflict lists.

We need to complete the trimmed portion of $\mathsf{antenna}_{i+1}(q)$ into a full antenna in H_i. Let v be one of these terminal points, lying in the intersection of all the hyperplanes pre-stored with the edge terminating at v, and in the new hyperplane that has cut the old antenna at v. Let f denote the intersection of all these hyperplanes. We then have to shoot within f, forward and backwards along some direction (say, a direction depending on ζ), until another hyperplane is hit, and keep doing so recursively, until a whole new portion of the antenna, "hanging" from v is constructed. See Figure 1. Repeating this completion process from all terminal vertices of the trimmed portion, we obtain the desired $\mathsf{antenna}_i(q)$.

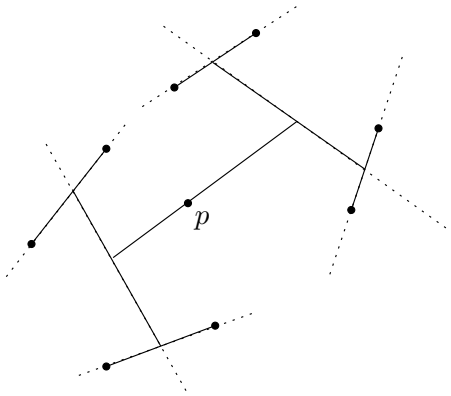

Figure 1: The antenna of p in $\mathbb{R}^3$.

The structure of Mulmuley [25] has performed this antenna completion phase by preparing a recursive copy of the structure within each intersection flat f of any subset of the hyperplanes of H_i (for each i), with respect to the hyperplanes $h \cap f$, for all $h \in H_i$ not containing f. However, the overall input size of all these subproblems (not to mention their overall output size) is $\Omega(n^d)$, as is easily checked, which is too expensive for our goal. Schwarzkopf's structure uses less storage, but it also maintains recursive (and rather complicated) substructures associated with lower-dimensional faces of the envelope.

Instead, we only maintain the full-dimensional gradation-based structure, and perform each of the ray shootings in the antenna completion stage as if it were a new independent ray shooting query in d-space, Thus, on one hand we essentially ignore the extra information that the ray lies in some lower-dimensional flat, but on the other hand we do keep track of the dimension of the flat within which the current shooting takes place, so that we stop when we reach 0-dimensional features (vertices of the current $LE(H_i)$). Each of these recursive shooting steps starts again at H_k, and constructs the antenna by stepping backwards through the gradation. To speed up the query time, when we reach 3-dimensional flats, we switch to the data structures stored with the corresponding faces of $LE(H_i)$, and use them to perform the remaining shootings for the completion of the antenna.

Analysis: Query time. Denote by $Q_j(n)$ the maximum expected cost of a query within a j-dimensional flat, formed by the intersection of $d-j$ of the given hyperplanes. We have $Q_3(n) = O(\log n)$. For $j > 3$ we argue as follows. We recursively obtain the antenna for the next set in the gradation, consisting of $n/2$ hyperplanes. Then we scan the conflict lists of the at most 2^j vertices of the antenna, and find all the intersections of the hyperplanes in these lists with the antenna. We obtain various terminal points, lying on various lower-dimensional flats, and we recurse from each

of them within the corresponding flat. We thus obtain the recurrence

$$Q_j(n) = Q_j(n/2) + O(C_p) + \sum_u Q_{j_u}(n), \tag{7}$$

where C_p is the sum of the sizes of the conflict lists of the vertices of the old $\mathsf{antenna}(p)$, and the sum is over the terminal points of the trimmed antenna, each with its own dimension j_u (all strictly smaller than j). It follows from the theory of ε-nets that, with high probability, each of the conflict lists contains at most $O(\log n)$ hyperplanes, for a total of $C_p = O(2^j \log n)$. In fact, as mentioned in the remark following Theorem 2.3 (with $r = n/2$), the expected sum of the sizes of these lists is only $O(1)$.

It is easy to verify by induction (on j and n) that the solution of (7) is $Q_j(n) = O(\log^{j-2} n)$, so the cost of the original query, where $j = d$, is $O(\log^{d-2} n)$.

Analysis: Storage and preprocessing. The maximum expected storage $S(n)$ satisfies the recurrence

$$S(n) = S(n/2) + O(n^{\lfloor d/2 \rfloor} + C), \tag{8}$$

where C is the total size of the conflict lists of all the vertices of $LE(H)$. A simple application of Theorem 2.3 shows that the expected value of C is $O(n^{\lfloor d/2 \rfloor})$, so, asymptotically, it has no effect on the recurrence (8), whose solution is then easily verified to be $S(n) = O(n^{\lfloor d/2 \rfloor})$.

To bound the maximum expected preprocessing time $T(n)$, we recall that most of it is devoted to the construction of the lower envelopes $LE(H_i)$, for $i \geq 0$, and the conflict lists of all the vertices of each envelope; the time to construct the ray-shooting structures for all 3-dimensional faces is linear in the overall complexity of these facets, and is thus subsumed by the cost of the other steps. As described above, the construction of all the lower envelopes and the conflict lists of their vertices can be performed using the standard randomized incremental construction technique, whose expected running time is $O(n^{\lfloor d/2 \rfloor})$ for $d \geq 4$, and $O(n \log n)$ for $d = 3$. The extra steps of copying envelopes and conflict lists at the appropriate prefixes do not increase the asymptotic bound on the running time. We have thus obtained the first main result of this section.

Theorem 3.2. *Given a set H of n hyperplanes in $\mathbb{R}^d$, one can construct a data structure of expected size $O(n^{\lfloor d/2 \rfloor})$, so that, for any query point $p \in \mathbb{R}^d$ that lies below $LE(H)$, and a query direction ζ, we can construct $\mathsf{antenna}(p)$ in H with respect to ζ in expected time $O(\log^{d-2} n)$. In particular, for any query ray ρ that emanates from a point below $LE(H)$, we can find the point where ρ hits $LE(H)$ (or report that no such point exists) within the same expected time. The expected preprocessing time is $O(n^{\lfloor d/2 \rfloor})$ for $d \geq 4$, and $O(n \log n)$ for $d = 3$.*

Remark: The algorithm can be easily modified so that it can detect whether the query point p lies above $LE(H)$. For this we take ζ to be the upward vertical direction (i.e., the x_d-direction). In this case, the "bottom half" of $\mathsf{antenna}(p)$ (rooted at the segment that emanates from p downwards) does not exist, or alternatively consists of a single unbounded ray. When we pass from $\mathsf{antenna}_{i+1}(p)$ to $\mathsf{antenna}_i(p)$, we scan the conflict lists of the vertices of the former antenna. If we find a hyperplane that passes below p, we stop the whole process and report that p lies above $LE(H)$. Otherwise we continue as above. It is easy to verify the correctness of this procedure.

Answering prefix-minima queries. We now return to our original goal, of using antennas to answer prefix-minima queries with respect to a random permutation π of the input hyperplanes. The pleasant surprise is that, with very few changes, the preceding algorithm solves this problem.

Specifically, as already described above, the gradation sequence $H_0, H_1, \ldots$ is constructed by taking each H_i to be the prefix of the first $n/2^i$ elements of π. We compute all the lower envelopes $LE(H_i)$ and the associated conflict lists using the technique described above, that is, by a single execution of the randomized incremental algorithm, using π as the insertion permutation. As above, we pause each time a full prefix H_i has been inserted (in the order $i = k, k-1, \ldots, 0$), and copy the current envelope and its conflict lists into our output.

Given a query point $q \in \mathbb{R}^{d-1}$, we regard it as a point in $\mathbb{R}^d$ whose x_d-coordinate is $-\infty$, and construct the antennas $\mathsf{antenna}_i(q)$, with respect to the vertical shooting direction.[5] Let e be the first edge of $\mathsf{antenna}_{i+1}(q)$, incident to q and to some point on $LE(H_{i+1})$. While constructing $\mathsf{antenna}_i(q)$ from $\mathsf{antenna}_{i+1}(q)$, we identify all hyperplanes of $H_i \setminus H_{i+1}$ that intersect e, by collecting them off the prefixes of the appropriate conflict lists. We then scan these hyperplanes by increasing weight (i.e., rank in π), which is the order in which they appear in the conflict lists, and add each hyperplane h whose intersection with e is lower than all previous intersections to the sequence of prefix minima. By concatenating the resulting sequences, we obtain the complete sequence of prefix minima at q.

The query time, storage, and preprocessing cost have the same expected bounds as above, and so we get the main result of this section.

Theorem 3.3. *Given a set H of n hyperplanes in $\mathbb{R}^d$, and a random permutation π thereof, one can construct a data structure of expected size $O(n^{\lfloor d/2 \rfloor})$, so that, for any query point $p \in \mathbb{R}^d$, the sequence of prefix minima of π among the hyperplanes intersecting the vertical line through p can be found in expected time $O(\log^{d-2} n)$. The expected preprocessing time is $O(n^{\lfloor d/2 \rfloor})$ for $d \geq 4$, and $O(n \log n)$ for $d = 3$.*

Remarks: (1) Theorem 3.3 has Corollaries analogous to Corollaries 2.8 and 2.9 of Theorems 2.6 and 2.7, which we do not spell out.
(2) The reason for including the antenna-based data structure in this paper is its elegance and simplicity (and also because of its applicability to point location and ray shooting below a lower envelope, providing a simple alternative to Schwarzkopf's structure). It is easy to implement—both the preprocessing stage and the procedure for answering queries. Although we have not implemented the algorithm, we suspect that it will behave well in practice, and will outperform the theoretically more efficient cutting-based approach of Section 2. Experimentation with the algorithm might reveal that in practice the bound $O(\log^{d-2} n)$ on the expected query time is over-pessimistic, and that its actual performance is much better on average.

4 Approximate Range Counting

In this section we exploit the machinery developed in the preceding sections, to obtain efficient algorithms for the approximate half-space range counting problem in $\mathbb{R}^d$. Recall that in this application we are given a set P of n points in $\mathbb{R}^d$, which we want to preprocess into a data structure, such that, given a lower halfspace h^- bounded by a plane h, we can efficiently approximate the number of points in $P \cap h^-$ to within a relative error of ε, with high probability (i.e., the error probability should go down to zero as $1/\text{poly}(n)$).

As explained in the introduction, following the framework of Cohen [9], we construct $O\left(\frac{1}{\varepsilon^2} \log n\right)$ copies of the data structure provided in Corollary 2.8, each based on a different random permutation of the points, obtained by sorting the points according to the random weights that they are assigned

[5] Here we do not need the "bottom half" of the antenna, which is empty anyway.

from the exponential distribution (see the introduction and [9] for details).[6] We now query each of the structures with the normal ω to h pointing into h^-. From the sequence of points that we obtain we retrieve, in $O(\log n)$ time, the point of minimum rank in $P \cap h^-$, and record its weight. We output the reciprocal of the average of these weights as an estimator for the desired count.

The preceding analysis thus implies the following results. Because of slight differences, we consider separately the cases $d \geq 4$ and $d = 3$.

Theorem 4.1. *Let P be a set of n points in $\mathbb{R}^d$, for $d \geq 4$. We can preprocess P into a data structure of expected size $O(\frac{1}{\varepsilon^2} n^{\lfloor d/2 \rfloor} \log^{2-\lfloor d/2 \rfloor} n)$, in $O(\frac{1}{\varepsilon^2} n^{\lfloor d/2 \rfloor} \log^{2-\lfloor d/2 \rfloor} n)$ expected time, so that, for a query halfspace h^-, we can approximate $|P \cap h^-|$ up to a relative error of ε, in $O(\frac{1}{\varepsilon^2} \log^2 n)$ expected time, so that the approximation is guaranteed with high probability.*

In $\mathbb{R}^3$ we can reduce the storage using the following technique. Let P be the given set of n points in $\mathbb{R}^3$. Consider the process that draws one of the $O\left(\frac{1}{\varepsilon^2} \log n\right)$ random permutations π of P. Let R denote the set of the first $t = \varepsilon^2 n / \log n$ points in π. Clearly, R is a random sample of P of size t, where each t-element subset is equally likely to be chosen. Moreover, conditioned on R having a fixed value, the prefix π_t of the first t elements of π is a random permutation of R.

We now construct our data structure of Corollary 2.8 for R and π_t only. The expected size of this data structure is $O(t) = O(\varepsilon^2 n / \log n)$. Repeating this for $O\left(\frac{1}{\varepsilon^2} \log n\right)$ permutations, the total expected storage is $O(n)$. The total expected construction cost is $O\left(\frac{1}{\varepsilon^2} \log n \cdot t \log t\right) = O(n \log n)$.

A query half-space h^- is processed as follows. For each permutation π and associated prefix R, we find the point of R of minimum rank that lies in h^-. If there exists such a point, it is also the minimum-rank point of $P \cap h^-$ and we proceed as above. Suppose however that $R \cap h^- = \emptyset$. In this case, since R is a random sample of P of size t, the ε-net theory [16] implies that, with high probability, $|P \cap h^-| = O\left(\frac{n}{t} \log t\right) = O\left(\frac{1}{\varepsilon^2} \log^2 n\right)$. In this case, we can afford to *report* the points in $P \cap h^-$ in time $O\left(\frac{1}{\varepsilon^2} \log^2 n\right)$, using the range reporting data structures mentioned in the introduction. For example, the algorithm of Chan [6] uses $O(n \log \log n)$ storage, $O(n \log n)$ preprocessing time, and reports the k points of $P \cap h^-$ in time $O(\log n + k) = O\left(\frac{1}{\varepsilon^2} \log^2 n\right)$. We then count the number of reported points exactly, by brute force. We proceed in this way if $R \cap h^-$ is empty for at least one of the samples R. Otherwise, we correctly collect the minimum-rank elements in each of the permutations, and can obtain the approximate count as above.

In summary, we thus have:

Theorem 4.2. *Let P be a set of n points in $\mathbb{R}^3$, and let $\varepsilon > 0$ be given. Then we can preprocess P into a data structure of expected size $O(n \log \log n)$, in $O(n \log n)$ expected time, so that, given a query halfspace h^-, we can approximate, with high probability, the count $|P \cap h^-|$ to within relative error ε, in $O\left(\frac{1}{\varepsilon^2} \log^2 n\right)$ expected time.*

We note, though, that the structure of Afshani and Chan [1] gives a better solution.

Remark: The same technique applies to the problem of approximate range counting of points in a query ball in $\mathbb{R}^{d-1}$, using the lifting transform, as described earlier. We obtain analogous theorems to Theorems 4.1 and 4.2, using Corollary 2.9; we omit their explicit statements.

References

[1] P. Afshani and T. M. Chan, On approximate range counting and depth, *Proc. 23rd Annu. ACM Sympos. Comput. Geom.*, 2007, 337–343.

[6] To get the best asymptotic bounds, we use the data structure of Section 2, instead of the antenna-based one, but the latter structure can of course also be used.

[2] B. Aronov and S. Har-Peled, On approximating the depth and related problems, *Proc. 16th Annu. ACM-SIAM Sympos. Discrete Algo.*, 2005, 886–894.

[3] B. Aronov and S. Har-Peled, On approximating the depth and related problems, 2006, submitted.

[4] B. Aronov, S. Har-Peled and M. Sharir, On approximate halfspace range counting and relative epsilon-approximations, *Proc. 23rd Annu. ACM Sympos. Comput. Geom.*, 2007, 327–336.

[5] B. Aronov and M. Sharir, Approximate range counting, Manuscript, 2007.

[6] T. M. Chan, Random smapling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions, *SIAM J. Comput.* 30 (2000), 561–575.

[7] B. Chazelle, *The Discrepancy Method*, Cambridge University Press, Cambridge, UK, 2000.

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

[9] E. Cohen, Size-estimation framework with applications to transitive closure and reachability, *J. Comput. Syst. Sci.* 55 (1997), 441–453.

[10] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd edition, Springer Verlag, Heidelberg, 2000.

[11] D. P. Dobkin and D. G. Kirkpatrick, Determining the separation of preprocessed polyhedra — A unified approach, *proc. 17th Internat. Colloq. Automata, Languages and Programming*, 1990, 400–413.

[12] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[13] H. Edelsbrunner and R. Seidel, Voronoi diagrams and arrangements, *Discrete Comput. Geom.* 1 (1986), 25–44.

[14] L. Guibas, D. E. Knuth, and M. Sharir, Randomized incremental construction of Voronoi and Delaunay diagrams, *Algorithmica* 7 (1992), 381–413.

[15] S. Har-Peled and M. Sharir, Relative ε-approximations in geometry, manuscript, 2006. (Also in [4].)

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

[17] H. Kaplan, E. Ramos and M. Sharir, The overlay of minimization diagrams during a randomized incremental construction, Manuscript, 2007.

[18] H. Kaplan and M. Sharir, Randomized incremental constructions of three-dimensional convex hulls and planar voronoi diagrams, and approximate range counting, *Proc. 17th Annu. ACM-SIAM Sympos. Discrete Algo.*, 2006, 484–493.

[19] Y. Li, P. M. Long, and A. Srinivasan, Improved bounds on the sample complexity of learning, *J. Comput. Syst. Sci.* 62 (2001), 516–527.

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

[21] J. Matoušek, Efficient partition trees, *Discrete Comput. Geom.* 8 (1992), 315–334.

[22] J. Matoušek, Range searching with efficient hierarchical cuttings, *Discrete Comput. Geom.* 10 (1993), 157–182.

[23] J. Matoušek and O. Schwarzkopf, On ray shooting in convex polytopes, *Discrete Comput. Geom.* 10 (1993), 215–232.

[24] P. McMullen, The maximum numbers of faces of a convex polytope, *Mathematika* 17 (1970), 179–184.

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

[26] O. Schwarzkopf, Ray shooting in convex polytopes, *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, 1992, 886–894.