Discrete Comput Geom (2011) 45: 3–33
DOI 10.1007/s00454-010-9308-6

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

Haim Kaplan · Edgar Ramos · Micha Sharir

Received: 30 September 2007 / Revised: 6 October 2010 / Accepted: 11 October 2010 /
Published online: 11 November 2010
© Springer Science+Business Media, LLC 2010

Abstract In approximate halfspace range counting, one is given a set P of n points in $\mathbb{R}^d$, 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 Kaplan and Sharir (Proc. 17th Annu. ACM-SIAM Sympos. Discrete Algo., pp. 484–493, 2006), which is based, as is the present paper, on Cohen's technique for approximate range counting (Cohen in J. Comput. Syst. Sci. 55:441–453, 1997). 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

The work by Haim Kaplan was partially supported by Grant 2006/204 from the U.S.—Israel Binational Science Foundation, and by Grant 975/06 from the Israel Science Fund (ISF). The work by Micha Sharir was partially supported by NSF Grants CCR-05-14079 and CCR-08-30272, by Grant 2006/194 from the U.S.—Israel Binational Science Foundation, by Grants 155/05 and 338/09 from the Israel Science Fund, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

H. Kaplan (✉) · M. Sharir
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
e-mail: haimk@post.tau.ac.il

M. Sharir
e-mail: michas@post.tau.ac.il

E. Ramos
Department of Computer Science, Universidad Nacional de Colombia, Medellín, Colombia
e-mail: earamosn@unalmed.edu.co

M. Sharir
Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA

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, $O(n^{\lfloor d/2\rfloor}(\log\log n)^c/\log^{\lfloor d/2\rfloor} n)$ expected storage and preprocessing time, for some constant c, and answers a range-minimum query in $O(\log n)$ expected time. We also present a different approach, based on "antennas," which is simple to implement, although the bounds on its expected storage, preprocessing, and query costs are worse by polylogarithmic factors.

Keywords Range searching · Random sampling · Approximate range counting · Randomized incremental constructions · Range minima queries

1 Introduction

1.1 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 also apply 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 time, the best solution requires $\Omega(n^d)$ storage, and if we allow only linear storage, the best known query time is $\Omega(n^{1-1/d})$ [23]. For example, when $d=3$ we need $\Omega(n^{2/3})$ time for an exact counting query, with 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

$$(1-\varepsilon)|P\cap r|\le N\le(1+\varepsilon)|P\cap r|. \tag{1}$$

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.

1.2 Using ε-Approximations

There is a simple well-known method that almost achieves this goal. Choose a random sample E of $\frac{c}{\varepsilon^2}\log\frac{1}{\varepsilon}$ points of P, for some sufficiently large absolute constant c

(which depends linearly on the *VC-dimension* of the problem [7]). Then, with constant probability, E is an ε-*approximation* for P (see, e.g., [7]), in the sense that, with constant 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 rescaling 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 the 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 *relative* ε-approximations; see [16, 20]. However, the technique continues to be applicable only to sufficiently large ranges (larger than some prespecified threshold, which affects the size of the ε-approximation).

1.3 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 [10], which estimates the number of data objects in a given range r as follows. We assign to each data object, independently, a random *weight*, drawn from an exponential distribution with density function e^{-x}, and find the *minimum weight* of an object in the query range r. We repeat this experiment $O(\frac{1}{\varepsilon^2} \log n)$ times, compute the average μ of the weights of the minimum elements, and approximate $|P \cap r|$ by $1/\mu$. (Cohen [10] also proposes several other estimators that have similar properties.) As shown in [10], this approximate count lies, with high probability,[1] within the 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 [10] 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,[2] and let $\pi = (p_1, \ldots, p_n)$ 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 with high probability; see [10].) 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 $P \cap h$ with the smallest rank in π.

[1] By the phrase "with high probability" we mean probability $1 - q_n$, where q_n is polynomially small in n, the size of the input.

[2] 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 [11].

1.4 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, independent of the random permutation, we want to find the first hyperplane in π (the one with the smallest weight) which passes below[3] 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. In the standard "oblivious" model, when the choice of q is independent of π, π induces a random permutation of the heights of the hyperplanes above q, and hence the expected number of prefix minima at q is $O(\log n)$. The hyperplane with minimum weight below p is the first hyperplane in the (decreasing) 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 first describe a straightforward black-box reduction from the prefix minima problem to the problem of vertical ray shooting into lower envelopes of hyperplanes. We construct a balanced binary tree T over the sequence $h_1, \ldots, h_n$. In each node v of T we construct a vertical ray shooting data structure into the lower envelope of the hyperplanes stored at the leaves of the subtree of v. We find the prefix minima above a point q as follows. We first perform a ray shooting query in the structure at the root of T, which contains all hyperplanes. This yields the last hyperplane h_j in the sequence of prefix minima above q. The next-to-last hyperplane in the sequence of prefix minima above q is the hyperplane attaining the lower envelope of $h_1, \ldots, h_{j-1}$ at q. We find this hyperplane by up to $\log n$ ray shooting queries using the data structures of the nodes hanging off to the left of the path in T from the root to h_{j-1}. We continue in the same way to construct the entire sequence of prefix minima backwards from last to first. Since the expected length of this sequence is $O(\log n)$ and finding each one of them requires at most $\log n$ ray shooting queries, the expected query time into the resulting data structure is larger by a multiplicative factor of $O(\log^2 n)$ than the time it takes to perform a vertical ray shooting query. The space and the preprocessing time are larger by a multiplicative factor of $O(\log n)$ than the respective space and the preprocessing time of the data structure for vertical ray shooting. Using the data structure of Matoušek and Schwarzkopf [24, Theorem 10] for vertical ray shooting, we obtain, for $d > 3$, a data structure with query time $O(\log^3 n)$ and space and preprocessing time $O(n^{\lfloor d/2 \rfloor} \log^{1+\delta-\lfloor d/2 \rfloor} n)$, for any $\delta > 0$. In $\mathbb{R}^3$ we can

[3] 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.

use the data structure of Dobkin and Kirkpatrick [12] and obtain an expected query time of $O(\log^3 n)$, $O(n \log n)$ space, and $O(n \log^2 n)$ preprocessing time.

In this paper we present two additional algorithms for the prefix minima problem, which work in any dimension. We obtain our algorithms by carefully adapting known randomized range searching techniques. 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.

The first algorithm improves by polylogarithmic factors the bounds obtained by the black-box reduction described above. The second one is obtained using a simplification of the "antenna" technique of Mulmuley [27, Chap. 6] and of Schwarzkopf [28, Sect. 4]. For large d, the bounds we obtain from the second structure are inferior to the ones we get by the black-box reduction. The main merit of this structure is its simplicity (even if we consider it only as a vertical ray shooting data structure).

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 in the resulting data structure is[4] $O(\log n)$, 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 $O(n^{\lfloor d/2 \rfloor} (\log \log n)^c / \log^{\lfloor d/2 \rfloor} n)$ expected storage and preprocessing time for some constant c (that depends on the dimension d). 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 resembles the ray shooting data structure of Matoušek and Schwarzkopf [24], but its storage and preprocessing costs are smaller by roughly a logarithmic factor than those in [24].

Our second data structure is based on "antennas" (see Mulmuley [27, Chap. 6] and Schwarzkopf [28, Sect. 4]). Let H_i be the set of the first $n/2^i$ hyperplanes in π. These subsets form a "gradation" $H_0 = H \supseteq H_1 \supseteq H_2 \supseteq \cdots \supseteq H_k$ of H, where $k = \lfloor \log n \rfloor$, and where $|H_k| = 1$. 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, as above, 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 the arrangement 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 Sect. 3

[4]Recall that we assume that the query is independent of the random permutation.

for details). While constructing these antennas, we can easily find the minimum-rank hyperplane below p.

The query time of this data structure in dimension $d > 3$ is $O(\log^{d-2} n)$ with high probability, its expected storage is $O(n^{\lfloor d/2 \rfloor})$, and the expected preprocessing time is also $O(n^{\lfloor d/2 \rfloor})$. In $\mathbb{R}^3$ the query time is $O(\log^2 n)$ with high probability, the expected storage is $O(n)$, and the expected preprocessing time is $O(n \log n)$. 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, its main merit is that it is much simpler than the previous variants considered in [27, 28].

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 [27] 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 (and 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 [28, Sect. 4] is designed for ray shooting queries from points below the lower envelope of H. The structure uses data collected during a randomized insertion process, and does not use a gradation. As a result, it 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, as already stated, is much simpler.

Returning to our original goal, we next plug the first algorithm into the general approximate range counting framework of Cohen [10]. In $\mathbb{R}^3$, we obtain an algorithm that uses $O(\frac{1}{\varepsilon^2} n \log n)$ expected storage and $O(\frac{1}{\varepsilon^2} n \log^2 n)$ expected preprocessing time, and answers a query in $O(\frac{1}{\varepsilon^2} \log^2 n)$ 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 a factor of $1 \pm \varepsilon$ 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 in Sect. 4, brings the expected storage down to $O(n \log \log n)$ and the expected preprocessing time to $O(n \log n)$, without affecting the bound on the query time. In $\mathbb{R}^d$, for $d > 3$, the

expected query time is $O(\frac{1}{\varepsilon^2}\log^2 n)$, and the expected storage and preprocessing time is $O(\frac{1}{\varepsilon^2}n^{\lfloor d/2\rfloor}\log^{1-\lfloor d/2\rfloor} n(\log\log n)^g)$.

1.5 Related Work

Besides 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 [19].

Aronov and Har-Peled [3] showed how to reduce the approximate range counting problem to *range emptiness*. As a particular application of their reduction, they obtain in $\mathbb{R}^3$ a data structure which requires $O(\frac{1}{\varepsilon^2}n\log n)$ storage and $O(\frac{1}{\varepsilon^2}n\log^2 n)$ expected preprocessing time, and answers a query in $O(\frac{1}{\varepsilon^2}\log^2 n)$ time, such that the result is correct with high probability. These bounds coincide with those that we have obtained in [18].

The technique of Aronov and Har-Peled [3] is applicable whenever range emptiness can be solved efficiently, and in particular for approximate halfspace range counting in higher dimensions. Indeed, assume that we have a data structure for range emptiness queries 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 $O((\varepsilon^{\lambda-3} + H_{\lambda-2}(\frac{1}{\varepsilon}))S(n)\log n)$ space, $O((\varepsilon^{\lambda-3} + H_{\lambda-2}(\frac{1}{\varepsilon}))T(n)\log n)$ preprocessing time, and answers a query with high probability in $O(\frac{1}{\varepsilon^2}Q(n)\log n)$ 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 $O(\log\frac{n}{k})$ 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 and Sharir [4] (see also [5]). In this approach, they take the partition-tree data structure of Matoušek [21], which facilitates

efficient range emptiness queries, or range reporting queries for *shallow* ranges,[5] 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) 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 that 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 $d > 3$, 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 [4, 5], 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 a 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 Computing Prefix Minima of a Random Permutation of Hyperplanes Above a Query Point

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 $q \in \mathbb{R}^{d-1}$, independent of π, 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 [24] 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

[5] That is, more efficient than the partition-tree structure for general (halfspace or simplex) range counting, which handles *arbitrary* ranges [22].

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 $d > 3$ in Sects. 2.1–2.5, and discuss the case $d = 3$ in Sect. 2.6.

2.1 The First Simple Structure

Our data structure is recursive, similar to other divide-and-conquer geometric algorithms based on sampling, as, e.g., in [8, Sect. 4.4]. Here however we use the random permutation π as the only source of randomness. Each recursive subproblem is specified by a subset H_0 of H, and the subpermutation π_0 of H_0 is 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 [25], 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

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

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 $Q(n) \leq c \log n$, for an appropriate constant c. Indeed, using induction, and the fact that $\log x$ is a concave function, Jensen's inequality implies that

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

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. We note that the results derived below can also be obtained by a careful and somewhat technical application of the general theory presented in [8, Theorem 3.6].

2.2 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 $b(\Delta) \leq d(d+1)/2$, 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 (one of which, say h, contains the simplex), and, recursively, the hyperplanes defining the $(d-1)$-dimensional simplex (inside h) obtained by removing the bottom vertex. 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 $d > 1$, $b(d) = d + b(d-1)$. Hence $b(d) = d(d+1)/2$. 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 (at most b elements 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*

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

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 Sect. 2.4.

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

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

where the constant of proportionality depends on s.

Proof We may assume that $r \geq \sqrt{3n}$. 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 $b(\Delta) = \beta$. Set

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

Then

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

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

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

where $x = c/\ell$ and $y = (r-\beta-c)/(n-\ell-\beta+1)$; note that $0 \leq x, y \leq 1$, by the restrictions on the values of ℓ.

By the definition of x and y we have that $x \leq \frac{y}{2}$ exactly when

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

which is equivalent to

$$\ell \geq \frac{2c(n-\beta+1)}{r+c-\beta}. \tag{2}$$

Since $r+c-\beta \geq \frac{2}{3}r$ for $r \geq 3(b+c)$ (actually for $r \geq 3b$), it follows from (2) that $x \leq \frac{y}{2}$ if

$$\ell > \frac{3c(n-\beta+1)}{r}.$$

The last inequality holds for $\ell > (n/r)^2$, provided that $r \le n/(3c)$, so $x \le \frac{y}{2}$ in this case.

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

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

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

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

(Note that if $2(n/r)^2 > n - r + c$, then $p_{\ell,\beta} = 0$ for each $\ell \ge 2(n/r)^2$. The assumptions $\sqrt{3n} \le r \le n/(3c)$ are easily seen to imply that $2(n/r)^2 \le n - r + c$, so the ongoing analysis does indeed have to be performed.) We conclude that the contribution of those Δ's with $\ell(\Delta) \ge 2(n/r)^2$ is at most

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

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 $t \ge 1$, $\gamma \ge 0$, and $3(b + \lceil t \rceil) \le r \le \min\{n/(3\lceil t \rceil), n/(4b \log n)\}$, we have*

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

where A is a constant that depends on d, t, and γ.

Proof Let

$$Z = \mathbf{E}\Big\{ \sum_{\Delta\in\mathbf{\Delta}^0(R)} \ell(\Delta)^t \log^\gamma\big(\ell(\Delta)+1\big)\Big\}$$
$$= \sum_{\Delta\in\mathbf{\Delta}(F)} \Pr\big\{\Delta\in\mathbf{\Delta}^0(R)\big\}\ell(\Delta)^t \log^\gamma\big(\ell(\Delta)+1\big). \tag{3}$$

Set $c=\lceil t\rceil$, $\alpha = t/c \le 1$, $s=\gamma/\alpha$, and $W(x)=x^\alpha$.

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

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

for any $\ell(\Delta)\ge 0$. We can therefore rewrite (3) as

$$Z \le a\sum_{\Delta\in\mathbf{\Delta}(F)} \Pr\big\{\Delta\in\mathbf{\Delta}^0(R)\big\} W\Big(\binom{\ell(\Delta)}{c}\log^s\big(\ell(\Delta)+1\big)\Big)$$
$$+ e\sum_{\Delta\in\mathbf{\Delta}(F)} \Pr\big\{\Delta\in\mathbf{\Delta}^0(R)\big\}$$
$$= \Bigg[a\sum_{\Delta\in\mathbf{\Delta}(F)} \frac{\Pr\{\Delta\in\mathbf{\Delta}^0(R)\}}{\mathbf{E}\{|\mathbf{\Delta}^0(R)|\}} W\Big(\binom{\ell(\Delta)}{c}\log^s\big(\ell(\Delta)+1\big)\Big)+e\Bigg]\cdot\mathbf{E}\big\{\big|\mathbf{\Delta}^0(R)\big|\big\}.$$

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

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

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

$$Z \le \Bigg[aW\Big(\sum_{\Delta\in\mathbf{\Delta}(F)} \frac{\binom{n-b(\Delta)-\ell(\Delta)}{r-b(\Delta)}}{\binom{n}{r}\mathbf{E}\{|\mathbf{\Delta}^0(R)|\}}\binom{\ell(\Delta)}{c}\log^s\big(\ell(\Delta)+1\big)\Big)+e\Bigg]\cdot\mathbf{E}\big\{\big|\mathbf{\Delta}^0(R)\big|\big\}. \tag{4}$$

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

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

and therefore, substituting into (4), we get

$$Z \le \Bigg[aW\Big(\frac{(n-r+c)(n-r+c-1)\cdots(n-r+1)}{(r-b)(r-b-1)\cdots(r-b-c+1)}$$
$$\times \sum_{\Delta\in\mathbf{\Delta}(F)} \frac{\binom{n-b(\Delta)-\ell(\Delta)}{r-b(\Delta)-c}}{\binom{n}{r}\mathbf{E}\{|\mathbf{\Delta}^0(R)|\}}\binom{\ell(\Delta)}{c}\log^s\big(\ell(\Delta)+1\big)\Big)+e\Bigg]\cdot\mathbf{E}\big\{\big|\mathbf{\Delta}^0(R)\big|\big\}.$$

Since we assume that $3(b+c) \le r \le \min\{n/(3c), n/(4b\log n)\}$, with $c = \lceil t \rceil$, Lemma 2.2 implies that

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

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

$$\begin{aligned} Z &\le A'\left[\left(\frac{n}{r}\right)^t \log^\gamma \frac{n}{r} \cdot \left(\frac{\mathbf{E}\{|\mathbf{\Delta}^c(R)|\}}{\mathbf{E}\{|\mathbf{\Delta}^0(R)|\}} \right)^\alpha + 1 \right] \cdot \mathbf{E}\{|\mathbf{\Delta}^0(R)|\} \\ &= A'\left[\left(\frac{n}{r}\right)^t \log^\gamma \frac{n}{r} \cdot \mathbf{E}\{|\mathbf{\Delta}^c(R)|\}^\alpha \mathbf{E}\{|\mathbf{\Delta}^0(R)|\}^{1-\alpha} + \mathbf{E}\{|\mathbf{\Delta}^0(R)|\} \right], \end{aligned} \tag{5}$$

where A' is an appropriate constant that depends on d, t, and γ. Recall that $|\mathbf{\Delta}^0(R)| = O(r^{\lfloor d/2 \rfloor})$. A standard application of another basic result of Clarkson and Shor [8, Corollary 3.3] implies that $|\mathbf{\Delta}^c(R)| = O(c^{\lceil d/2 \rceil} r^{\lfloor d/2 \rfloor})$. 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 (5), 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 (where the latter assertion is an easy consequence of the Clarkson-Shor technique). 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.

2.3 Analysis: Storage and Preprocessing

The expected storage required by the algorithm satisfies the recurrence

$$S(n) \le \begin{cases} \mathbf{E}\{\sum_{\Delta \in \mathbf{\Delta}^0(H^r)} S(\ell(\Delta))\} + O(r^{\lfloor d/2 \rfloor}), & \text{for } n \ge r, \\ O(r), & \text{for } n < r, \end{cases} \tag{6}$$

where the expectation is over the choice of the random permutation π. Indeed, 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 (6).

We prove, by induction on n, that, for any $\delta > 0$ there is a constant $B = B(\delta)$, such that $S(n) \le Bn^{\lfloor d/2 \rfloor + \delta}$. Indeed, this will be the case for $n \le r$ 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 (6), we obtain

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

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

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

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

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

where the overhead nonrecursive cost is dominated by the cost of constructing the conflict lists $C(\Delta)$, by 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 the following.

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.*

2.4 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, nonconstant 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 $r = |H_0|^{\beta}$. For the initial instance, $H_0 = H$ and $r = n^{\beta}$. We describe below how to process this initial instance. Processing of all recursive instances is analogous, with n replaced by the number of hyperplanes in the recursive instance and r redefined accordingly. The random permutation of a recursive instance involving a subset H_0 of H is the restriction of π to H_0.

Similar to the first structure, we take, for each subproblem, the set H^r of the first r hyperplanes in π, construct their lower envelope $LE(H^r)$, 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. This is easily done in $O(r^{\lfloor d/2 \rfloor})$ expected time using the randomized incremental construction algorithm of Clarkson and Shor [8].

However, in contrast with the algorithm of Sect. 2.1, 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 [24], for finding the hyperplane in $LE(H^r)$ which is first hit by a query ray originating below $LE(H^r)$.

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 $LE(H^r)$, 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 $LE(H^r)$ (there is a unique such face lying on h^1). 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 $LE(H^r)$. Our next query is with a ray emanating from $bot(F(h^1, h^2))$ and passing through q^2. We continue in this manner until our query ray is contained in an edge e of $LE(H^r)$. 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 (see [24] for details) and answers a query in $O(\log r)$ time. Indeed, the cost of a single ray shooting query in [24] is $O(\log r)$, and we perform $d-1$ of them.[6]

In addition, we build the simple prefix minima data structure of Sect. 2.1 for H^r since, 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 $\overline{LE}(H^r)$. We obtain $C(\Delta)$ easily if we compute $LE(H^r)$ using randomized incremental construction, where we insert the first r hyperplanes one by one in their order in π. As is standard with randomized incremental constructions, we maintain a bipartite conflict graph in which each edge[7] e in the lower envelope is connected to the hyperplanes, not yet inserted, conflicting with e. The twist here is that we run the randomized incremental algorithm for r steps only, but we maintain the conflict graph with respect to all n hyperplanes. Thus, after inserting the first r hyperplanes, it is straightforward to recover the conflict lists of the prisms in our triangulation from the conflict lists of the edges of $LE(H^r)$ (using the fact that any hyperplane that crosses a vertical prism Δ must pass below at least one of its vertices).[8]

As mentioned above, it is standard to prove, either using the technique of [8] or using backwards analysis (see, e.g., [9]), that constructing $LE(H^r)$ using randomized incremental construction takes $O(r^{\lfloor d/2 \rfloor})$ expected time. Here, however, we maintain conflict lists with respect to all n hyperplanes, which makes the computation more expensive. Nevertheless, it still follows from the analysis mentioned above (see Theorem 3.9 of [8]) that running the first r steps out of n in a randomized incremental

[6]This algorithm to identify Δ is similar to the one described by Meiser [26]. Meiser's result preceded the ray shooting data structure of Matoušek and Schwarzkopf [24] and used instead a naive ray shooting algorithm.

[7]This is the dual setup to the one used in the randomized incremental construction of the convex hull of the dual points, where we maintain conflict lists of "ridges" ($(d-2)$-dimensional faces) of the hull.

[8]In fact, it may be more natural to use the randomized incremental construction algorithm to construct $LE(H^r)$, its triangulation, and the conflict lists of the prisms in the decomposition of $\overline{LE}(H^r)$ all at the same time.

construction algorithm takes $O(\frac{n}{r} r^{\lfloor d/2 \rfloor})$ expected time. (Note that by Theorem 2.3 (with $t = 1$, $\gamma = 0$) this is asymptotically the same as the expected size of conflict lists of the simplices in the triangulation of $LE(H^r)$.)

An alternative way to compute $C(\Delta)$ for each Δ in the decomposition of $\overline{LE}(H^r)$ is to take the set V of all the vertices of the prisms Δ (that is, the set of all vertices of $LE(H^r)$), and to preprocess it for halfspace range reporting, as in Matoušek [21]. Then, for each $h \in H$, 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. Using the algorithm in [21, Theorem 1.1], and setting $m = |V| = O(r^{\lfloor d/2 \rfloor})$, this can be done, in dimension $d > 3$, 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 n queries is

$$O\left(m \log m + n m^{1-1/\lfloor d/2 \rfloor} \log^c m + \sum_{h \in H_0} k_h\right)$$

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

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, as mentioned before, by Theorem 2.3 with $t = 1$, $\gamma = 0$, it follows that the expected sum of the sizes of the conflict lists is

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

Hence, the overall cost of this step is

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

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 dominated by the size of the simple data structure of Sect. 2.1 constructed for the hyperplanes in H^r, which is $O(r^{\lfloor d/2 \rfloor + \delta})$, for any $\delta > 0$, as implied by Theorem 2.4.

Once we have computed $C(\Delta)$ for all Δ in the decomposition of $\overline{LE}(H^r)$, 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 [24] 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

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

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, assuming inductively that $Q(n_\Delta) \leq c \log n_\Delta$, we get the inequality

$$\begin{aligned} Q(n) &\leq \mathbf{E}\{c \log n_\Delta\} + O(\log n) \leq c \log\big(\mathbf{E}\{n_\Delta\}\big) + O(\log n) \\ &\leq c \log\big(O(n/r)\big) + O(\log n) \leq c \log\big(O\big(n^{1-\beta}\big)\big) + O(\log n) \leq c \log n, \end{aligned}$$

with an appropriate choice of c, as asserted.

The maximum expected storage required by the algorithm satisfies the recurrence

$$S(n) \leq \begin{cases} \mathbf{E}\big\{\sum_{\Delta \in \mathbf{\Delta}^0(H^r)} S(\ell(\Delta))\big\} + O(r^{\lfloor d/2 \rfloor + \delta}), & \text{for } n \geq n_0, \\ O(1), & \text{for } n < n_0. \end{cases} \tag{7}$$

We prove, using induction on n, that $S(n) \leq B n^{\lfloor d/2 \rfloor} \log n$, 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 (7), and obtain

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

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

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

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

$$S(n) \leq B A n^{\lfloor d/2 \rfloor} \log \frac{n}{r} + O\big(r^{\lfloor d/2 \rfloor + \delta}\big) \leq B(1-\beta) A n^{\lfloor d/2 \rfloor} \log n + C n^{\beta(\lfloor d/2 \rfloor + \delta)},$$

for an appropriate constant C that depends on d and on δ. We now choose β so that $(1-\beta)A \leq 1/2$, then choose δ so that $\beta(\lfloor d/2 \rfloor + \delta) \leq \lfloor d/2 \rfloor$, and finally choose B to satisfy $B \geq 2C$. 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

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

where the overhead nonrecursive 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 sufficiently close to 1, to satisfy a similar constraint as above. We have thus shown the following.

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.*

2.5 The Final Structure

In this subsection we show how to reduce the storage and preprocessing time even further, so that the query time remains $O(\log n)$. Assume that $d \geq 4$, and set $\gamma = \frac{\lfloor d/2 \rfloor}{\lfloor d/2 \rfloor - 1}$. We now take $r = n(\log\log n)^c / \log^\gamma n$ for some constant $c > 0$ which we specify later, and build the data structure of Sect. 2.4 for the set H^r of the first r hyperplanes in π. This takes

$$O\bigl(n^{\lfloor d/2 \rfloor} \log\log^{c\lfloor d/2 \rfloor} n / \log^{\frac{\lfloor d/2 \rfloor^2}{\lfloor d/2 \rfloor - 1} - 1} n\bigr)$$

expected storage and preprocessing time.

We construct $LE(H^r)$, triangulate it, and build the ray shooting data structure of Matoušek and Schwarzkopf [24], which supports the operation of locating the prism Δ in the decomposition of $\overline{LE}(H^r)$ that intersects a query vertical line; see Sect. 2.4 for details.

For each prism in the decomposition, we compute $C(\Delta)$, using randomized incremental construction, as in Sect. 2.4. Note that here, to obtain the best preprocessing time, we have to use randomized incremental construction rather than batched halfspace range reporting. (In the previous subsection it did not matter which of the two methods we use.)

Let $m = |C(\Delta)|$; we store $C(\Delta)$ in a data structure for prefix minima queries that takes $O(m \log m)$ space, $O(m \log^2 m)$ preprocessing time, and answers a query in $O(m^{1-1/\lfloor d/2 \rfloor} \log^{c'} m)$ time, for an appropriate constant c'. We obtain this data structure by applying the black-box reduction described in the introduction to the vertical ray shooting data structure of [2] (which is obtained by combining parametric search with the data structure of Matoušek [21] for range emptiness).

We answer a query with a point $q \in \mathbb{R}^{d-1}$ 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 [24] 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 use the data structure of $C(\Delta)$, described in the previous paragraph, to find the prefix minima at q that belong to $C(\Delta)$. Since the expected size of $C(\Delta)$ is $O(n/r) = O(\log^\gamma n/(\log\log n)^c)$ (see the remark following Theorem 2.3), then by choosing $c = c'/(1 - 1/\lfloor d/2 \rfloor)$ we obtain that the query time within $C(\Delta)$ is also $O(\log n)$.

The expected storage required for H^r in the data structure of the preceding subsection is $O(n^{\lfloor d/2\rfloor}\log\log^{c\lfloor d/2\rfloor} n/\log^{\frac{\lfloor d/2\rfloor^2}{\lfloor d/2\rfloor-1}-1} n)$, which dominates the space required for H^r in the ray shooting data structure of Matoušek and Schwarzkopf. In addition, we store a data structure of size $O(m\log m)$ for each conflict list $C(\Delta)$ of size m. The overall expected size of these data structures, as implied by Theorem 2.3 (with $t=1$, $\gamma=1$), is $O(\frac{n}{r}\cdot r^{\lfloor d/2\rfloor}\log\frac{n}{r}) = O(n^{\lfloor d/2\rfloor}(\log\log n)^{c''}/\log^{\lfloor d/2\rfloor} n)$ for some constant c''. Hence, the total expected size of our data structure is

$$O\bigl(n^{\lfloor d/2\rfloor}(\log\log n)^{c''}/\log^{\lfloor d/2\rfloor} n\bigr).$$

The expected preprocessing time required to compute the data structure of the preceding subsection, and the ray shooting data structure of Matoušek and Schwarzkopf, for H^r is

$$O\bigl(n^{\lfloor d/2\rfloor}\log\log^{c\lfloor d/2\rfloor} n/\log^{\frac{\lfloor d/2\rfloor^2}{\lfloor d/2\rfloor-1}-1} n\bigr).$$

As discussed in Sect. 2.4, the time required to compute the conflict lists $C(\Delta)$ is proportional to their size, and therefore dominated by the time required to compute the data structures for these conflict lists. The expected value of the latter is $O(\frac{n}{r}\cdot r^{\lfloor d/2\rfloor}\log^2\frac{n}{r}) = O(n^{\lfloor d/2\rfloor}(\log\log n)^{c''+1}/\log^{\lfloor d/2\rfloor} n)$ by Theorem 2.3 (with $t=1$, $\gamma=2$).

We thus obtain the main result of this section.

Theorem 2.6 *Let H be a set of n nonvertical hyperplanes in $\mathbb{R}^d$, for $d\geq 4$, 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 $O(n^{\lfloor d/2\rfloor}(\log\log n)^g/\log^{\lfloor d/2\rfloor} n)$ expected storage and preprocessing time, for an appropriate constant g which depends on d, where the expectation is with respect to the random choice of π.*

2.6 Prefix Minima in $\mathbb{R}^3$

The same technique can also be applied in three dimensions, with few modifications. First, we can compute the conflict lists in the second and third stages of our construction either by randomized incremental construction or by using the halfplane range reporting data structure of Chan [6]. The latter requires $O(n\log\log n)$ storage and $O(n\log n)$ 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 [24], we use a planar point location data structure for the minimization diagram of the lower envelope (see, e.g., [11]). 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, modified as above, 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. [15], 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 modeling 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 [15].

2.7 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 [13]), 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 $d > 3$. 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 $d > 3$. 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 [13] 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 $Vor(P_i)$ [11, 14]. 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 prefix minima problem, which uses a variant of the technique of Mulmuley [27, Chap. 6] and of Schwarzkopf [28, Sect. 4]. This technique is based on structures known as *antennas*, a notion that we review and apply next. Mulmuley uses this structure to efficiently perform 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 hyperplanes. Our solution uses some tools from these previous studies, but it is considerably simpler than either of them. We focus first on the case $d > 3$ and indicate later how this technique can also be applied when $d = 3$.

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. (Assuming a generic position of p, h is unique, and so is each of the hyperplanes hit in subsequent steps of this process.) 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 $k = \lfloor \log n \rfloor$ and $|H_k| = 1$. For each $i = 0, \ldots, k-1$, H_{i+1} is a random subset of exactly $\lfloor |H_i|/2 \rfloor$ 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 (in the prefix minima application, given later, π is the input random permutation), and take each H_i to be the prefix of π consisting of the first $\lfloor n/2^i \rfloor$ elements of π. Clearly, each H_{i+1} is a random subset of exactly $\lfloor |H_i|/2 \rfloor$ 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 this data is 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 $d > 3$.

Finally, for each i and for each three-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. For three-dimensional faces we use the Dobkin–Kirkpatrick hierarchy [12], and for each planar face we simply store its edges in a search tree ordered consistently with their cyclic order along the boundary of the face.

3.1 Answering Ray Shooting Queries

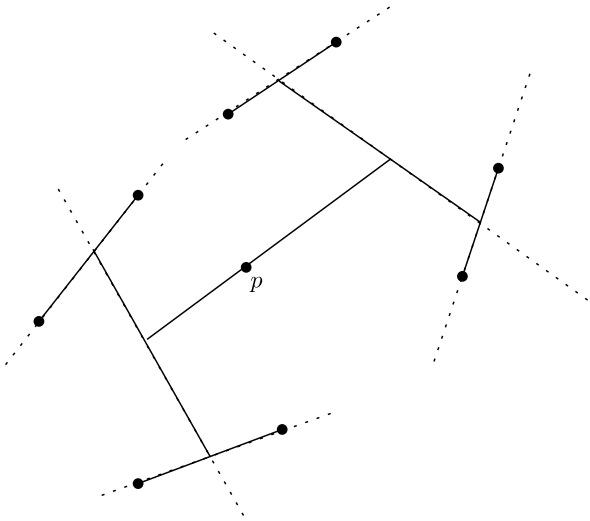
Let ρ be a query ray, emanating from a point q that lies below $LE(H)$ in direction ζ. We answer the query by constructing $\mathsf{antenna}(q)$ in H with respect to ζ. Clearly, this antenna encodes the point where ρ hits $LE(H)$.

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 $\mathsf{antenna}_k(q)$, by brute force (recall that $|H_k| = 1$), 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 (nonvertical) 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.*

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



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.

Next, 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 the terminal points of the trimmed antenna, lying in the intersection of all the hyperplanes prestored 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 from v 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 Fig. 1. Repeating this completion process from all terminal vertices of the trimmed portion, we obtain the desired $\mathsf{antenna}_i(q)$.

The structure of Mulmuley [27] 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 three-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.

3.2 Analysis: Query Time

Let p be an arbitrary query point in some j-dimensional flat defined by the given hyperplanes, and let $C_p(m)$ be the sum of the sizes of the conflict lists of the vertices of the antenna of p in the arrangement of the first $m/2$ hyperplanes in π, with respect to the next $m/2$ hyperplanes. Here $m = \lfloor n/2^i \rfloor$, for $i = 0, 1, \ldots, \log n - 1$. We argue that, with high probability, $C_p(m) = O(2^j \log n)$, where the failure probability is polynomially small in n. Note that the bound is expressed in terms of the total number n of hyperplanes.

Abusing the notation slightly, we let $C_v(m)$ denote the size of the conflict list of a vertex v of the dual plane arrangement, which appears on the lower envelope of the first $m/2$ hyperplanes of π, with respect to the next $m/2$ hyperplanes. $C_v(m)$ is a random variable which depends on π; it is zero if v does not appear on that envelope. $C_p(m)$ is the sum of at most 2^j of these quantities.

We claim that, for any $b > 1$, $C_v(m) \le A \log n$ with probability at least $1 - 1/n^b$, where $A = A(b)$ depends on b. This follows[9] from the ε-net theory [17], which asserts that a random sample of size $cr(\log r + \log(1/q))$, for an appropriate constant c proportional to d, is a $(1/r)$-net with probability at least $1 - q$. Set $q = 1/n^b$ and choose r so that $cr(\log r + \log(1/q)) = cr(\log r + b \log n) = m/2$. That is, we choose $r = m/(c'b \log n)$, for another absolute constant c'. Then, with probability at least $1 - 1/n^b$, we have, for every v as above,

$$C_v(m) \le m/r = c'b \log n.$$

We apply this bound for each of the values $m_i = \lfloor n/2^i \rfloor$, for $i = 0, \ldots, \log n - 1$. By the probability union bound, and by the definition of $C_p(m)$, it follows that, for any fixed b, the following holds with probability at least $1 - (\log n)/n^b$: For any query point p in a j-dimensional flat, and for any $i = 0, \ldots, \log n - 1$, we have that

$$C_p(m_i) \le 2^j c'b \log n.$$

[9] This bound can also be established directly using the probability union bound over the $O(m^d)$ vertices of the arrangement of the m hyperplanes, since each such vertex appears on the lower envelope with conflict list of size $\ge b' \log n$ with probability at most $O(1/n^{b'})$.

Now, to complete the analysis of the query time, let $Q_j(m)$ be an upper bound on the maximum cost of a query within a j-dimensional flat, formed by the intersection of $d-j$ of the first m hyperplanes, which will be guaranteed to hold when the upper bound on $C_p(m)$ holds for every p and m. That is, $Q_j(m)$ is an upper bound on the query time within a j-dimensional flat which holds with the high probability guaranteed above. We have $Q_3(m) = O(\log n)$ (which actually holds with certainty). For $j > 3$ we get that $Q_j(m)$ satisfies the following recurrence:

$$Q_j(m) \le Q_j(m/2) + 2^j c'' \log n + \sum_u Q_{j_u}(m), \tag{8}$$

for yet another constant c'', which depends (linearly) on the polynomial degree b for which the probability that $Q_j(m)$ indeed satisfies the recurrence (8), for every j and m, is $1 - (\log n)/n^b$; note that, as above, the overhead terms are all expressed in terms of the same initial value n.

It is now straightforward to verify, by induction on j and m, that (8) solves to

$$Q_j(m) \le B_j \log^{j-2} n,$$

for an appropriate constant B_j which depends on j and on b.

Hence, for any $b > 1$, the cost of the original query, where $j = d$ and $m = n$, is $O(\log^{d-2} n)$, with probability at least $1 - 1/n^b$, where the constant of proportionality depends on d and on b.

3.3 Analysis: Storage and Preprocessing

The maximum expected storage $S(n)$ satisfies the recurrence

$$S(n) = S(n/2) + O\bigl(n^{\lfloor d/2 \rfloor} + C\bigr), \tag{9}$$

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 (9), 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 $i \ge 0$, and of the conflict lists of all the vertices of each envelope. The time to construct the ray shooting structures for all three-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 $d > 3$. 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 $\mathbb{R}^d$, for $d > 3$, 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 antenna(p) *in H with respect to ζ in $O(\log^{d-2} n)$ time, where this running time holds with high probability. 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 time. The expected preprocessing time is $O(n^{\lfloor d/2 \rfloor})$.*

Remark One can easily detect if p is below $LE(H)$ as follows. Do a vertical ray shooting query with a vertical ray through p that starts at $-\infty$. The point p is below $LE(H)$ if and only if it is below the hyperplane returned by the ray shooting query.

3.4 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, we take π as the random permutation used by the algorithm. 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 $\lfloor n/2^i \rfloor$ 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.[10] 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 bounds as above. (For the query cost, we note that scanning the conflict lists is already done by the part of the algorithm that constructs the antennas.) We thus get the main result of this section.

Theorem 3.3 *Given a set H of n hyperplanes in $\mathbb{R}^d$, for $d > 3$, 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 $p \in \mathbb{R}^d$, we can compute the sequence of prefix minima of π among the hyperplanes intersecting the vertical line through p, so that the running time is $O(\log^{d-2} n)$ with high probability. The expected preprocessing time is $O(n^{\lfloor d/2 \rfloor})$.*

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

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 its applicability to point location and ray shooting below a lower envelope, providing a simple alternative to Schwarzkopf's structure). It is relatively easy to implement—both the preprocessing stage and the procedure for answering queries. In particular, one can implement a version of this structure which does not use the Dobkin–Kirkpatrick data structure for three-dimensional faces but rather uses the gradation to answer ray shooting queries inside these faces as well. See (3) below. The structure becomes simpler, at the cost of an extra logarithmic factor in the query time. 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 Sect. 2. Experimentation with the algorithm might reveal that in practice the bound $O(\log^{d-2} n)$ on the expected query time is too pessimistic, and that its actual performance is much better on average.

(3) As just mentioned, we can also apply this antenna-based algorithm in $\mathbb{R}^3$. As before, we let H_i be the prefix of length $\lfloor n/2^i \rfloor$ of the hyperplanes in the permutation π. For each vertex v of the lower envelope of H_i, we construct the *conflict list* of v with respect to H_{i-1} using randomized incremental construction. Here we do not use the Dobkin–Kirkpatrick ray shooting data structure, but we store the edges of each (planar) facet of each lower envelope in a search tree ordered consistently with their cyclic order along the boundary of the facet. The expected space required by the data structure is linear, and the expected preprocessing time is $O(n \log n)$. A prefix minima query takes $O(\log^2 n)$ time with high probability since the reconstruction of $\mathsf{antenna}_i(q)$ from $\mathsf{antenna}_{i+1}(q)$ takes $O(\log n)$ time with high probability.

4 Approximate Range Counting

In this section we exploit the machinery developed in the preceding sections to obtain efficient algorithms for the approximate halfspace 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 $1/\mathrm{poly}(n)$).

As explained in the introduction, following the framework of Cohen [10], we construct $O(\frac{1}{\varepsilon^2} \log n)$ 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 from the exponential distribution (see the introduction and [10] for details).[11] 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 $O(\log n)$ expected time, the point of minimum rank in $P \cap h^-$, and record

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

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 $d > 3$ and $d = 3$.

Theorem 4.1 *Let P be a set of n points in $\mathbb{R}^d$. For $d > 3$, we can preprocess P into a data structure of expected size $O(\frac{1}{\varepsilon^2} n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n (\log\log n)^g)$, in $O(\frac{1}{\varepsilon^2} n^{\lfloor d/2 \rfloor} \log^{1-\lfloor d/2 \rfloor} n (\log\log n)^g)$ expected time, for an appropriate constant g (depending on d), so that, for a query halfspace h^-, we can approximate $|P \cap h^-|$ up to a relative error of ε with probability polynomially small in n, in $O(\frac{1}{\varepsilon^2} \log^2 n)$ expected time. For $d = 3$ we can preprocess P into a data structure of expected size $O(\frac{1}{\varepsilon^2} n \log n)$, in $O(\frac{1}{\varepsilon^2} n \log^2 n)$ expected time, and answer an approximate counting query, as above, in $O(\frac{1}{\varepsilon^2} \log^2 n)$ expected time.*

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 $O(\frac{1}{\varepsilon^2} \log n)$ random permutations π of P. Let R denote the set of the first $t = \varepsilon^2 n / \log n$ points in π. (For this to be meaningful, n has to be larger than $2^{1/\varepsilon^2}$.) 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 $O(\frac{1}{\varepsilon^2} \log n)$ permutations, the total expected storage is $O(n)$. The total expected construction cost is $O(\frac{1}{\varepsilon^2} \log n \cdot t \log t) = O(n \log n)$.

A query halfspace 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 [17] implies that, with high probability, $|P \cap h^-| = O(\frac{n}{t} \log t) = O(\frac{1}{\varepsilon^2} \log^2 n)$. In this case, we can afford to *report* the points in $P \cap h^-$ in time $O(\frac{1}{\varepsilon^2} \log^2 n)$, 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 $O(\log n + k) = O(\frac{1}{\varepsilon^2} \log^2 n)$. 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 the following.

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 probability polynomially small in n, the count $|P \cap h^-|$ to within relative error ε, in $O(\frac{1}{\varepsilon^2} \log^2 n)$ 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.

Acknowledgements The authors thank two anonymous referees for their helpful comments, which have led to significant improvements in the presentation of the paper. In particular, one of the referees pointed out the black-box reduction of the prefix minima problem to vertical ray shooting described in the introduction.

References

1. Afshani, P., Chan, T.M.: On approximate range counting and depth. In: Proc. 23rd Annu. ACM Sympos. Comput. Geom., pp. 337–343 (2007)
2. Agarwal, P.K., Matoušek, J.: Ray shooting and parametric search. SIAM J. Comput. **22**(4), 794–806 (1993)
3. Aronov, B., Har-Peled, S.: On approximating the depth and related problems. SIAM J. Comput. **38**(3), 899–921 (2008)
4. Aronov, B., Sharir, M.: Approximate range counting. SIAM J. Comput. **39**(7), 2704–2725 (2010)
5. Aronov, B., Har-Peled, S., Sharir, M.: On approximate halfspace range counting and relative epsilon-approximations. In: Proc. 23rd Annu. ACM Sympos. Comput. Geom., pp. 327–336 (2007)
6. Chan, T.M.: Random sampling, halfspace range reporting, and construction of $(\leq k)$-levels in three dimensions. SIAM J. Comput. **30**, 561–575 (2000)
7. Chazelle, B.: The Discrepancy Method. Cambridge University Press, Cambridge (2000)
8. Clarkson, K., Shor, P.: Applications of random sampling in computational geometry, II. Discrete Comput. Geom. **4**, 387–421 (1989)
9. Clarkson, K., Mehlhorn, K., Seidel, R.: Four results on randomized incremental constructions. Comput. Geom. **3**, 185–212 (1993)
10. Cohen, E.: Size-estimation framework with applications to transitive closure and reachability. J. Comput. Syst. Sci. **55**, 441–453 (1997)
11. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Heidelberg (2000)
12. Dobkin, D.P., Kirkpatrick, D.G.: Determining the separation of preprocessed polyhedra—A unified approach. In: Proc. 17th Internat. Colloq. Automata, Languages and Programming, pp. 400–413 (1990)
13. Edelsbrunner, H.: Algorithms in Combinatorial Geometry. Springer, Heidelberg (1987)
14. Edelsbrunner, H., Seidel, R.: Voronoi diagrams and arrangements. Discrete Comput. Geom. **1**, 25–44 (1986)
15. Guibas, L., Knuth, D.E., Sharir, M.: Randomized incremental construction of Voronoi and Delaunay diagrams. Algorithmica **7**, 381–413 (1992)
16. Har-Peled, S., Sharir, M.: Relative ε-approximations in geometry. Discrete Comput. Geom. in press. (Also in http://arxiv.org/abs/0909.0717)
17. Haussler, D., Welzl, E.: Epsilon-nets and simplex range queries. Discrete Comput. Geom. **2**, 127–151 (1987)
18. Kaplan, H., Sharir, M.: Randomized incremental constructions of three-dimensional convex hulls and planar Voronoi diagrams, and approximate range counting. In: Proc. 17th Annu. ACM-SIAM Sympos. Discrete Algo, pp. 484–493 (2006)
19. Kaplan, H., Ramos, E., Sharir, M.: The overlay of minimization diagrams during a randomized incremental construction. Manuscript (2007)
20. Li, Y., Long, P.M., Srinivasan, A.: Improved bounds on the sample complexity of learning. J. Comput. Syst. Sci. **62**, 516–527 (2001)
21. Matoušek, J.: Reporting points in halfspaces. Comput. Geom. Theory Appl. **2**, 169–186 (1992)
22. Matoušek, J.: Efficient partition trees. Discrete Comput. Geom. **8**, 315–334 (1992)

23. Matoušek, J.: Range searching with efficient hierarchical cuttings. Discrete Comput. Geom. **10**, 157–182 (1993)
24. Matoušek, J., Schwarzkopf, O.: On ray shooting in convex polytopes. Discrete Comput. Geom. **10**, 215–232 (1993)
25. McMullen, P.: The maximum numbers of faces of a convex polytope. Mathematika **17**, 179–184 (1970)
26. Meiser, S.: Point location in arrangements of hyperplanes. Inf. Comput. **106**, 286–303 (1993)
27. Mulmuley, K.: Computational Geometry: An Introduction Through Randomized Algorithms. Prentice-Hall, Englewood Cliffs (1994)
28. Schwarzkopf, O.: Ray shooting in convex polytopes. In: Proc. 8th Annu. ACM Sympos. Comput. Geom., pp. 886–894 (1992)