

Self-improving Algorithms for Convex Hulls

Kenneth L. Clarkson *

Wolfgang Mulzer †

C. Seshadhri ‡

October 16, 2009

Abstract

We describe an algorithm for computing planar convex hulls in the self-improving model: given a sequence I_1, I_2, \dots of planar n -point sets, the upper convex hull $\text{conv}(I)$ of each set I is desired. We assume that there exists a probability distribution \mathcal{D} on n -point sets, such that the inputs I_j are drawn independently according to \mathcal{D} . Furthermore, \mathcal{D} is such that the individual points are distributed independently of each other. In other words, the i 'th point is distributed according to \mathcal{D}_i . The \mathcal{D}_i 's can be arbitrary but are independent of each other. The distribution \mathcal{D} is not known to the algorithm in advance. After a learning phase of n^ϵ rounds, the expected time to compute $\text{conv}(I)$ is $O(n + H(\text{conv}(I)))$. Here, $H(\text{conv}(I))$ is the entropy of the output, which is a lower bound for the expected running time of *any* algebraic computation tree that computes the convex hull. (More precisely, $H(\text{conv}(I))$ is the minimum entropy of any random variable that maps I to a description of $\text{conv}(I)$ and to a labeling scheme that proves nonextremality for every point in I not on the hull.) Our algorithm is thus asymptotically optimal for \mathcal{D} .

1 Introduction

Twenty three years after “The ultimate planar convex hull algorithm?” [12], what more could possibly be said about the task of computing the convex hull of a set of points in the plane? This problem, among the most fundamental of computational geometry, has been treated in a wealth of settings, and often serves as a “test subject” for novel computational models and analyses.

Here we consider the classical computational setting of comparison-based, exact algorithms on a random access machine. (The comparisons may involve, for exam-

ple, algebraic expressions over the input coordinates.) Our analysis is in the *self-improving* model introduced by Ailon et al. [3]: a sequence I_1, I_2, \dots of pointsets are given, and the convex hull $\text{conv}(I_j)$ of each is desired. We assume that the pointsets are related to each other, so that data from past problem instances may be helpful in speeding up computation for new instances.

Throughout, we refer to the convex hull, but our algorithms and discussion consider only the *upper hull*, the upper boundary of the convex hull. This is no loss of generality.

Self-improving algorithms. If the instances are random, then the output description is a random variable, and has an entropy $H(\text{conv}(I))$.¹ For any algorithm that uses comparisons, or other primitive operations that give at most a constant number of bits of information per call, the output entropy is a lower bound on the time needed: the results of the comparisons determine the output, and therefore the algorithm gives a kind of coding scheme for the output; by Shannon's theorem, any such coding scheme must use a number of bits proportional to the entropy.

For instances in an initial *learning phase*, we only promise a running time bounded by the worst case. Thereafter, in the *limiting phase*, the running time for each instance I is expected $O(n + H(\text{conv}(I)))$, for a large class of distributions of inputs, and so our running time in the limiting phase is optimal among comparison-based algorithms.

The self-improving model of algorithmic analysis, in which it is possible to give algorithms that use past instances to improve performance for new instances, was introduced by Ailon et al. [3], who gave and analyzed algorithms for sorting, and extended by Clarkson and Seshadhri [7] to the case of computing Delaunay triangulations. (These two efforts were merged and improved [2].) As also done here, these prior results show that *entropy-optimal* performance is achievable in the limiting phase.

*IBM Almaden Research Center, San Jose, USA. Email: klclarks@us.ibm.com

†Dept. of Computer Science, Princeton University, Princeton, USA. Email: wmulzer@cs.princeton.edu. Supported in part by NSF grant CCF-0634958, NSF CCF 083279, and a Wallace Memorial Fellowship.

‡IBM Almaden Research Center, San Jose, USA. Email: csesha@us.ibm.com

¹When we refer to $H(\text{conv}(I))$ we are abusing notation a bit: it is a particular kind of description of the upper hull that has an entropy, not the upper hull regarded as simply a polygonal chain. The specifics are in § 1.1.

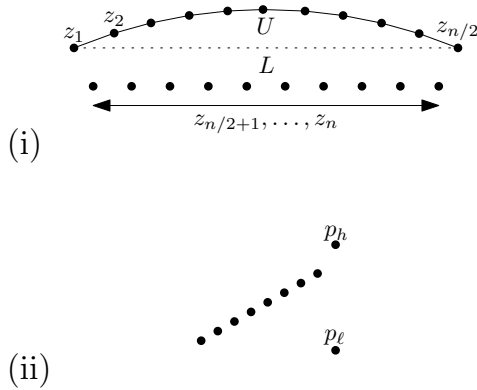


Figure 1: (i) A pointset and distribution that is slow in other models: the upper hull U is fixed, while the points $z_{n/2+1}, \dots, z_n$ roughly constitute a random permutation of the points in L . (ii) Point z_1 is either at p_h or p_ℓ , and its location affects the processing for the other points.

As mentioned, our results hold for a large class of input distributions; as previously shown [2], some distributions require exponential space for optimal running time, so some restrictions on the input distribution must be made to obtain reasonable results. We follow the previous work in our assumptions about the input distribution: denoting an instance I by (z_1, \dots, z_n) , for each i , z_i is a point in the plane drawn independently from a point distribution \mathcal{D}_i . Thus, the distribution \mathcal{D} of the instances has the form $\prod_i \mathcal{D}_i$. For any constant $\varepsilon \in (0, 1)$, our algorithm can be set to have a learning phase of n^ε rounds, during which it constructs data structures of $n^{1+\varepsilon}$ total space. After that, we reach the limiting phase where our algorithm will have an expected running time of $O(\varepsilon^{-1}(n + H(\text{conv}(I))))$ per instance. By results in [2], this time-space tradeoff is optimal.

Prior algorithms. With a plethora of planar convex hull algorithms available, it might seem that there could already be one that is entropy-optimal like ours. A single example shows that for several prior algorithmic approaches, this is not so. Please see the first example in Figure 1. The input points in the example are in two groups: a lower group L , that is not on the upper hull, and an upper group U , arranged so that all points in U are vertices of the upper hull. The sets L and U have the same number of points $n/2$. Suppose that the input distribution \mathcal{D} takes the following form: the points z_1 through $z_{n/2}$ take fixed positions to form U , and for z_i with $i > n/2$, a random point of L is chosen to be z_i (some points of L may be chosen in this way more than once). Thus the “lower” points are a randomly per-

mutated subset of L , with the number of distinct points $\Omega(n)$. The output convex hull will always have vertex set U , while the points of L are all shown to be non-extremal because they are below the line segment with endpoints z_1 and $z_{n/2}$. Thus the entropy of the output convex hull is $O(n)$ (in fact, it is zero), and this bounds the running time of our algorithm in the limiting phase, since we assume that $\Omega(n)$ time is needed to write the output.

However, in several other algorithmic models, this example needs $\Omega(n \log n)$ time: Since the output size is $n/2$, an output-sensitive algorithm requires $\Omega(n \log n)$. This implies that the *structural entropy* introduced by Barbay [5] is here $\Omega(n \log n)$. Since the expected number of upper hull vertices of a random subset of $U \cup L$ is $r/2$, a randomized incremental algorithm takes $\Theta(n \log n)$ time to compute $\text{conv}(I)$ [8]. Because computation of the hull takes linear time for points sorted by a coordinate, say the x coordinate, it might be thought that self-improving algorithms for sorting [2] would be helpful; however, since the entropy of the sorting output is $\Omega(n \log n)$, such an algorithm would not give a speedup here. By a similar argument, a self-improving algorithm for Delaunay triangulations will also not help. For this input distribution, the entropy of the output Delaunay triangulations is $\Omega(n \log n)$ (because of the randomness among the points of L).

Afshani et al [1] give *instance optimal algorithms*, that are “optimal” (in certain restricted models) for *instances* of the planar convex hull problem. In their setup, all inputs from this distribution would be considered “difficult” and would be dealt with in $\Omega(n \log n)$ time. The main difference is that they are interested in an optimal algorithm for any given input considered as a *set*, whereas we want an optimal algorithm for an ensemble of inputs from a special distribution, where each input is considered as a *sequence*. Indeed, for our algorithm it is essential to know the identity of each point (ie, to know that z_i is the i -th point). Moreover, we are demanding a more stringent optimality condition, namely, that our algorithm be competitive with *any comparison-based algorithm*.

To conclude, we also mention the paradigm of preprocessing imprecise points for faster (Delaunay) triangulation [6, 10, 13, 14], where we are given a set of planar regions which we would like to preprocess in order to quickly find the (Delaunay) triangulation for any point set which contains exactly one point from each region. This setting is adversarial, but if we only consider point sets where a point is randomly drawn from each region, it can be regarded as a special case of our problem. In this view, these results tell us that if \mathcal{D} draws its points from disjoint planar regions,

our algorithm will eventually achieve linear expected running time.

Why is this hard? Since the convex hull is essentially part of the Delaunay triangulation, generally algorithms for the former are simpler than those for the latter; since a self-improving algorithm for Delaunay triangulation was already known, it seems natural to assume that a planar convex hull algorithm in the same model should follow easily, and be simpler than the Delaunay algorithm.²

As we explained above, this is not true. Let us try to give some more intuition for this. The reason seems to be the split in output status among the points: some points are simply listed as extremal, and others need a certificate of nonextremality; the certificate may be easy to find, or hard to find. In the first example of Figure 1, the certificates of nonextremality are all “easy”. However, if the points of L are placed *just below* the edges of the upper hull, then to find the certificate for a given point in z_i for $i > n/2$, a search must be done to find the single edge of the upper hull that the point is below, and the certificates are “hard”. A simple example shows that even though the points are independent, the convex hull can exhibit very dependent behavior. In the second example of Figure 1, point z_1 can be at either p_h or p_ℓ , but the other points are fixed. The other points become extremal *depending* on the position of z_1 . This makes life rather hard for entropy-optimality, since for $z_1 = p_\ell$ the ordering of the remaining points must be determined, but otherwise that is not necessary.

In our algorithm, and plausibly for any algorithm, some form of point location must be done for each x_i . (Here, one search we do involves dividing the plane by x -coordinate, and searching among the resulting vertical slabs.) If point z_i is “easily” shown to be nonextremal, then the point location search should be correspondingly shallow. However, it doesn’t seem to be possible to determine, in advance, how deep the point location could go: imagine the points L of Figure 1 doubled up and placed at *both* the “hard” and “easy” positions, and z_i for $i > n/2$ chosen randomly from among those positions; the necessary search depth can only be determined using the position of the point. Also, the certificates may be easy to find, if we know which points are extremal, or at least “near extremal,” but determining that is why we are doing the search to begin with.

Basic ideas and outline. To handle the split nature of our concerns, we perform two kinds of searches

for each input point z_i , using trees T_i and A_i . Roughly, the T_i serve to determine if z_i is likely to be extremal, and A_i is used to help find a pair of extreme points a and b such that line segment \overline{ab} is above z_i , certifying that it is not extremal.

Every node of either kind of tree has an associated *vertical slab* of points whose x -coordinates lie in an interval. The slabs of a node’s children partition the slab of the node (up to bounding vertical lines), and so search in a tree corresponds to refining the knowledge of the x -coordinate of the input point. The properties of the system of slabs we use are discussed in §1.2, while the trees, and some of their key properties, are discussed in section 2.

Search proceeds in parallel over all input points and both kinds of trees, and for a given point and tree, the search advances only if some particular conditions are satisfied. For the T_i trees, these conditions hold only if the point is high enough (relative to estimates of the hull) that it cannot be ruled to be nonextremal. For the A_i trees, a given slab has bounding vertical lines determined by extreme points of this instance. The search for point z_i proceeds in A_i only if those associated extreme points have been found, and z_i is above the line segment connecting them. (If z_i is below that line segment, those extreme points give a certificate that it is not extremal, and thus processing for it is complete.)

The main challenge is to avoid that the algorithm makes too many steps in the T_i trees, that is, we need a way to predict whether a point z_i is extremal or not. For this, the algorithm maintains a *cutoff-hull* C , which is a rough guess of what $\text{conv}(I)$ looks like. The T_i search continues only as long as we cannot be sure that z_i is below C . However, we have no one-size-fits-all solution: if we try to use just one C , it might well happen that relevant points of I end up inside C and hence the T_i searches get stuck.³ Therefore, we need a whole sequence of cutoff hulls C_1, C_2, \dots that is specific to a given input I . To achieve this, the algorithm proceeds in rounds, such that during round k the T_i searches are guided by C_k . After round k , we use the information from the T_i searches to construct the next cutoff hull C_{k+1} to advance the searches for the relevant points that were not identified in round k . The cutoff hulls are engineered such that in any round an expected constant fraction of the active T_i searches for relevant points succeeds. This ensures that the number of rounds is $O(\log n)$, and it is crucial in achieving an expected linear overhead for the maintenance of the C_k . Since the C_k are adaptive and maintained dynamically, we

²The authors certainly thought so for awhile, before painfully learning otherwise.

³Indeed, this is what sets convex hulls apart from sorting and Delaunay triangulations; for those, we can get away with using *just one* typical structure for all inputs.

are confronted with an interesting technical challenge: to decide whether to proceed with a T_i search, we need to know the intersections between C_k and the boundaries of the vertical slab for the current position in T_i . More precisely, z_i is “high enough”, that is, the search proceeds, exactly if z_i is above the segment between those intersections. To obtain this information in constant amortized time, we need several tricks from the “Tarjan toolkit”. In particular, we use a carefully balanced bootstrapping scheme and lookup tables for small sets.

After all rounds are complete, the relevant points in I have been identified, and with this information $\text{conv}(I)$ can be found in linear expected time. The specifics of the searches, and the specification of the algorithm, are in section 3. Our running time bound is stated as Theorem 3.1, under the assumption that the T_i and A_i trees are computed in full, that is, with quadratic storage. The learning phase, in which all necessary information is found to construct the data structures, such as T_i and A_i , needed by the algorithm, is described in section 3.1. The main results for the learning phase, relating the number of training instances used and the associated storage, are given in a series of lemmas in this section. Lemma 3.4 relates the slowdown needed to allow the storage to be nearly linear. The subsection just below discusses what a reasonable output for the planar convex hull problem might be, and shows that such outputs can take a certain canonical form with little loss of generality.

1.1 Output Entropy. What does it mean to “output the convex hull”? The convex hull is represented as a sequence of extremal points. For every other point $z \in I$, we certify that z is inside the convex hull by describing z as a convex combination of some other input points, or by supplying some polygon with vertices in I that contains z . The desired output is the ordered list, together with all the certificates of nonextremality. Even though such certificates are not commonly included in the output, any correct algorithm must implicitly compute them, so requiring the enhanced output does not increase the running time artificially.

Let us start by defining the entropy of the output. We output the upper hull $\text{conv}(I)$ as a sequence of points, from left to right. For every other input point $z \in I$, we give two other points⁴ $u, v \in I$, a *witness pair*, such that z is below the line segment \overline{uv} , see Figure 2. We will call this collection a *legal output* for I , and denote the finite set of all legal outputs by Γ . A *labeling*

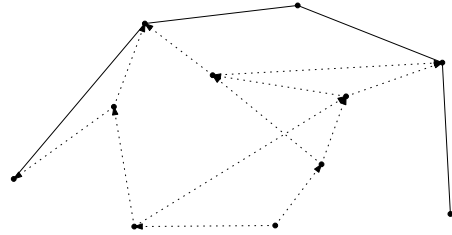


Figure 2: An example output: every nonextremal point indicates two points that span a line segment above it.

scheme \mathcal{L} is a function that maps every input I to some legal output.

Fix an input I . Abusing notation, we will say that for $z \notin \text{conv}(I)$, $\mathcal{L}(z) = (u, v)$, where the segment \overline{uv} is above z . Through the labeling scheme \mathcal{L} , the input distribution \mathcal{D} induces a distribution on Γ . If for each output $\gamma \in \Gamma$ we set $p_\gamma = \Pr[\mathcal{L}(I) = \gamma]$, the entropy of scheme \mathcal{L} is $H(\mathcal{L}) = -\sum_{\gamma \in \Gamma} p_\gamma \log p_\gamma$. Let $\mathcal{M} = \text{argmin}_{\mathcal{L}} H(\mathcal{L})$ be a labeling scheme with minimum entropy, and note that $H(\mathcal{M})$ is a lower bound for *any* comparison-based convex hull algorithm for \mathcal{D} . A labeling scheme \mathcal{M}' is *extremal* if for all $z \notin \text{conv}(I)$, both points of $\mathcal{M}'(z)$ lie on $\text{conv}(I)$. The following lemma implies that without loss of generality, we can assume that the output labeling is extremal. Abusing notation, we will refer to \mathcal{M}' given by the next lemma simply as \mathcal{M} .

LEMMA 1.1. *For any labeling scheme \mathcal{M} , there is an extremal labeling scheme \mathcal{M}' with $H(\mathcal{M}') = H(\mathcal{M}) + O(n)$.*

Proof. We will maintain a directed graph G over the points in I and a stack of points S . A point z is *handled* if we find a pair (u, v) of extremal points such that z is below the segment \overline{uv} . All extremal points are (vacuously) handled, and our aim is to handle all points of I . Observe that if $\mathcal{M}(z) = (u, v)$ and u, v are handled, then z can be handled in $O(1)$ comparisons. We will modify the data structures G and S through a series of operations, until all points are handled.

We first describe the initialization. The stack S is empty. The graph G initially has a vertex for every $z \in I$. If $\mathcal{M}(z) = (u, v)$, we connect z to u and v by a directed edge. In addition, we maintain the set H of handled vertices, initialized to contain all extremal points. We will use point and vertex interchangeably, since they always refer to some $z \in I$. We now explain the operations that are performed on these data structures.

- Operation 1: If a vertex $z \in G$ has both out-

⁴Even if our output provides a set of points such that z is below the upper hull of these points, we can easily convert it to such an output in linear time.

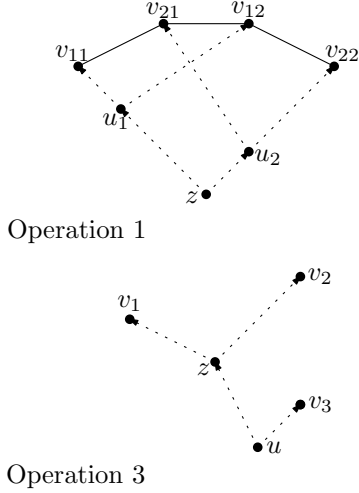


Figure 3: Illustrating Operations 1 and 3: (1) it takes $O(1)$ comparisons to find an extremal witness pair for z once u_1 and u_2 are handled; (2) a witness pair for u that does not include z can be found in $O(1)$ comparisons.

neighbors in H , find an extremal pair (u, v) above z and add z to H . Also delete all edges out of z .

- Operation 2: If $z \in G$ has indegree zero and is not in H , delete it from G and push it onto S .
- Operation 3: Suppose a vertex z has indegree 1 or 2. Then, for each in-neighbor u , find a witness pair that does not include z . Remove all edges going out of u and connect u to this new pair. The indegree of z is now zero.

We state some important properties. Vertices in G either have outdegree 0 or 2. Vertices with outdegree 0 are in the handled set H . For a vertex $z \in H$, the two outneighbors form a line segment above z . A point is either present in G or in S , but never both. When a point z is added to S , then there is a pair (u, v) with $u, v \notin S$ such that z is below the segment \overline{uv} .

We show that each operation takes a constant number of comparisons, see Figure 3. Note that deciding *which* operation to perform needs no comparisons (since it is just a function of G). For Operation 1, vertex z points to (u_1, u_2) , both of which are handled. For each u_i , there is a pair of extremal points (v_{i1}, v_{i2}) above u_i . The upper hull of all v_{ij} 's is above z and has at most 4 points, so we can find an extremal pair above z in $O(1)$ comparisons. Operation 2 requires no comparisons. In Operation 3, consider the in-neighbor u and the upper hull of the out-neighbors of u and z . Since neither u or z is extremal, we can find a pair above u that does not include z in $O(1)$ time. Since the indegree of z is

at most 2, this operation takes $O(1)$ comparisons. It is easy to see that each operation can be performed at most n times, so the total number of comparisons is $O(n)$.

We now argue that if G has at least one edge, one of these operations applies. Suppose Operations 1 and 2 cannot be performed, and there is at least one edge in G . Consider the subgraph G' restricted to the vertices not in H . This subgraph must have an edge. If not, then all edges in G are incident to H . Since all vertices in H have outdegree zero, all edges are directed towards H and thus Operation 1 can be performed. Also, G' has no vertex with indegree zero. Otherwise, that vertex must have indegree zero in G , and Operation 2 can be performed. Therefore, G' has some edges, has no zero indegree vertex, and average outdegree at most 2. There must be a vertex with indegree (in G') 1 or 2. This indegree is the same in G , and Operation 3 applies.

After performing all these operations, we are left with no edges in G , so all points are either in H or on the stack S . Pop the stack to get a point z . There must be a pair above z which is not on the stack, and hence in H . Therefore, z can be handled in $O(1)$ comparisons and is then added to H . We repeat this process until all points are handled, taking $O(n)$ comparisons. Thus, we were able to obtain an extremal labeling scheme \mathcal{M}' from \mathcal{M} with $O(n)$ additional comparisons. Therefore, $H(\mathcal{M}') = H(\mathcal{M}) + O(n)$ by Lemma 1.2. \square

1.2 Slab Structures and Slab Outputs. Define a *slab structure* \mathbf{S} as a sequence of vertical lines partitioning the plane into vertical regions, called *leaf slabs*. A *slab*, more generally, is the union of any contiguous interval of leaf slabs. Given a slab S and $\text{conv}(I)$, we define the segment *induced* by S , $\text{seg}(S)$. Let the left boundary of S be the line L and right boundary R . Consider the point z_L of $\text{conv}(I)$ that is the rightmost point to the left of L . Analogously, define z_R . The line segment $\overline{z_L z_R}$ is $\text{seg}(S)$. Thus, if a point $z \in S$ is below $\text{seg}(S)$, it is below $\text{conv}(I)$, and $\text{seg}(S)$ is the “highest” such line segment with both endpoints outside S .

Given input I , we now define a *slab output*: for every point $z \in \text{conv}(I)$, we indicate that it is indeed extremal and provide the leaf slab it lies in. For every other input point z , we indicate that it is not extremal and either (i) provide the leaf slab that contains z , or (ii) output a slab S such that $\text{seg}(S)$ is above z , witnessing that z is not on $\text{conv}(I)$. Again, there are many possible slab outputs for an input I , and a *slab labeling scheme* is a function that maps inputs to slab outputs.

We also have an *angular slab structure* \mathbf{A} . For a direction v with a nonnegative y -component, we let e_v denote the left-most vertex of $\text{conv}(I)$ that is extremal

in direction v . In other words, the line through e_v normal to v has I on one side. Since we consider upper hulls, this point is unique, for a given v . Two points of $\text{conv}(I)$ define a slab by the vertical lines that pass through them. The angular slab structure \mathbf{A} consists of a sequence v_1, v_2, \dots, v_m , where v_1 is the negative x -direction, v_m is the positive x -direction, and v_i is rotated clockwise from v_{i-1} for all $i > 1$. We will set $m = O(n)$. A point is present in angular slab $S = [v, v']$ if it is inside the vertical slab defined by $e_v, e_{v'}$. We denote the line segment $\overline{e_v e_{v'}}$ by $\text{seg}(S)$. A point is *above* an angular slab S if it is present in S and above $\text{seg}(S)$. An angular leaf slab is of the form $S^L = [v_i, v_{i+1}]$. We define an *angular slab output*: for every point $z \notin \text{conv}(I)$, we provide a leaf slab containing z or a slab S such that $\text{seg}(S)$ is above z . Just as above, we have *angular slab labeling schemes*. Our main tool to handle these entropies is an information-theoretic lemma that relates entropies algorithmically [2, Claim 2.7].

LEMMA 1.2. *Let \mathcal{D} be a distribution on a universe \mathcal{U} , and let $X : \mathcal{U} \rightarrow \mathcal{X}$ and $Y : \mathcal{U} \rightarrow \mathcal{Y}$ be two random variables. Suppose that the function $f : \mathcal{U} \times X(\mathcal{U}) \rightarrow \mathcal{Y}$ defined by $f : (I, X(I)) \mapsto Y(I)$ can be computed with $O(n)$ expected comparisons (where the expectation is over \mathcal{D}). Then $H(Y) = O(n + H(X))$, where all the entropies are with respect to \mathcal{D} .*

We have a claim about the entropy of slab labeling schemes.

CLAIM 1.1. *For every extremal labeling scheme \mathcal{M} , there is a slab labeling scheme \mathcal{S} such that $H(\mathcal{S}) = O(H(\mathcal{M}) + n)$. Analogously, there is an angular slab labeling scheme \mathcal{A} such that $H(\mathcal{A}) = O(H(\mathcal{M}) + n)$.*

Proof. We will show that given $\mathcal{M}(I)$, using a linear number of comparisons, we can find a slab output for I . Note that we immediately have the labeling of points as extremal and nonextremal. Furthermore, by merging the sequence $\text{conv}(I)$ with the list of slab boundaries, we can determine the leaf slab for every point of $\text{conv}(I)$ in $O(n)$ comparisons.

For $z \notin \text{conv}(I)$, $\mathcal{M}(z) = (c, d)$ with $c, d \in \text{conv}(I)$. The point z either belongs to the same leaf slab as c or d , or to the slab S between these leaf slabs, in which case $\text{seg}(S)$ is above z . This can be determined in $O(1)$ comparisons, so with $O(n)$ comparisons we can provide a slab output. The proof for \mathcal{A} is almost identical. We are given $\mathcal{M}(I)$. In $O(n)$ time, we can find e_v , for all $v \in V := \{v_1, \dots, v_m\}$. Furthermore, for any point in $\text{conv}(I)$ that is not extremal for any $v \in V$, we can find an angular leaf slab containing it. For $z \notin \text{conv}(I)$,

$\mathcal{M}(z) = (c, d)$ where both are in $\text{conv}(I)$. The point z either belongs to the same angular leaf slab as c or d , or to the angular slab S between them, in which case $\text{seg}(S)$ is above z . This can be determined in $O(1)$ comparisons, and therefore, with $O(n)$ comparisons we can provide an angular slab output. \square

2 Search trees and convenient forms of the lower bound

We will now interpret the information-theoretic lower bound $H(\mathcal{M})$ algorithmically by describing some fictitious procedures whose running times are $O(H(\mathcal{M}))$. The running times can be expressed in a simpler form, which we use later to bound the running time of our actual algorithm.

Our construction will make crucial use of optimal search trees for \mathbf{S} and \mathbf{A} . For each distribution \mathcal{D}_i , we have a search tree T_i over \mathbf{S} , which is roughly the optimal search tree for \mathcal{D}_i over \mathbf{S} . Analogously, we have search trees A_i over \mathcal{A} for each \mathcal{D}_i . Since $\text{conv}(I)$ can depend on z_i , the same point (at the same location) may lie in different angular slabs depending on the rest of I . Thus, searching in A_i requires some knowledge of $\text{conv}(I)$: to decide which child slab $[v, v']$ contains a point, we must know e_v and $e_{v'}$ for I .

For a slab S , let $p(i, S)$ be the probability that z_i lies in S . We construct T_i based on these probabilities. For an angular slab S , let $p_A(i, S)$ be the probability that z_i belongs in angular slab S *conditioned on* $z_i \notin \text{conv}(I)$. Using the $p_A(i, S)$, we construct the A_i trees. For both kinds of trees, the probability of a search reaching an internal node of depth k is at most μ^k , for some fixed $\mu \in (0, 1)$. When we use the term slab, we are usually referring to a slab in \mathbf{S} . To reduce confusion, we will always use the phrase angular slab for slabs of \mathbf{A} .

We construct T_i recursively as follows: suppose that the slab S is contained in the tree T_i . If possible, we split into children slabs S_1, S_2 such that the conditional probabilities $p(i, S_1)/p(i, S), p(i, S_2)/p(i, S)$ are both in $[1/3, 2/3]$. Otherwise there is a leaf slab λ in S with $p(i, \lambda)/p(i, S) \geq 1/3$, and we split S into three contiguous children S_1, λ, S_2 in the natural way, so that the conditional probabilities for all children are bounded by $2/3$.

Consider the following procedure to compute the convex hull of input I . It will not be implemented, but serves as a thought experiment to find convenient forms of the lower bound $H(\mathcal{M})$. Suppose an oracle tells us the points of $\text{conv}(I)$, in no particular order, free of charge. For each such point z_i , we use T_i to locate it in a leaf slab. Next, we sort the points within each leaf slab by x -coordinate and compute $\text{conv}(I)$. By our assumptions about the number of points in

each leaf slab, this takes $O(n)$ time. For each point $z \notin \text{conv}(I)$, we search for it in T_i . Upon reaching a slab S , we check whether $\text{seg}(S)$ is above z_i . If so, we terminate the search, having certified that z_i is nonextremal. Otherwise, we continue. If the search reaches a leaf slab λ without certification occurring, we just stop the search. Note that this procedure computes a valid slab output for I : we have $\text{conv}(I)$ in sorted order. For every nonextremal point, we have either certified that it is not extremal or placed it in a leaf slab.

CLAIM 2.1. *For any slab labeling scheme \mathcal{S} , the expected time for this procedure is $O(H(\mathcal{S}) + n)$.*

Before proving this claim, we need a technical claim about the searches in T_i . Fix a slab S . We will consider a new distribution \mathcal{D}'_i with the following behavior: for each leaf slab $\lambda \in S$, choose some $0 \leq p'(i, \lambda) \leq p(i, \lambda)$. For every other leaf slab, set $p'(i, \lambda) = 0$. Define $p'(i, S) = \sum_{\lambda \in S} p'(i, \lambda)$. Then the probability according to \mathcal{D}'_i of being in a slab $R \subseteq S$ is $p'(i, R)/p'(i, S)$. Let a search for a point z_i (using T_i) be S -restricted if it terminates once we locate z_i in any slab contained in S .

CLAIM 2.2. *Given any contiguous slab S , conditioned on z_i drawn from \mathcal{D}'_i , the expected time for an S -restricted search in T_i is $O(-\log p'(i, S) + 1)$.*

Proof. (of Claim 2.1) The entropy $H(\mathcal{S})$ is given as $-\sum_{\gamma} p_{\gamma} \log p_{\gamma}$, where the sum is over all slab outputs γ , and p_{γ} is the probability that $\mathcal{S}(I) = \gamma$. Consider some slab output γ : this is a list S_1, \dots, S_n such that for all i , the point z_i lies in S_i . Take a nonextremal point z_i . As soon as the tree T_i locates z_i in any slab $S \subseteq S_i$, the search for z_i will stop (because $\text{seg}(S)$ witnesses the nonextremality of z_i). In other words, an S_i -restricted search for z_i in T_i is sufficient for processing z_i .

Consider the set $\mathcal{I}_{\gamma} = \mathcal{S}^{-1}(\gamma)$ (note that this is not necessarily a discrete set). For every $I \in \mathcal{I}_{\gamma}$, the nonextremal point $z_i \in I$ lies in S_i , and by definition, $\Pr[\mathcal{I}_{\gamma}] = p_{\gamma}$. Let us now restrict our attention to the point z_i . The inputs in \mathcal{I}_{γ} give us a set of locations for z_i ; let \mathcal{E}_i denote the event that z_i lies on one of these locations. Note that \mathcal{E}_i implies that $z_i \in S_i$, but not the other way around. Obviously, for any slab S we have $\Pr[\mathcal{E}_i \wedge z_i \in S] \leq p(i, S)$, so the distribution \mathcal{D}'_i , obtained by conditioning \mathcal{D}_i on \mathcal{E}_i satisfies the requirements of Claim 2.2. By the independence of the \mathcal{D}'_i 's, the \mathcal{E}_i 's are independent events, so $p_{\gamma} \leq \Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \dots \wedge \mathcal{E}_n] = \prod_i \Pr[\mathcal{E}_i]$.

By Claim 2.2, the expected S_i -restricted search time for z_i (conditioned on \mathcal{E}_i) is $O(-\log \Pr[\mathcal{E}_i] + 1)$ and the total expected running time for slab output

γ is $O(-\sum_i \log \Pr[\mathcal{E}_i] + n)$. Hence, the expected total running time of our procedure is proportional to $\sum_{\gamma} p_{\gamma} (-\sum_i \log \Pr[\mathcal{E}_i] + n) \leq \sum_{\gamma} p_{\gamma} (-\log p_{\gamma} + n) \leq H(\mathcal{S}) + n$. \square

We now prove Claim 2.2.

Proof. In this proof, we only deal with z_i , so all probabilities p' are with respect to that. For a tree slab A , we will write p'_A for $p'(i, A)$, and p_A for $p(i, A)$.

The work is proportional to the number of nodes visited, so we will count the latter. We will prove, by induction on the distance from a leaf, that for all visited nodes A with $p_A \leq 1/e$, the expected number of visited nodes below A is $C_1 + C \log(p_A/p'_A)$, for a constant C . This bound clearly holds for leaves. Moreover, since for A at depth k , $p_A \leq \mu^k$, we have $p_A \leq 1/e$ for all but the root and at most $\log(e)/\log(1/\mu)$ nodes below it on the search path.

It will be helpful to consider the possible ways that S can intersect the children (slabs) of nodes that are examined in a search.

Say that the intersection $S \cap A$ of S with tree slab A is *trivial* if it is either empty, A , or S . Say that it is *anchored* if it shares at least one boundary line with A . If S has nontrivial intersection with any of the child slabs of A , its intersection with those children is anchored. Moreover, if $S \cap A$ is anchored, then there is at most one nontrivial intersection with the child slabs, and that intersection is anchored, see Figure 4. Thus, in considering the nodes visited during a search of T_i , starting at the root, the intersections of S with the children of each visited node are all trivial for an initial sequence of nodes, with S entirely contained in one of the child slabs. Then there may be a node R with two nontrivial intersections, and below R , for each visited node there is at most one nontrivial, anchored intersection of S with the child slabs of the node.

From the above, it follows that for all examined nodes A with $A \neq R$, there is at most one child slab B whose intersection with S is neither empty nor B . Let $W(A)$ be the expected number of nodes examined below A , conditioned on A being visited. We have $W(A) \leq 1 + W(B)p'_B/p'_A$, using the fact that when a search for z_i shows that it is contained in a node $B \subset S$, the S -restricted search is complete.

CLAIM 2.3. *For A, B as above, with $p_A \leq 1/e$, if $W(B) \leq C_1 + C \log(p_B/p'_B)$, then for $C \geq C_1/\log(1/\mu)$, with $\mu \in (0, 1)$,*

$$(2.1) \quad W(A) \leq 1 + C \log(p_A/p'_A).$$

Proof. By hypothesis, using $p_B \leq \mu p_A$, and letting

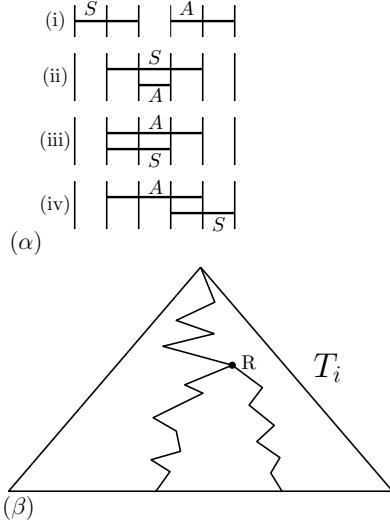


Figure 4: (α) The intersections $S \cap A$ in (i)-(iii) are trivial, the intersections in (iii),(iv) are anchored; (β) every node of T_i has at most one non-trivial child, except for R .

$\beta := p'_B/p'_A \leq 1$, $W(A)$ is no more than

$$\begin{aligned}
& 1 + (C_1 + C \log(p_B/p'_B))p'_B/p'_A \\
& \leq 1 + (C_1 + C(\log(p_A/p'_B) + \log(\mu)))\beta \\
& = 1 + C_1\beta + C \log(p_A)\beta \\
& \quad + C \log(1/p'_B)\beta + C \log(\mu)\beta.
\end{aligned}$$

Noting that $p'_B \log(1/p'_B) \leq p'_A \log(1/p'_A)$ for $p'_A \leq p_A \leq 1/e$, and using $\beta \leq 1$, we have

$$\begin{aligned}
W(A) & \leq 1 + C_1\beta + C \log(p_A) \\
& \quad + C \log(1/p'_A) + C \log(\mu)\beta \\
& = 1 + C \log(p_A/p'_A) + \beta(C_1 + C \log(\mu)) \\
& \leq 1 + C \log(p_A/p'_A),
\end{aligned}$$

for $C \geq C_1/\log(1/\mu)$. \square

Only a slightly weaker statement can be made for the node R having two nontrivial intersections at child nodes R_1 and R_2 .

CLAIM 2.4. *For R, R_1, R_2 as above, if $W(R_i) \leq C_1 + C \log(p_{R_i}/p'_{R_i})$, for $i = 1, 2$, then for $C \geq C_1/\log(1/\mu)$,*

$$W(R) \leq 1 + C \log(p_R/p'_R) + C \log(2).$$

Proof. We have

$$W(R) \leq 1 + W(R_1)p'_{R_1}/p'_R + W(R_2)p'_{R_2}/p'_R.$$

Let $\beta := (p'_{R_1} + p'_{R_2})/p'_R$. With the given bounds for $W(R_i)$, then using $p_{R_i} \leq \mu p_R$, $W(R)$ is bounded by

$$\begin{aligned}
& 1 + \sum_{i=1,2} [C_1 + C \log(p_{R_i}/p'_{R_i})]p'_{R_i}/p'_R \\
& \leq 1 + C_1\beta + C\beta \log(\mu) + C\beta \log(p_R) \\
& \quad + C \sum_{i=1,2} (p'_{R_i}/p'_R) \log(1/p'_{R_i}).
\end{aligned}$$

The sum takes its maximum value when each $p'_{R_i} = p'_R/2$, yielding

$$\begin{aligned}
W(R) & \leq 1 + C_1\beta + C\beta \log(\mu) + C\beta \log(p_R) + C\beta \log(2/p'_R) \\
& \leq 1 + C \log(p_R/p'_R) + \beta(C_1 + C \log(\mu)) + C \log(2) \\
& \leq 1 + C \log(p_R/p'_R) + C \log(2),
\end{aligned}$$

for $C \geq C_1/\log(1/\mu)$, as in (2.1), except for the addition of $C \log 2$. \square

Now to complete the proof of Claim 2.2. For the visited nodes below R , we may inductively take $C_1 = 1$ and $C = 1/\log(1/\mu)$, using Claim 2.3. The hypothesis of Claim 2.4 then holds for R . For the visited node just above R , we may apply Claim 2.3 with $C_1 = 1 + \log(2)/\log(1/\mu)$ and $C \geq C_1/\log(1/\mu)$. The result is that for the node A just above R , $W(A) \leq 1 + C \log(p_1/p'_A)$. This bound holds then inductively (with the given value of C) for nodes further up the tree, at least up until the $1 + \log(e)/\log(1/\mu)$ top nodes. Thus the expected number of visited nodes below the root T is at most $\log(e)/\log(1/\mu) + 1 + C \log(p_T/p'_T) = O(1 - \log(p'_S))$, as desired. \square

We will now express the running time of the procedure in a more convenient form. For $z_i \in I$ and slab S , let $\text{ex}(i, S)$ be the probability that the z_i lies in S and is above $\text{seg}(S)$.

CLAIM 2.5. *The expected running time of the fictitious procedure is at least $\sum_i \sum_{S \in T_i} \text{ex}(i, S)$.*

Proof. This follows from linearity of expectation. Let $\chi(i, S)$ be the indicator variable of the event z_i being in S and above $\text{seg}(S)$. The running time of the procedure is $\sum_i \sum_{S \in T_i} \chi(i, S) + n$. \square

Combining Claims 2.1 and 2.5, we get that -

LEMMA 2.1. $\sum_i \sum_{S \in T_i} \text{ex}(i, S) = O(H(\mathcal{M}) + n)$

We now prove a similar claim based on a procedure that uses the A_i trees. For point z_i and slab S , let $\text{ex}_A(i, S)$ be the conditional probability that the z_i lies in angular slab S and is above $\text{seg}(S)$, conditioned on $z_i \notin \text{conv}(I)$. Analogous to Lemma 2.1, we get:

LEMMA 2.2. $\sum_i \sum_{S \in T_i} ex_A(i, S) = O(H(\mathcal{M}) + n)$

We assume that $\text{conv}(I)$ is known, as well as e_v for $v \in \{v_1, \dots, v_m\}$. For every leaf slab λ , we also have an ordered list of the extremal points in λ . This procedure searches for every *nonextremal* point z_i in its respective tree A_i . As before, when z_i is in angular slab S , we check if z_i is above $\text{seg}(S)$. If not, we have witnessed that z_i is nonextremal and end the search. Otherwise, we continue. If we reach a leaf slab, then we stop.

By a proof identical to that of Claim 2.2, an analogous claim holds for angular slabs, using the $p_A(i, S)$ probabilities instead. We fix an angular slab S . For every angular leaf slab $\lambda \in S$, choose some $0 \leq p'(i, \lambda) \leq p_A(i, \lambda)$. The distribution \mathcal{D}'_i is now defined as in the T_i case.

CLAIM 2.6. *Given any contiguous angular slab S , conditioned on z_i drawn from \mathcal{D}'_i , the expected time for an S -restricted search in A_i is $O(-\log p'(i, S) + 1)$.*

We are now ready to connect the running time of this procedure to the entropy lower bound.

CLAIM 2.7. *The expected running time of this procedure is $O(H(\mathcal{A}) + n)$.*

Proof. This proof is very similar to the one of Claim 2.1. The entropy $H(\mathcal{A})$ is $-\sum_\gamma p_\gamma \log p_\gamma$, summing over all angular slab outputs γ . For every nonextremal z_i , the output γ maps it to an angular slab S_i . Either S_i is an angular leaf slab or z_i is below $\text{seg}(S_i)$. If A_i locates z_i in any angular slab S contained in S_i , then the search for z_i ends. It suffices to look at S_i -restricted searches for z_i .

As before, consider the set $\mathcal{I}_\gamma = \mathcal{A}^{-1}(\gamma)$. Note that for all $I \in \mathcal{I}_\gamma$, the set N of indices of nonextremal points is fixed. For $i \in N$, let \mathcal{E}_i be the event that $z_i \in S_i$ and that the indices of the nonextremal points are N . The events \mathcal{E}_i and \mathcal{E}_j are independent, since when z_i is not extremal, its position does not affect $\text{conv}(I)$ and the angular slab that contains z_j . Hence, $p_\gamma \leq \Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \dots \wedge \mathcal{E}_n] = \prod_i \Pr[\mathcal{E}_i]$. For an angular slab S we clearly have $\Pr[\mathcal{E}_i \wedge z_i \in S] \leq p_A(i, S)$, the latter being the probability that $z_i \in S$, conditioned on $z_i \notin \text{conv}(I)$. By Claim 2.6, the expected S_i -restricted search time in A_i , conditioned on \mathcal{E}_i is $O(-\log \Pr[\mathcal{E}_i] + 1)$. The total expected running time is asymptotically

$$\begin{aligned} & \sum_\gamma p_\gamma \left(- \sum_i \log \Pr(\mathcal{E}_i) + n \right) \\ & \leq \sum_\gamma p_\gamma (-\log p_\gamma + n) \leq H(\mathcal{A}) + n \end{aligned}$$

□

3 Algorithm and analysis

3.1 Learning phase. During the learning phase, our aim is to learn the slab structures \mathbf{S} , \mathbf{A} , a set of special lines ℓ_{ak} , and all the search trees. We fix some constant $\varepsilon > 0$. This will take place for the first n^ε rounds and take a total space of $n^{1+\varepsilon}$. We give the properties of these structures and defer the proofs of these lemmas to Section 4.

LEMMA 3.1. *For a leaf slab λ of \mathbf{S} , let X_λ be the number of points of a random I that fall in λ . With probability $> 1 - n^{-3}$ over the construction of \mathbf{S} , for every leaf slab λ of \mathbf{S} , $\mathbf{E}[X_\lambda^2] = O(1)$. The list \mathbf{S} can be constructed after $O(\log n)$ rounds in $O(n \log n)$ time.*

LEMMA 3.2. *For a leaf slab λ of \mathbf{A} , let X_λ be the number of points of a random I above λ . With probability $> 1 - n^{-3}$ over the construction of \mathbf{A} , $\mathbf{E}[\sum_\lambda X_\lambda \log(X_\lambda + 1)] = O(n)$. The list \mathbf{A} can be constructed after $O(\log^2 n)$ rounds in $O(n \text{poly}(\log n))$ time.*

We also need a special set of lines to construct the cutoff hulls. Let $V = \{v_1, \dots, v_m\}$ be the directions of \mathbf{A} , and $\delta < 1$ a constant. Let $\varepsilon \in (0, 1)$ be any constant of our choice.

LEMMA 3.3. *In $n^\varepsilon \log n$ rounds and $n^{1+O(\varepsilon)}$ time, we can find lines ℓ_{ak} , for $v_a \in V$, and $k \leq \varepsilon \log n$, such that, with probability $> 1 - n^{-3}$: (i) the lines ℓ_{ak} are normal to direction v_a ; (ii) denote by ℓ^+ the halfplane above ℓ and by ℓ^- the halfplane below ℓ . Then,*

$$(3.2) \quad \mathbf{E}[|I \cap \ell_{a1}^+|] \in [\delta, 2\delta] \text{ and for all } k \geq 1 : \\ \mathbf{E}[|I \cap \ell_{a(k+1)}^+| \mid I \cap \ell_{ak}^+ = \emptyset] \in [\delta, 2\delta].$$

Our final lemma about the T_i and A_i trees is a little different. In the rest of the paper, we will just assume that we have the exact T_i and A_i trees, as discussed before. Note that in reality, this could take $O(n^2)$ space to store.⁵ Based on techniques in [2], we can adapt our algorithm to use approximate *smaller* trees, which are good enough for our purposes. We get a time-space tradeoff (shown to be optimal in [2]) using these smaller trees.

LEMMA 3.4. *In n^ε rounds and $n^{1+O(\varepsilon)}$ time, we can find approximate T_i and A_i trees of size n^ε . The algorithm can be adapted to these trees, having expected running time $O(\varepsilon^{-1}(H(\text{conv}(I)) + n))$.*

⁵Also, approximate versions of the full T_i trees could take $\Omega(n)$ rounds to learn.

3.2 Description of the algorithm. We have slab structures \mathbf{S} and \mathbf{A} with $O(n)$ leaf slabs. The algorithm tries to locate each point z_i in \mathbf{S} and \mathbf{A} , using the trees T_i and A_i . Ideally, we would like to first identify the extremal points, and then use them to prove nonextremality for those that remain. However, we cannot know *a priori* which points are extremal, so the challenge is to distinguish between important searches and those that can wait. A point z_i is witnessed to be nonextremal if we find a segment above z_i joining two other input points. The T_i 's help find the extremal points, and the A_i 's provide witnesses for the nonextremal ones.

The algorithm proceeds in *rounds*, each with several steps. At the beginning of round k , we have a set $V_k \subseteq V$ of *active* directions, a set $Q_k \subseteq I$ of *points under consideration*, and a *cutoff hull* $C_k = \bigcap_{v_a \in V_k} \ell_{ak}^-$. At the end of the round, the algorithm finds a set X_k that it processes. We set $X_{\leq k} = \bigcup_{l=1}^k X_l$. Initially, we set $V_1 = V$ and $Q_1 = I$. For notational convenience, we define $C_0 = \mathbb{R}^2$, $X_0 = \emptyset$, and $\ell_{a0}^+ = \ell_{a0}^- = \emptyset$, for $v_a \in V$. Furthermore, for every $z_i \in Q_k$, we maintain its *current locations* S_i and S_i^A in the trees T_i and A_i , respectively. The current locations are initialized to the roots of the respective trees. Moreover, each active direction $v_a \in V_k$ stores an *extreme candidate* e'_a , initially set to the origin.⁶ There are two invariants that hold before each round k :

(i) For every $v_a \in V \setminus V_k$, the extreme point e_a in direction v_a has been found.

(ii) Let ℓ_{ak} be active and a point z_i be above it. Either the search for z_i in T_i terminates by the end of the k th round, or the algorithm has already found a point (in $X_{\leq k-1}$) that is more extreme than z_i in the direction v_a .

The steps of round k are as follows:

Step 1. Compute the cutoff hull C_k .

Step 2. For every $z_i \in Q_k$, advance S_i to the next level of the tree. Now we have a test to see if the search will proceed further. Let L and R be the left and right boundaries of S_i . Let the line that supports the edge of C_k intersecting L (resp. R) be ℓ_{ak} (resp. ℓ_{bk}). First, determine if $z_i \in \ell_{ak}^+$ or $z_i \in \ell_{bk}^+$. If yes, put z_i into a list L_a or L_b , for the corresponding direction, and stop the search. Determine the point u_L where C_k intersects L . Find the edge of $\text{conv}(X_{\leq k-1})$ that intersects L and let w_L be its left endpoint. Analogously, find u_R and w_R . Determine $\text{conv}\{u_L, u_R, w_L, w_R\}$, and denote its highest edge by $\text{seg}_k(S_i)$. Note that $\text{seg}_k(S_i)$ straddles S_i completely. If z_i is below $\text{seg}_k(S_i)$, stop the search. Otherwise, advance S_i . Stop when the search reaches a

leaf. Let $X_k \subseteq Q_k$ be the points that have reached a leaf slab.

Step 3. For every $z_i \in Q_k \setminus X_k$, advance its current location in A_i : if the extreme points for the directions corresponding to the children of S_i^A have been found, proceed to the next levels of A_i and T_i . If this witnesses the nonextremality of z_i , stop, otherwise repeat until either (i) not all extreme points for the children of S_i^A have been found; (ii) z_i is witnessed to be nonextremal; (iii) the search reaches a leaf in A_i ; or (iv) the search reaches a leaf in T_i . Let Z_k be the points for which (ii) holds, Y_k be the points for which (iii) holds, and add all the points for which we have (iv) to X_k .

Step 4. For every point z_i in a list L_a , traverse C_k to find all edges of C_k that are visible from z_i . Then, for each edge of C_k , find the most extreme point e''_a that can see it. Compute $\text{conv}(X_k)$. Then, by simultaneously traversing $\text{conv}(X_k)$ and V_k , find the extreme points $e_a^{(k)}$ for $v_a \in V_k$ in X_k . For each $v_a \in V_k$, update the extreme candidate e'_a by taking the more extreme of the old candidate, $e_a^{(k)}$ and e''_a . If $e'_a \in \ell_{ak}^+$, set $e_a = e'_a$. Let $V_k^{(f)}$ be the directions for which e_a has thus been determined, let X'_k be the points in the lists L_a that are extreme for a direction $V_k^{(f)}$, and let Y'_k be rest of these points. For all points in X'_k , finish the T_i search and add them to X_k . Recompute $\text{conv}(X_k)$, and then use it to find $\text{conv}(X_{\leq k})$.

Step 5. Set $V_{k+1} = V_k \setminus V_k^{(f)}$ and $Q_{k+1} = Q_k \setminus (X_k \cup Y_k \cup Z_k)$.

The rounds end when $k = K := \varepsilon \log n$. After that, try to locate all the points in $\bigcup_k Y'_k$ by simultaneously traversing A_i and T_i for at most $\varepsilon \log n$ steps. Add those points for which the search succeeds to X_k or Y_k , depending on whether they were located in a regular or an angular leaf slab, and add the rest to Q_k . Compute $\text{conv}(Q_K)$ in time $O(|Q_K| \log |Q_K|)$ with a regular algorithm. Then let $X = \bigcup_{k < K} X_k$ and $Y = \bigcup_{k < K} Y_k$, and compute $\text{conv}(X \cup Y)$ in time $O(|X \cup Y|)$ (explained in Lemma 3.9). Obtain $\text{conv}(I)$ by merging the two hulls in $O(n)$ time.

3.3 A few observations. To develop some intuition, let us begin with a few claims. First, we show that the invariants are valid and characterize the active directions. We mention that the T_i -search for point z_i terminates when either z_i is located in a T_i leaf slab or is put in some list L_a .

CLAIM 3.1. *Invariant (ii) is maintained.*

Proof. Suppose that some ℓ_{ck} is active and point z_i is above it. On the other hand, the search in the T_i tree

⁶For readability, we will often write e_a instead of e_{v_a} .

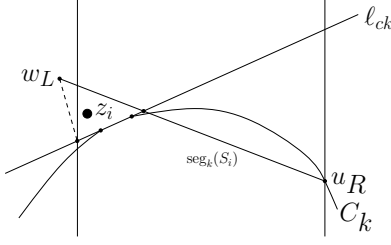


Figure 5: If z_i is in l_{ck}^+ , but below $\text{seg}_k(S_i)$, then w_L must be more extremal in direction v_c than z_i .

for z_i does not terminate. This is because there is some slab $S_i \ni z_i$ such that z_i is below $\text{seg}_k(S_i)$. We will show that there is some point in $X_{\leq k-1}$ that is more extremal than z_i for v_c . We will refer to the notation in Step 2.

First, note that the segment $\text{seg}_k(S_i)$ cannot be $\overline{u_L u_R}$ because then this segment is internal to C_k (and z_i is outside C_k). Suppose that $\text{seg}_k(S_i)$ is $\overline{w_L u_R}$. Then $\text{seg}_k(S_i)$ witnesses the non-extremality of z_i , and for any direction v_a , there is a point in $X_{\leq k-1}$ that is more extremal than z_i . That leaves that case that $\text{seg}_k(S_i)$ is $\overline{w_L u_R}$. Refer to Figure 5. Because l_{ck} supports C_k and because u_R lies on C_k , l_{ck} must intersect L below $\overline{w_L u_R}$ and R above u_R . Consider the triangle Z formed by w_L , $l_{ck} \cap L$, and $l_{ck} \cap \text{seg}_k(S_i)$. Since $z_i \in Z$ and since w_L is the only vertex of Z not on l_{ck} , w_L is more extremal than z_i for v_c . Since $w_L \in X_{\leq k-1}$, we are done. \square

We now consider Invariant (i). For a direction $v_a \in V$, let d_a be the maximum k such that $v_a \in V_k$, the *depth* of v_a .

CLAIM 3.2. *Let $v_a \in V$. We have $d_a = k$ if and only if $I \cap \ell_{a(k-1)}^+ = \emptyset$, but $I \cap \ell_{ak}^+ \neq \emptyset$. Also, the extreme point e_a for direction v_a has been found after round d_a , that is, Invariant (i) holds.*

Proof. Let us first argue the “only if” part. By the test in Step 4, v_a can only become inactive in round $k+1$ if $I \cap \ell_{ak}^+ \neq \emptyset$. Furthermore, if $I \cap \ell_{a(k-1)}^+$ were non-empty, we would get a contradiction as follows: by $d_a = k$, the direction v_a is active in the $(k-1)$ th round. By Invariant (ii), for any point in $I \cap \ell_{a(k-1)}^+$, one of the following happens: it is searched for by the end of the round (and put in X_{k-1}), it is put in some L_c , or there is a point in $X_{\leq k-2}$ that is more extremal. In any case, the extremal point in direction v_a is discovered by the end of this round in Step 4, contradicting $d_a = k$.

For the “if” part, note that by the test in Step 4 and the assumption, v_a has been active in rounds 1 to k . Therefore, by Invariant (ii), we see that $e_a \in X_{\leq k} \cup L_a$, so v_a becomes inactive after round k . Furthermore,

e_a is the most extreme point in the v_a -direction of $\text{conv}(X_1), \dots, \text{conv}(X_k)$, and hence found in Step 4. \square

We can now show that the probability that a direction is active in round k decays exponentially.

CLAIM 3.3. *For any $v_a \in V$, we have $\exp(-4k\delta) \leq \Pr[d_a > k] \leq \exp(-k\delta)$. Thus, $\mathbf{E}[d_a] = O(1)$.*

Proof. Let p_{li} be the probability that $z_i \in \ell_{al}^+$, conditioned on $\ell_{a(l-1)}^+ \cap I = \emptyset$. By (3.2), we have $\sum_i p_{li} \in [\delta, 2\delta]$ for all l , and if we let $\delta < 1/4$, then $p_{li} < 1/2$ for all i, l . By Claim 3.2,

$$\begin{aligned} \Pr[d_a > k] &= \prod_{l=1}^k \Pr[I \cap \ell_{al}^+ = \emptyset \mid I \cap \ell_{a(l-1)}^+ = \emptyset] \\ &= \prod_{l=1}^k \prod_{i=1}^n (1 - p_{li}) \leq \prod_{l=1}^k \exp\left(-\sum_i p_{li}\right) \leq e^{-k\delta}, \end{aligned}$$

and

$$\Pr[d_a > k] = \prod_{l=1}^k \prod_{i=1}^n (1 - p_{li}) \geq \prod_{l=1}^k e^{-2 \sum_i p_{li}} \geq e^{-4k\delta},$$

because $1 - x \geq \exp(-x/(1-x))$ for $x < 1$. Thus,

$$\mathbf{E}[d_a] = 1 + \sum_{k=1}^{\infty} (k+1) \Pr[d_a > k] \leq \sum_{k=0}^{\infty} (k+1) e^{-k\delta},$$

which is $O(1)$. \square

In particular, the number of active directions per round decreases geometrically.

CLAIM 3.4. *We have $\mathbf{E}[\sum_k |V_k|] = O(n)$ and there is a constant $\sigma > 1$ such that $\mathbf{E}[|V_k|] = O(n/\sigma^k)$.*

Proof. By linearity of expectation, double counting, and Claim 3.3, $\mathbf{E}[\sum_k |V_k|] = \sum_{v_a \in V} \mathbf{E}[d_a] = O(n)$. The second statement follows from linearity of expectation, Claim 3.3, and because $v_a \in V_k$ iff $d_a > k-1$. \square

3.4 The data structures. We now describe the data structures needed by the algorithm in Steps 2 and 4. Let us begin with Step 2.

CLAIM 3.5. *There exists a data structure that can answer the following query in constant time: give the edge of C_k that intersects a given slab boundary. The data structure uses linear space and can be maintained in total time $O(n)$.*

Proof. The method works as follows: in each round before Step 2, we first determine for each vertex of C_k the leaf slab of \mathbf{S} that contains it. Using this information, we set up a predecessor searching structure that can find the edge of C_k that intersects a given slab boundary in constant time.

First, we describe how to locate the vertices of C_k in \mathbf{S} . For this proof only, all logarithms are to the base σ , the constant from Claim 3.4. Let t be such that $k \in [3 \log^{(t+1)} n, 3 \log^{(t)} n)$, where $\log^{(t)} n$ denotes the t -th iterated logarithm: $\log^{(0)} n = n$ and $\log^{(t+1)} n = \log(\log^{(t)} n)$. By Claim 3.4, we have

$$(3.3) \quad \mathbf{E}[|C_k|] = O(n/(\log^{(t)} n)^3).$$

Subdivide the leaf slabs of \mathbf{S} into chunks, each having size $\log^{(t-1)} n$. By merging C_k with these chunks, determine the containing chunk for every vertex of C_k in time $O(|C_k| + n/\log^{(t-1)} n)$. Next, use binary search within each chunk to locate the vertices of C_k in the leaf slabs in time $O(|C_k| \log^{(t)} n)$. By (3.3), the total expected overhead is proportional to

$$\begin{aligned} & \sum_{t=1}^{\log^* n} \sum_{k=3 \log^{(t+1)} n}^{3 \log^{(t)} n} \left(\frac{n}{\log^{(t-1)} n} + \mathbf{E}[|C_k|] \log^{(t)} n \right) \\ &= O\left(\sum_{t=1}^{\log^* n} (n \log^{(t)} n / \log^{(t-1)} n + n / \log^{(t)} n) \right) \\ &= O\left(\sum_{t=1}^{\log^* n} n / \log^{(t)} n \right) = O(n), \end{aligned}$$

since

$$\begin{aligned} \sum_{t=1}^{\log^* n} (\log^{(t)} n)^{-1} &= \sum_{t=0}^{\log^* n - 1} \left(\underbrace{\sigma \sigma \cdots \sigma}_t \right)^{-1} \\ &\leq \sum_{t \geq 0} \sigma^{-t} = O(1). \end{aligned}$$

Now, given a slab boundary y , in order to find the intersection of C_k with y it suffices to know the closest leaf slab to the left of y containing a vertex of C_k . Thus, we need a data structure for predecessor searching in a linear-size universe. Our problem resembles a special case of the well-known split-find problem which was studied by Gabow and Tarjan [9].

We adapt their approach to our purposes. The method is based on table lookup. Let \mathcal{U} be a totally ordered set of size $\alpha \log n$, for some constant α . We call it the *representative universe*. Subdivide the leaf slabs into $O(n/\log n)$ consecutive *chunks* $\mathcal{C}_1, \mathcal{C}_2, \dots$ of size $|\mathcal{U}|$ each. The leaf slabs in each chunk are associated with

the elements in \mathcal{U} in the natural way. Furthermore, with every subset $U \subseteq \mathcal{U}$ we associate an *encoding* $\langle U \rangle$, its rank in the lexicographic ordering of subsets. There are three different lookup-tables: (i) the *correspondence table* L_C stores for every chunk \mathcal{C}_j and every element $u \in \mathcal{U}$ the leaf slab in \mathcal{C}_j corresponding to u ; (ii) the *predecessor table* L_P stores for every subset $U \subseteq \mathcal{U}$ and every $u \in \mathcal{U}$ the predecessor of u in U ; and (iii) the *insertion table* L_I stores for every $U \subseteq \mathcal{U}$ and every $u \in U$ the encoding $\langle U \cup u \rangle$. Here, the subsets U are indexed via $\langle U \rangle$ in L_P and L_I . If α is chosen properly, the total size of the tables and the preprocessing time is $O(n)$. Every chunk \mathcal{C}_j stores an encoding $\langle U_j \rangle$, which is set to $\langle \emptyset \rangle$ at the beginning of round k . For every non-empty leaf slab λ , use L_I to update the encoding $\langle U_j \rangle$ to $\langle U_j \cup \lambda \rangle$, where chunk \mathcal{C}_j contains λ . After that, perform a linear scan through the \mathcal{C}_j and link together the non-empty chunks. This takes time $O(|C_k| + n/\log n)$, which yields total linear overhead by Claim 3.4 and the fact that there are $O(\log n)$ rounds.

To find the predecessor of a slab boundary, determine the chunk \mathcal{C}_j that contains it and the corresponding representative element $u \in \mathcal{U}$. Use L_P , u , and $\langle U_{jk} \rangle$ to find u 's predecessor u' in U_{jk} . If u' does not exist, we let u' be the maximum of the preceding non-empty chunk, which can be found by following the link from \mathcal{C}_j . Finally, use L_C to translate u' back into a slab boundary. This takes constant time, as claimed. \square

Let us now discuss Step 4.

CLAIM 3.6. *With $O(n)$ space and time, we can maintain a data structure that does the following: fix a round k , and let $\lambda_1, \dots, \lambda_\alpha$ be the leaf slabs of \mathbf{S} containing points from X_k . Then $\lambda_1, \dots, \lambda_\alpha$ can be sorted according to x -coordinate in time $O(\alpha)$. This data structure allows us to find the hulls $\text{conv}(X_k)$ in total linear time.*

Proof. We use the same predecessor structure as in Claim 3.5. To sort the λ_i , we insert them into the predecessor structure, and then we perform α predecessor queries, starting from the maximum, to find the sorted order. The time bounds follow as in Claim 3.5.

Let us now describe how to find $\text{conv}(X_k)$. We use the first part to sort the leaf slabs that meet X_k in time $O(|X_k|)$, and then we sort the points in each leaf slab individually. This takes total time $O(\sum_k (|X_k| + \sum_\lambda |X_k \cap \lambda|^2))$, where λ ranges over all leaves of \mathbf{S} . By Lemma 3.1 and since the X_k are disjoint,

$$\begin{aligned} \mathbf{E}\left[\sum_k \sum_\lambda |X_k \cap \lambda|^2\right] &= \mathbf{E}\left[\sum_\lambda \sum_k |X_k \cap \lambda|^2\right] \\ &\leq \mathbf{E}\left[\sum_\lambda |I \cap \lambda|^2\right] = \sum_\lambda \mathbf{E}[|I \cap \lambda|^2] = O(n), \end{aligned}$$

so the total expected sorting time is linear. Now, Graham's scan finds $\text{conv}(X_k)$ in $O(|X_k|)$ time. \square

CLAIM 3.7. *There exists a data structure that stores $\text{conv}(X_{\leq k})$ and can answer the following query in constant time: given a slab boundary B , find the edge of $\text{conv}(X_{\leq k})$ that intersects B . The total time to maintain the data structure is $O(n)$.*

Proof. We store a similar structure as in Claim 3.5: subdivide the leaf slabs into $O(n/\log n)$ chunks of size $O(\log n)$. For each chunk, we store a bitmap that represents which leaf slabs in that chunk contain vertices of $\text{conv}(X_{\leq k})$. A chunk is non-empty if it contains at least one vertex of $\text{conv}(X_{\leq k})$. Each chunk has a pointer to the preceding non-empty chunk.

Now suppose we are at the end of round $k+1$ and want to insert X_{k+1} into $\text{conv}(X_{\leq k})$. By Claim 3.6, we already know $\text{conv}(X_{k+1})$. Furthermore, for each point in X_{k+1} , we can find its predecessor and successor (in x -order) in $\text{conv}(X_{\leq k})$ in constant time, using the bitmaps and the links to the next non-empty chunk. Thus, in time $O(|X_{k+1}|)$, we can partition $\text{conv}(X_{k+1})$ into maximal contiguous subhulls U_1, U_2, \dots such that there is no vertex of $\text{conv}(X_{\leq k})$ between any two vertices in a given U_i . Using Graham's scan together with the predecessor/successor information for the first and last vertex in each U_i , we can now merge these hulls into $\text{conv}(X_{\leq k})$ in time $O(|X_{k+1}| + d)$, where d is the number of points that are deleted from $\text{conv}(X_{\leq k})$. When inserting or deleting a point from the hull, we also update the information in the corresponding bitmap, using a table we computed during preprocessing. Finally, we sweep over all the chunks to determine those which are nonempty, and set up the appropriate links. This takes $O(n/\log n)$ time. Since every point is deleted only once, all the X_k 's are disjoint, and there are $O(\log n)$ rounds, the data structure can thus be maintained in total linear time.

To find the intersection with a given slab-boundary, we use a predecessor query just as in Claim 3.5 \square

3.5 Analysis.

LEMMA 3.5. *The total time for Step 1 is $O(n)$.*

Proof. To compute C_k in time $O(|V_k|)$, it suffices to know the sorted order of V_k . From the preprocessing phase we have the order of $V_1 = V$, and this ordering can be maintained in each round by removing the newly inactive directions. This takes total time $O(\sum_k |V_k|) = O(n)$, by Claim 3.4. \square

We introduce some terminology. Consider the beginning of the k th round. In the previous rounds, the

algorithm has found some extremal points e_v and proven them to be extremal by Invariant (i). Call these points *discovered*. The remaining points are *undiscovered*. Points that have been searched down to leaf slabs but not certified as extremal points are also considered undiscovered. Note that all discovered points are in $X_{\leq k-1}$. We stress that we are only considering the situation at the beginning of the k th round.

CLAIM 3.8. *Consider a fixed point z_i , vertical slab S_i , and round k . Let ℓ_{ak} ⁷ be the line supporting the edge of C_k that intersects the left (resp. right) boundary of S_i . Conditioned on $z_i \in S_i$, z_i is not discovered, and z_i is above $\text{seg}_k(S_i)$, the probability that there is an undiscovered point other than z_i above ℓ_{ak} is at most 2δ .*

Proof. We will first simplify the event to help us prove this. Then, we shall add the conditioning. Let us disregard z_i , and assume that the input is $I \setminus \{z_i\}$. We will run the algorithm on this partial input. Fix a direction a . Let E be the event that the line ℓ_{ak} supports the edge of C_k that intersects the left boundary of S_i . We will prove the claim conditioned on E . Then, we shall introduce the remaining conditions.

Let us try to understand the conditioning. Let L denote the left boundary of S_i . Consider the directions v_c such that the line ℓ_{ck} intersects L below ℓ_{ck} . When does E occur? Firstly, there is no point in $\ell_{a(k-1)}^+$ (ie, e_a has not been discovered yet), and secondly, for all v_c , no line ℓ_{ck} is still active. This final condition is equivalent to all points e_c having been discovered by the beginning of the k th round.

The intersection of the above events is equal to E . We let R denote the region between $\ell_{a(k-1)}$ and ℓ_{ak} . All further discussion will be restricted to conditioning on there being no point in $\ell_{a(k-1)}^+$. Fix some $r \neq i$ and consider the event E_r that z_r falls in R and is not discovered. Remember that the point z_i is not considered. We are interested in the probability $\Pr[E_r|E]$. By Bayes' rule,

$$(3.4) \quad \Pr[E_r|E] = \Pr[E|E_r] \Pr[E_r] / \Pr[E]$$

Let us now estimate $\Pr[E|E_r]$. Take a random input I' without the point z_r . We will condition on the choice of these points (so the randomness is only on z_r). These conditional probabilities will be denoted by $\Pr_{I'}$. Run the algorithm, as before, for $k-1$ rounds on I' . We can ask if the event E occurred on I' . Suppose it did not. This implies that some e_c was not discovered (by the k th round). Let us add z_r conditioned on E_r (fixing I'). We know that z_r is not detected as an extremal point

⁷Note that the direction v_a is a random variable.

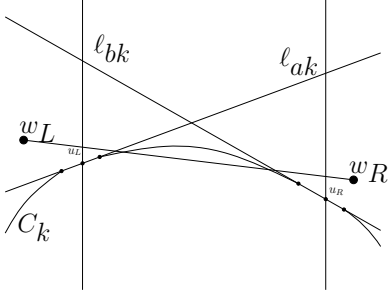


Figure 7: Case 2 of the proof of Claim 3.9

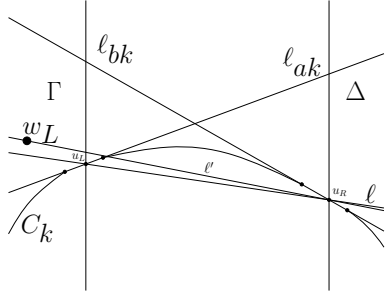


Figure 8: Case 3 of the proof of Claim 3.9

x in Γ . Let the line through x and u_L be ℓ' . If w_L is above ℓ' , that violates the case condition. Therefore, w_L is below ℓ' (and to the left of L). Since both x and w_L are on $\text{conv}(I)$, both endpoints of $\text{seg}(S)$ are below ℓ' . So $\text{seg}(S_i)$ intersects L below u_L , and similarly R below u_R .

Case 2: $\text{seg}_k(S_i)$ is $\overline{w_L w_R}$. This means that u_L is below the intersection of $\overline{w_L w_R}$ with L . The rightmost vertex of $\text{conv}(I)$ to the left of L must be to the right of w_L and above $\overline{w_L w_R}$. This region contains no undiscovered points because it is in ℓ_{ak}^+ . Refer to Figure 7. The point w_L must be the rightmost discovered point, and hence the rightmost point in $\text{conv}(I)$, in this region. Arguing similarly for w_R , we conclude $\text{seg}_k(S_i)$ is exactly $\text{seg}(S_i)$.

Case 3: $\text{seg}_k(S_i)$ is $\overline{w_L u_R}$. Let ℓ be the line through u_L and u_R . The regions Γ and Δ are the same as in Case 1. The region Γ contains the point w_L . Note that w_L is below ℓ_{bk} . Let ℓ' be the line through w_L and u_R . Refer to Figure 8. The rightmost point of $\text{conv}(I)$ to the left of L must be either w_L or below ℓ' . This is because the region to the right of w_L and above ℓ' is in ℓ_{ak}^+ . The portion of ℓ' to the right of R must be in ℓ_{bk}^+ (because $w_L \in \ell_{bk}^-$). Suppose the leftmost point of $\text{conv}(I)$ to the right of R is above ℓ' . Then the point is in ℓ_{bk}^+ and is a discovered point. Therefore, the leftmost *discovered*

point in that region, which is w_R , must be that point. But then $\overline{w_L w_R}$ would be above $\overline{w_L u_R}$, violating the case condition. Therefore, this point is below ℓ' and $\text{seg}(S_i)$ is below ℓ' . \square

LEMMA 3.6. *The expected total time for Step 2 is $O(\sum_k |Q_k| + n + H(\mathcal{M}))$.*

Proof. In each round, Step 2 considers all of Q_k , which explains the $\sum_k |Q_k|$ term. The total number of moves into to L_a lists is at most $O(n)$, since each point can be put into such a list only once. We now bound the total expected number of moves in the T_i 's by $O(n + H(\mathcal{M}))$, which implies the lemma by Claim 3.5. Fix a point z_i with current slab S_i . The search proceeds from S_i when z_i crosses S_i , for some round k . This immediately implies that z_i has not been discovered. Fix such a round k . We want to argue that conditioned on this, the probability that z_i is extremal for S_i is large.

Let ℓ_{ak} and ℓ_{bk} be the supporting lines of C_k that intersect the left and right boundaries (L and R) of S_i . By Claim 3.8 and a union bound, with probability at least $1 - 4\delta$ (conditioned on the behavior of z_i), there is no undiscovered point above either of these lines. Note that by Claims 3.5 and 3.7, the points u_L, u_R, w_L, w_R can be found in constant time. By Claim 3.9, z_i is extremal in S_i with probability more than $1 - 4\delta$ conditioned on $z_i \in S_i$, z_i not being discovered by the k th round, and z_i being above $\text{seg}_k(S_i)$.

The choice of k is arbitrary. Conditioned on $z_i \in S_i$, let \mathcal{E}_1 be the event that z_i is extremal in S_i , and \mathcal{E}_2 the event that the search for z_i crosses S_i . We can conclude that $\Pr[\mathcal{E}_1 | \mathcal{E}_2] \geq 1 - O(\delta) \geq 1/2$, for sufficiently small δ . By Bayes' Rule,

$$\begin{aligned} \Pr[\mathcal{E}_2] &= \frac{\Pr[\mathcal{E}_2]}{\Pr[\mathcal{E}_1 \cap \mathcal{E}_2]} \cdot \frac{\Pr[\mathcal{E}_2 \cap \mathcal{E}_1]}{\Pr[\mathcal{E}_1]} \cdot \Pr[\mathcal{E}_1] \\ &\leq 2 \Pr[\mathcal{E}_2 | \mathcal{E}_1] \Pr[\mathcal{E}_1] \leq 2 \Pr[\mathcal{E}_1]. \end{aligned}$$

Using Lemma 2.1, the total expected number of moves is bounded by

$$\begin{aligned} \sum_i \sum_{S \in T_i} \Pr[\mathcal{E}_2] \Pr[z_i \in S] &\leq 2 \sum_i \sum_{S \in T_i} \Pr[\mathcal{E}_1] \Pr[z_i] \\ &\leq 2 \sum_i \sum_{S \in T_i} \text{ex}(i, S) = O((H(\mathcal{M}) + n)). \end{aligned}$$

\square

LEMMA 3.7. *The expected total time for Step 3 is $O(\sum_k |Q_k| + H(\mathcal{M}) + n)$.*

Proof. We count the moves in the A_i 's. First, consider an extremal point z_i . Since Step 3 moves simultaneously

in A_i and T_i , the number of moves is at most the depth of the leaf slab in T_i . The fictitious procedure of Claim 2.1 searches for all extremal points of the input in T_i , so the claim shows that the total expected number of search moves in Step 3 for extremal points is $O(H(\mathcal{M}) + n)$.

For nonextremal points, the searches correspond to the procedure considered in Claim 2.7, so the number of moves for nonextremal points is $O(H(\mathcal{M}) + n)$. \square

LEMMA 3.8. *We have $\mathbf{E}[\sum_k |Q_k|] = O((H(\mathcal{M}) + n))$.*

Proof. For $z_i \in I$, let r_i be the largest k with $z_i \in Q_k$. By double counting $\sum_k |Q_k| = \sum_i r_i$. We split the sum into two parts. If z_i is extremal, r_i is at most the depth of the leaf slab in T_i that contains z_i , because S_i is advanced at least once per round. As argued in the proof of Lemma 3.7, the total expected time to locate all *extremal* points in their respective leaf slabs is $H(\mathcal{M}) + O(n)$.

Suppose z_i is not extremal. We will focus on the search for z_i in A_i . Note that we cross an angular slab $S^A = [v_a, v_b]$ only after certifying that z_i is extremal for S^A , for which we first need to find both e_a and e_b . We want to bound the number of rounds where z_i has S^A as its current location. Let P^A be S^A 's parent. Let \mathcal{E}_1 be the event that z_i is not extremal, and let \mathcal{E}_2 be the event that z_i is extremal in P^A . The point z_i reaches S^A only if \mathcal{E}_2 holds. The contribution of the rounds where z_i has current location S^A is at most $\mathbf{E}[d_a + d_b \mid \mathcal{E}_1 \cap \mathcal{E}_2] \Pr[\mathcal{E}_1 \cap \mathcal{E}_2]$, using $\max(d_a, d_b) \leq d_a + d_b$. Note that $\mathbf{E}[d_a + d_b \mid \mathcal{E}_1 \cap \mathcal{E}_2] = \mathbf{E}[d_a + d_b \mid \mathcal{E}_1]$, since conditioned on z_i not being on the hull, d_a and d_b are independent of whether z_i is extremal in P^A . By Claim 3.3,

$$\begin{aligned} & \mathbf{E}[d_a + d_b \mid \mathcal{E}_1 \cap \mathcal{E}_2] \Pr[\mathcal{E}_1 \cap \mathcal{E}_2] \\ &= \mathbf{E}[d_a + d_b \mid \mathcal{E}_1] \Pr[\mathcal{E}_1] \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \\ &\leq \mathbf{E}[d_a + d_b] \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] = O(\Pr[\mathcal{E}_2 \mid \mathcal{E}_1]). \end{aligned}$$

It follows that the contribution of z_i in S^A to the total expected running time is $O(\text{ex}_A(i, P^A))$, and the total expected running time can be bounded by $O(\sum_i \sum_{S^A \in A_i} \text{ex}_A(i, S^A))$, which is $O(H(\mathcal{M}) + n)$ by Lemma 2.2. \square

LEMMA 3.9. *The total expected time for Step 4 and to compute $\text{conv}(X \cup Y)$ is $O(n)$.*

Proof. To find $\text{conv}(X \cup Y)$, we sort the points in each leaf slab individually and concatenate the resulting lists. After that, we use Graham's scan. By Lemmas 3.1 and 3.2, this takes linear expected time. By Claim 3.6, the total time to find all the $\text{conv}(X_k)$

in Step 4 is linear. Updating all the extreme candidates takes total time $O(\sum_k (|X_k| + |V_k|)) = O(n)$, by Claim 3.4. Finally, handling all the L_a list is at most proportional to the total conflict size of C_k , ie, it is $O(\sum_k \sum_a \Pr[v_a \in V_k] \mathbf{E}[|I \cap \ell_{ak}^+| \mid v_a \in V_k])$, which is $O(n)$ by (3.2) and Claim 3.4. \square

THEOREM 3.1. *The algorithm computes $\text{conv}(I)$ in expected time $O(H(\mathcal{M}) + n)$.*

Proof. To see correctness, note that $I \setminus (X \cup Y \cup Q_K) = \bigcup_{k < K} Z_k$, all of which have been witnessed to be nonextremal in Step 3. The expected running time for the K phases is $O(H(\mathcal{M}) + n)$ by Lemmas 3.5, 3.6, 3.7, and 3.9. This also covers the time for computing $\text{conv}(Q_K)$, because the points in Q_K have been active for $\varepsilon \log n$ rounds and $|Q_K| \log |Q_K| \leq |Q_K| \log n \leq \varepsilon^{-1} \sum_{k < K} |Q_k| = O(H(\mathcal{M}) + n)$, by Lemma 3.8. Since computing $\text{conv}(X \cup Y)$ takes $O(n)$ time by Lemma 3.9, we can find $\text{conv}(X \cup Y \cup Q_K)$ in time $O(H(\mathcal{M}) + n)$, as claimed. Note that the searches for X'_k and Y'_k do not affect this bound, because we only perform a T_i search for a point in X'_k after it has been proven extremal for a direction v_a , and we only do an A_i -search for a point in Y'_k when it cannot be extremal for a direction. \square

4 Learning phase proofs

Here we provide the deferred proofs for the lemmas stated in Section 3.1.

4.1 Learning S. Learning the vertical slabs **S** is identical to learning the V -lists of [Lemma 2.3] [3]. We repeat the construction and proof for convenience: take the union of the first $k = \log n$ inputs I_1, I_2, \dots, I_k , and sort the x -coordinates of these points in order to obtain a list $x_0, x_1, \dots, x_{nk-1}$. Then, take the n values $x_0, x_k, x_{2k}, \dots, x_{(n-1)k}$. These define the boundaries for **S**. We recall a useful fact [3, Claim 2.4].

CLAIM 4.1. *Let $Z = \sum_i Z_i$ be a sum of nonnegative random variables such that, $Z_i = O(1)$ for all i , $\mathbf{E}[Z] = O(1)$, and for all i, j , $\mathbf{E}[Z_i Z_j] = \mathbf{E}[Z_i] \mathbf{E}[Z_j]$. Then $\mathbf{E}[Z^2] = O(1)$.*

Proof. By linearity of expectation,

$$\begin{aligned} \mathbf{E}[Z^2] &= \mathbf{E}\left[\left(\sum_i Z_i\right)^2\right] = \sum_i \mathbf{E}[Z_i^2] + 2 \sum_{i < j} \mathbf{E}[Z_i Z_j] \\ &= \sum_i \mathbf{E}[Z_i^2] + 2 \sum_{i < j} \mathbf{E}[Z_i] \mathbf{E}[Z_j] \\ &\leq \sum_i O(\mathbf{E}[Z_i]) + \left(\sum_i \mathbf{E}[Z_i]\right)^2 = O(1). \end{aligned}$$

\square

Proof. (of Lemma 3.1) Consider two values x_i, x_j from the original list. Note that all the other $kn - 2$ values are independent of these two points. For every $r \notin \{i, j\}$, let $Y_t^{(r)}$ be the indicator random variable for $x_r \in t := [x_i, x_j]$. Let $Y_t = \sum_r Y_t^{(r)}$. Since the $Y_t^{(r)}$'s are independent, by Chernoff's bound [4], for any $\beta \in (0, 1]$, $\Pr[Y_t \leq (1 - \beta)\mathbf{E}[Y_t]] \leq \exp(-\beta^2\mathbf{E}[Y_t]/2)$. With probability at least $1 - n^{-5}$, if $\mathbf{E}[Y_t] > 12 \log n$, then $Y_t > \log n$. By applying the same argument for any pair x_i, x_j and taking a union bound over all pairs, with probability $> 1 - n^{-3}$ the following holds: for any pair t , if $Y_t \leq \log n$, then $\mathbf{E}[Y_t] \leq 12 \log n$.

For any leaf slab $\lambda = [x_{ak}, x_{(a+1)k}]$, we have $Y_\lambda \leq \log n$. Let $X_\lambda^{(i)}$ be the indicator random variable for the event that $x_i \in_R \mathcal{D}_i$ lies in λ , so that $X_\lambda = \sum_i X_\lambda^{(i)}$. Since $\mathbf{E}[Y_\lambda] \geq (\log n - 2)\mathbf{E}[X_\lambda]$, we get $\mathbf{E}[X_\lambda] = O(1)$. By independence of the \mathcal{D}_i 's, for all i, j , $\mathbf{E}[X_\lambda^{(i)} X_\lambda^{(j)}] = \mathbf{E}[X_\lambda^{(i)}] \mathbf{E}[X_\lambda^{(j)}]$, $\mathbf{E}[X_\lambda^2] = O(1)$, by Claim 4.1. The running time bound follows trivially. \square

4.2 Learning A. First, we set up some useful definitions. We will take $k = \log^2 n$ inputs I_1, I_2, \dots, I_k . For any angle θ_a and instance hull $\text{conv}(I)$, $\text{ext}(I, \theta_a)$ is the extreme point of $\text{conv}(I_j)$ along θ_a (if there are two, we choose the leftmost point). A point $z \in I$ is *above* the pair of directions (θ_a, θ_b) if z is (i) in the angular slab defined by $\text{ext}(I, \theta_a)$ and $\text{ext}(I, \theta_b)$, and (ii) strictly above the segment between these points. We call $\text{ab}(I, \theta_a, \theta_b)$ the number of points in I above θ_a, θ_b and let $\text{tot}(\theta_a, \theta_b) := \sum_j \text{ab}(I, \theta_a, \theta_b)$. From the instances I_1, \dots, I_k , we will construct a list of directions with the following property.

CLAIM 4.2. *Given inputs I_1, I_2, \dots, I_k , we can find a sequence of angles $\gamma_0, \gamma_1, \dots, \gamma_r$ ($r = O(n)$) with $\text{tot}(\gamma_i, \gamma_{i+1}) \leq \log^2 n$. This takes $n \text{poly}(\log n)$ time.*

This is the list of directions V for **A**, which we shortly show to have the right properties. Why the condition of *strictly above* for saying a point is above a slab? This ensures that such a list V exists. Suppose we said a point z_i is above an angular slab (θ_a, θ_b) if z_i is on or above the segment between $\text{ext}(I, \theta_a)$ and $\text{ext}(I, \theta_b)$. For discrete distributions, the convex hull might always be a fixed line with many collinear points on it. We cannot find an angular slab with few points above it.

Proof. We compute the upper hull $\text{conv}(I_j)$, for all j . Each $\text{conv}(I_j)$ can be represented as an ordered list of angles, each corresponding to the normal direction of an edge, that is, we have an increasing list of angles $\alpha_1, \alpha_2, \dots, \alpha_r$ for $\text{conv}(I_j)$ such that the outward normal

to the i th edge of $\text{conv}(I_j)$ makes angle α_i with the negative x -axis. We merge the angle lists to get a single sorted list $\theta_0, \theta_1, \dots, \theta_s$, where $s \leq kn$. We will set $\theta_0 := 0$ (it is parallel to the negative x -axis) and $\theta_s := \pi$ (parallel to the positive x -axis). For every θ_a and $\text{conv}(I_j)$, we can easily determine $\text{ext}(I_j, \theta_a)$. Note that for every point of $\text{conv}(I_j)$, there exists some θ_a such that the point is extreme for θ_a .

We give a data structure to compute these values quickly. Fix an I_j . We construct vertical slabs through the points in $\text{conv}(I_j)$, and locate every other point of I_j in its respective vertical leaf slab. This takes $O(n \log n)$ time. We take all pairs of extremal points z_c, z_d with at most $\log^2 n$ vertices on $\text{conv}(I_j)$ (considered as a polygonal chain) between them. For each such pair, we compute the number of points above the segment $\overline{z_c z_d}$ and store this in a table. For this, we need only consider points in leaf slabs between z_c and z_d . How much time does this take for all pairs? Each point is considered for $O(\log^4 n)$ segments, so the total time required to get all this information is $O(n \log^4 n)$. We construct a binary search tree over the slopes of the edges of $\text{conv}(I_j)$. Given θ_a, θ_b , we can find them in this binary search tree, and get $\text{ab}(I_j, \theta_a, \theta_b)$ (if it exists) from the table. If $\text{ab}(I_j, \theta_a, \theta_b)$ is at most $\log^2 n$, we have the exact value. If $\text{ab}(I_j, \theta_a, \theta_b)$ is not present in the table, then it is more than $\log^2 n$. We have such a data structure for all j . Given any θ_a, θ_b , if $\text{tot}(\theta_a, \theta_b)$ is at most $\log^2 n$, then we can determine its value in $O(\log^3 n)$ time. Otherwise, we know that it is more than $\log^2 n$. This will suffice for our purposes.

We now describe how to get the list of directions V for **A**. We perform a greedy pass through the angle list and include some θ_i 's into V as we go. Initially, we include θ_0 in V and set the current position θ_a to θ_0 . Next, we try to find a θ_b such that $\text{tot}(\theta_a, \theta_b)$ is in the range $[\log^2 n/2, \log^2 n]$. If θ_b exists, we add it to V , set $\theta_a := \theta_b$ and continue. In not, let θ_b be the largest angle (in the list) such that $\text{tot}(\theta_a, \theta_b) < \log^2 n$. If θ_b is the last angle in the list, we add it to V and stop. Otherwise, we include θ_b and θ_{b+1} in V , set $\theta_a := \theta_{b+1}$ and continue. Note that $\text{tot}(\theta_b, \theta_{b+1}) = 0$. This is because, for every j , the points $\text{ext}(I_j, \theta_b)$ and $\text{ext}(I_j, \theta_{b+1})$ are either identical or are consecutive on $\text{conv}(I_j)$. The number of points in the angular slab (θ_a, θ_{b+1}) summed over all j is at least $\log^2 n$. How large is V ? Since there are a total of $n \log^2 n$ points, the number of these iterations is at most $O(n)$. We add at most 2 new directions to V , yielding the desired list. \square

Ideally, we would like the expected number of points above any leaf slab λ of **A** to be constant. Unfortunately, that is not the case, since this random variable is the sum of highly dependent random variables. Since

the position of a point can change the angular slab, we cannot resort to a direct argument as before. Nonetheless, we can argue that the expected *logarithm* of the number of points in a leaf slab is constant. This is still not enough to sort leaf slabs in linear time, so we then exploit the independence of the points to show that the expected sorting time will be linear.

We will first prove a claim showing that the expectation of $\log X_\lambda$ is small, using the property given in Claim 4.2. Then, we prove Lemma 3.2.

CLAIM 4.3. *With probability $> 1 - n^{-3}$ over the construction of \mathbf{A} , for all leaf slabs λ , $\mathbf{E}[\log(X_\lambda + 1)] = O(1)$.*

Proof. Fix two angles θ_a and θ_b . Let t be the angular slab (θ_a, θ_b) . Let p_t be the probability that, for a random I , $\text{ab}(I, t)$ is larger than $\log^2 n$. The probability that for *no* I_j , $\text{ab}(I_j, t)$ is larger than $\log^2 n$ is $(1 - p_t)^k = (1 - p_t)^{\log^2 n}$. So if $p_t > 6/\log n$, the probability that some $\text{ab}(I_j, t)$ is larger than $\log^2 n$ is at least $1 - 1/n^6$. Taking a union bound over all t (and rephrasing), with probability $> 1 - 1/n^4$, for any t , if all $\text{ab}(I_j, t)$ are less than $\log^2 n$, then $p_t < 6/\log n$.

Now suppose we draw r independent instances I'_1, I'_2, \dots, I'_r ($r \geq \log^2 n/2$) conditioned on $\text{ab}(I'_j, t)$ being less than $\log^2 n$. Consider the random variables $Y'_{t,j}$ defined as $Y'_{t,j} = \log(\text{ab}(I'_j, t) + 1)$. These are independent and in the range $[0, 3 \log \log n]$. We can apply Hoeffding's inequality [11, Theorem 2] on the sum $Y'_t = \sum_j Y'_{t,j}$.

$$\begin{aligned} & \Pr[|Y'_t - \mathbf{E}[Y'_t]| \geq \log^2 n] \\ & \leq 2 \exp\left(-\frac{2 \log^4 n}{18 \log^2 n (\log \log n)^2}\right) = n^{-\omega(1)}. \end{aligned}$$

Taking a union bound over all t , with probability $> 1 - 1/n^4$, if $Y'_t < 2 \log^2 n$, then $\mathbf{E}[Y'_t] \leq 3 \log^2 n$. From now on, we assume that all these events hold.

By Claim 4.2, we can conclude that for every λ , $p_\lambda < 6/\log n$. Fix a λ . This is defined by angles from two instances from I_1, \dots, I_k . All the other instances are independent of λ . Since all $\text{ab}(I_j, \lambda)$ are less than $\log^2 n$, so we can think of these instances as being drawn independently conditioned on the event $\mathcal{E}_\lambda := \{\text{ab}(I_\lambda) < \log^2 n\}$. Let $Y_{\lambda,j} = \log(\text{ab}(I_j, \lambda) + 1)$ and $Y_\lambda = \sum_j Y_{\lambda,j}$. By Claim 4.2, $Y_\lambda \leq \sum_j (\text{ab}(I_j, \lambda) + 1) \leq \text{tot}(\lambda) + \log^2 n \leq 2 \log^2 n$. Using the condition proven in the previous paragraph, this implies $\mathbf{E}_{\mathcal{E}_\lambda}[Y_\lambda] \leq 3 \log^2 n$. We can argue that $\mathbf{E}_{\mathcal{E}_\lambda}[Y_\lambda] \geq (\log^2 n - 2) \mathbf{E}_{\mathcal{E}_\lambda}[\log(X_\lambda + 1)]$ (because there are $k - 2$ independent instances). This shows that $\mathbf{E}_{\mathcal{E}_\lambda}[\log(X_\lambda + 1)] \leq \alpha$, for some constant α .

We are now ready to bound $\mathbf{E}[\log(X_\lambda + 1)]$. Note that $\Pr[\overline{\mathcal{E}_\lambda}] < 6/\log n$. Also, $\log(X_\lambda + 1)$ can be trivially

bounded by $2 \log n$, so

$$\begin{aligned} & \mathbf{E}[\log(X_\lambda + 1)] \\ & = \mathbf{E}_{\mathcal{E}_\lambda}[\log(X_\lambda + 1)] \Pr[\mathcal{E}_\lambda] + \mathbf{E}_{\overline{\mathcal{E}_\lambda}}[\log(X_\lambda + 1)] \Pr[\overline{\mathcal{E}_\lambda}] \\ & \leq \alpha + (2 \log n)(6/\log n) = O(1). \end{aligned}$$

□

Proof. (of Lemma 3.2) Let $\chi(i, \lambda)$ be the indicator random variable for the event that point z_i is *in* λ or is one of the two boundary points for λ . We define a new random variable $W(i, \lambda)$ such that $W(i, \lambda) = \log(X_\lambda + 1)$ if z_i is not a boundary point for λ , and $W(i, \lambda) = 0$, otherwise. We can see that $X_\lambda \log(X_\lambda + 1) = \sum_i \chi(i, \lambda) W(i, \lambda)$.

$$\begin{aligned} \mathbf{E}\left[\sum_\lambda X_\lambda \log(X_\lambda + 1)\right] &= \mathbf{E}\left[\sum_\lambda \sum_i \chi(i, \lambda) W(i, \lambda)\right] \\ &= \sum_i \sum_\lambda \mathbf{E}[\chi(i, \lambda) W(i, \lambda)] \end{aligned}$$

We split into two cases. If $\mathbf{E}[\chi(i, \lambda)] > 1/2$, we trivially bound $\chi(i, \lambda)$ by 1 to get, by Claim 4.3, $\mathbf{E}[\chi(i, \lambda) W(i, \lambda)] \leq \mathbf{E}[\chi(i, \lambda) \log(X_\lambda + 1)] \leq \mathbf{E}[\log(X_\lambda + 1)] = O(1)$. This happens for at most four λ s, since $\sum_\lambda \chi(i, \lambda) \leq 2$.

We now assume that $\Pr[\chi(i, \lambda) = 1] = \mathbf{E}[\chi(i, \lambda)] \leq 1/2$. Let \mathcal{E} be the event that $z_i \notin \lambda$, ie, $\chi(i, \lambda) = 0$. Hence, $\Pr[\mathcal{E}] \geq 1/2$. Since $\mathbf{E}_{\mathcal{E}}[\log(X_\lambda + 1)] \Pr[\mathcal{E}] \leq \mathbf{E}[\log(X_\lambda + 1)]$, we have $\mathbf{E}_{\mathcal{E}}[\log(X_\lambda + 1)] \leq 2 \mathbf{E}[\log(X_\lambda + 1)] = O(1)$. Consider the random experiment where we take the point set $I' = I \setminus z_i$. We can define X'_λ 's analogous to X_λ for this experiment. Now suppose we add z_i conditioned on \mathcal{E} . By independence, this does not affect the position of any other point, so the extreme points defining λ are the same in I' and I . Furthermore, $X_\lambda = X'_\lambda$, as $z_i \notin \lambda$, so $\mathbf{E}[\log(X'_\lambda + 1)] = \mathbf{E}_{\mathcal{E}}[\log(X_\lambda + 1)] = O(1)$.

Let us now analyze $\mathbf{E}[\chi(i, \lambda) W(i, \lambda)]$. We split this into two events. Let \mathcal{E}_1 be the event that z_i is one of the boundary points for λ , and \mathcal{E}_2 be the event that z_i is in λ but not on the boundary. Note that when neither \mathcal{E}_1 or \mathcal{E}_2 occur, $\chi(i, \lambda) = 0$. Therefore, $\mathbf{E}[\chi(i, \lambda) W(i, \lambda)] = \mathbf{E}_{\mathcal{E}_1}[W(i, \lambda)] \Pr[\mathcal{E}_1] + \mathbf{E}_{\mathcal{E}_2}[W(i, \lambda)] \Pr[\mathcal{E}_2]$. For the first term, note that in \mathcal{E}_1 , $W(i, \lambda) = 0$. Conditioned on \mathcal{E}_2 , the point z_i does not affect the angular slab λ and $W(i, \lambda) = \log(X_\lambda + 1)$. Hence the conditioned X_λ is at most the unconditioned $X'_\lambda + 1$, and $\mathbf{E}_{\mathcal{E}_2}[W(i, \lambda)] = \mathbf{E}_{\mathcal{E}_2}[\log(X_\lambda + 1)] \leq \mathbf{E}[\log(X'_\lambda + 2)] = O(1)$. Thus, $\mathbf{E}[\chi(i, \lambda) W(i, \lambda)] = O(\Pr[\mathcal{E}_2]) = O(\mathbf{E}[\chi(i, \lambda)])$.

The sum $\sum_\lambda \mathbf{E}[\chi(i, \lambda) W(i, \lambda)]$ can be split into two parts, where the first is summed over λ such that $\mathbf{E}[\chi(i, \lambda)] > 1/2$ and the other summed over the condition $\mathbf{E}[\chi(i, \lambda)] \leq 1/2$. The first sum contributes $O(1)$.

The second contributes at most $O(\sum_{\lambda} \mathbf{E}[\chi(i, \lambda)]) = O(1)$. Therefore, the total sum $\mathbf{E}[\sum_{\lambda} X_{\lambda} \log(X_{\lambda} + 1)] = \sum_i \sum_{\lambda} \mathbf{E}[\chi(i, \lambda)W(i, \lambda)]$ is bounded by $O(n)$. \square

4.3 Learning the ℓ_{ak} 's. It is more convenient to think of this problem in the dual space. The input I is a set of n lines. Each direction $v_a \in V$ is now (using the appropriate duality transform) an x -coordinate v_a^* in the dual space. All points ℓ_{ak}^* have x -coordinate v_a^* . The expected number of input lines below point ℓ_{a1}^* is in the range $[\delta, 2\delta]$. Conditioned on there being no lines below ℓ_{ak}^* , the expected number of points below $\ell_{a(k+1)}^*$ is in the range $[\delta, 2\delta]$. We will construct all these (dual) points up to $k \leq \varepsilon \log n$.

Proof. of Lemma 3.3 We first describe the procedure used to generate the points ℓ_{ak}^* .

We start by choosing $\tau = n^{\varepsilon} \log n$ random instances I_1, \dots, I_{τ} . For all j , we compute $\text{conv}(I_j)$. We take the union of all these instances, and look at the dual \mathcal{I}^* . We compute the ($\leq n^{\varepsilon} \log^2 n$)-level of the arrangement of \mathcal{I}^* . Since we assume that the distributions are continuous, this arrangement has no degeneracies. By a bound of Clarkson and Shor [8], the total size of this arrangement is $n^{1+2\varepsilon} \log^3 n$ (it can also be computed in this time). We are now ready to find the points ℓ_{ak}^* . We walk upwards along the line $x = v_a^*$, starting from the point, denoted ℓ_{a0}^* , that this line hits the lower envelope of \mathcal{I}^* . We maintain a redundant set R of instances. This is initially empty. We proceed in phases, and find ℓ_{ak}^* in the k th phase. In the k th phase, we start from $\ell_{a(k-1)}^*$, an empty set S , and a counter set to 0. We proceed upwards along the line v_a^* and find the next line ℓ it hits. If $\ell \in I_j$ such that $j \notin R$, then we increment a counter and add j to S . If $j \in R$, we keep the counter's value. We then proceed to the next line that is hit. By the non-degeneracy of the arrangement, the counter increments one by one and never jumps by more. This keeps on going until our counter reaches $\lfloor 3\delta(\tau - |R|)/2 \rfloor$. The point we stop at is ℓ_{ak}^* . We add all elements in S to R and go to the next phase. Intuitively, in each phase, we remove all the $I_j, j \in R$ and deal with the remaining arrangement. We keep doing this until $k = \varepsilon \log n$. We now show below that the process cannot terminate before that and we get the desired properties.

CLAIM 4.4. *With probability $> 1 - n^{-3}$, the above procedure gives all points ℓ_{ak}^* with the desired properties, for $k \leq \varepsilon \log n$. It takes $n^{\varepsilon} \log n$ rounds and $n^{1+O(\varepsilon)}$ time.*

Proof. We prove by induction on k . The base case is identical to the induction step, so we shall just prove the latter. Suppose we have obtained all points up

to $\ell_{a(k-1)}^*$ with the desired properties. The probability that all lines in a random instance I^* are above $\ell_{a(k-1)}^*$ is at least $1/2^{k-1}$ (Claim 3.3). Since $k \leq \varepsilon \log n$ and $\tau = cn^{\varepsilon} \log n$, a Chernoff bound argument tells us that with probability $> 1 - n^{-6}$, there are at least $c \log n/2$ instances in \mathcal{I} that are above $\ell_{a(k-1)}^*$. This means that in the k th phase, the size of the set of remaining instances (called $\mathcal{I}_{\overline{R}}$) ($\tau - |R|$) is at least $c \log n/2$. We can imagine that these remaining instances $\mathcal{I}_{\overline{R}}$ are drawn from \mathcal{D} conditioned on I^* being above $\ell_{a(k-1)}^*$. This still preserves the independence of the \mathcal{D}_i since we can apply this conditioning to each distribution. Let p be the intersection point of some line of $\mathcal{I}_{\overline{R}}$ with $x = v_a^*$. Let Y_p be the random variable that is the number of lines in $\mathcal{I}_{\overline{R}}$ strictly below p . This can be expressed as the sum of independent random variables and we get by a Chernoff bound that $Y_p \in (9\mathbf{E}[Y_p]/10, 11\mathbf{E}[Y_p]/10)$ with probability at least $1 - e^{-\Omega(\mathbf{E}[Y_p])}$.

By a union bound over all p , with probability $> 1 - n^{-6}$, if $Y_p = \lfloor 3\delta(\tau - |R|)/2 \rfloor$, then $\mathbf{E}[Y_p] \in [5\delta(\tau - |R|)/4, 7\delta(\tau - |R|)/4]$. This implies that the expected number of lines below ℓ_{ak}^* , conditioned on there being none below $\ell_{a(k-1)}^*$ is in the range $[\delta, 2\delta]$. (For $k = 1$, there is no conditioning.) We apply this argument inductively and perform a union bound over all errors to complete the proof. \square

4.4 Learning the T_i and A_i trees. We show how to build the search trees T_i and A_i . These techniques are identical to those used in For the sake of clarity, we give a detailed explanation for this setting. After learning \mathbf{S} , we construct a simple binary search tree B for this slab structure. Given any point z_i , we can find the vertical slab it belongs to in $O(\log n)$ time.

Proof. (of Lemma 3.4) We will construct T_i and A_i trees (over the respective slab structures) of size n^{ε} . We do the search as before on these trees. Obviously, these trees can be complete search trees because of their small size. Suppose a point reaches a T_i leaf. If this is a leaf in \mathbf{S} , we are done anyway. If not, we search for the point in B and find the correct leaf slab. We do the same for the A_i trees. If a search reaches a leaf of A_i that is not a leaf of A_i , we just search for the point in B . In round k , we add all these points found in round k to X_k before Step 4. We will still find the extreme points e_a in the same (or lesser) number of rounds, since we only add points to X_k . All claims hold as before. We only need to bound the *extra* search time we incur by our searches in B . We will show that to be at most an $O(1/\varepsilon)$ factor more than the old search time.

First, let us describe the construction of these trees.

We will need $O(n^\varepsilon \log n)$ rounds to learn approximate T_i and A_i trees. The main observation is that (up to constant factors), the only probabilities that are relevant are those $> n^{-\varepsilon}$. In each round, for each z_i , we record the leaf slab of \mathbf{S} that it belongs to. If z_i is not extremal, we record the appropriate leaf slab of \mathbf{A} .

Let us focus on \mathbf{S} and the T_i -trees for now. Based on the $O(n^\varepsilon \log n)$ instances of z_i and their positions in \mathbf{S} , we will construct an approximate T_i -tree. For a slab S , let $N(S)$ be the number of times z_i was in S . We use $N(S)$ divided by the total number of rounds as an estimate $\hat{p}(i, S)$ of $p(i, S)$.⁸ We now follow the old procedure that finds the T_i -tree, using these estimates. A slight change is that whenever we reach a slab S such that $N(S) \leq \log n$, we do not proceed beyond S . In the end, we have an approximate T_i -tree of size $O(n^\varepsilon)$. We perform this construction to get all T_i and analogously, A_i -trees. The total space used is $O(n^{1+\varepsilon})$. By a Chernoff bound argument, with probability $> 1 - n^{-4}$, for any slab S such that $N(S) > \log n$, $\hat{p}(i, S) = \Theta(p(i, S))$. Furthermore, if $p(i, S) = \Omega(n^{-\varepsilon})$, then $N(S) > \log n$. We will henceforth assume these to be true for all T_i 's.

We show why these T_i -trees are sufficient. As a thought experiment, we extend this to an almost optimal T_i' -tree. Consider any leaf S of T_i . We construct the optimal tree for S (assuming we know \mathcal{D}_i) and attach it to S . Doing this for all leaves of T_i gives us T_i' . We argue that T_i' is an almost optimal search tree for \mathcal{D}_i . We prove that:

CLAIM 4.5. *For any slab S of T_i' at depth k , $p(i, S) \leq \mu^k$ (for some fixed constant $0 < \mu < 1$).*

Proof. Suppose S with depth k belongs to T_i , and is not a leaf. We will set μ to be some appropriate constant close to 1. By construction, $\hat{p}(i, S) \leq (2/3)^k$. By the properties given above, $p(i, S) = O(\hat{p}(i, S)) \leq \mu^k$. If S is a T_i leaf, then by looking at the parent of S , we get $p(i, S) = O((2/3)^{k-1}) \leq \mu^k$. Now, suppose S at depth k is not in T_i . Let P be the leaf of T_i that is an ancestor of S . Break up the depth of S into the depth k_P of P and the depth of S k_S in the subtree rooted at P . By construction of the subtree, $p(i, S)/p(i, P) \leq (2/3)^{k_S}$. Thus, $p(i, S) \leq p(i, P)(2/3)^{k_S} \leq \mu^{k_P}(2/3)^{k_S} \leq \mu^k$. \square

Since T_i' is almost optimal, if we use it for our self-improving algorithm, we obtain the proven bounds. Now, suppose we perform the searches in parallel in both T_i and T_i' . Note that the behavior is identical as long as z_i is in a non-leaf slab of T_i . Suppose we reach a leaf of T_i . If this is leaf slab of \mathbf{S} , we are done. If not,

then we search for z_i in B and pay $O(\log n)$ search cost. Why is this alright? Let us denote the T_i leaf reached by S . By the Chernoff argument, $p(i, S) = O(n^{-\varepsilon})$, since $N(S) \leq \log n$. By the optimality properties, the depth of S is at least $\Omega(-\log p(i, S)) = \Omega(\varepsilon \log n)$. Since this part of T_i is identical to T_i' , we have spent at least (up to constant factors) $\varepsilon \log n$ time searching in T_i' for z_i . By doing the $O(\log n)$ search, we pay a multiplicative factor of at most $O(1/\varepsilon)$ over the (fictitious) algorithm that used T_i' . Therefore, our search cost is at most $O(1/\varepsilon)$ times the old search cost. The old search cost, by Lemmas 3.6, 3.7, 3.8, is $O(H(\text{conv}(I)))$. A similar argument applies to the A_i trees. \square

5 Dealing with discrete distributions

Our main analysis postulates continuous distributions, since this setting appears more natural for the problem. However, this assumption can be easily removed. The main problem with discrete distributions is that the ℓ_{ak} may no longer exist: for example, if z_i occupies a fixed location with probability $1/2 > \delta$, we might never find appropriate ℓ_{ak} 's. Here we sketch how to avoid this problem.

Consider the construction of the ℓ_{ak} 's in the learning phase. In the dual, we walk along the line $\ell : x = v_a^*$ and increment a counter as we hit lines of the arrangement \mathcal{I}^* . If many lines intersect ℓ in the same point, the counter could skip values. To avoid this, we perform a symbolic perturbation of the input. More precisely, we impose an arbitrary ordering on the lines in \mathcal{I}^* , and process coincident lines on ℓ in this order, incrementing the counter one by one. Conceptually, we imagine that the intersection points are distinct and our sweep point “moves” across these points. While the counter is less than $\lfloor 3\delta(\tau - |R|)/2 \rfloor$, we assume that the lines are below our sweep point. Once the counter reaches this value, we assume that the remaining coincident intersections (if any) are “above” our sweep point. This procedure yields a set of lines ℓ_{ak} as before, and because of the symbolic perturbation, the analysis goes through as before.⁹ Since some of the ℓ_{ak} lines might actually be identical, preprocessing now yields a more subtle variant of Condition (3.2). Let ℓ_{ak}^+ be the open and $\overline{\ell_{ak}^+}$ the closed halfplane above ℓ_{ak} . Then,

$$\mathbf{E}[\lceil |I \cap \overline{\ell_{a1}^+}| \rceil] \geq \delta, \quad \mathbf{E}[\lceil |I \cap \ell_{a1}^+| \rceil] \leq 2\delta.$$

⁹Indeed, we can imagine a continuous distribution which first draws points from the discrete distribution and then perturbs the points infinitesimally.

⁸We remind the reader that this the probability that $z_i \in S$.

For $k \geq 1$,

$$\mathbf{E} \left[\left| I \cap \overline{\ell_{a(k+1)}^+} \mid I \cap \overline{\ell_{ak}^+} = \emptyset \right| \right] \geq \delta$$

$$\mathbf{E} \left[\left| I \cap \ell_{a(k+1)}^+ \mid I \cap \overline{\ell_{ak}^+} = \emptyset \right| \right] \leq 2\delta.$$

We modify the algorithm slightly: in Step 2, we continue the search in T_i if (i) z_i is strictly above $\text{seg}_k(S_i)$ or (ii) z_i lies on $\text{seg}_k(S_i)$ and the segment is an edge of C_k . This ensures that any extremal point on the boundary of C_k is found in round k . The analysis also needs to be refined, the condition from Claim 3.2 becomes $I \cap \overline{\ell_{a(k-1)}^+} = \emptyset \wedge I \cap \overline{\ell_{ak}^+} \neq \emptyset$, and with the variant of Condition (3.2), the upper bounds from Claim 3.3 are still valid and the analysis of Lemma 3.6 holds, so everything goes through as before. The lower bound from Claim 3.3 was only used for the learning phase, and this part still holds because of the symbolic perturbation.

References

- [1] P. Afshani, J. Barbay, and T. M. Chan. Instance-optimal geometric algorithms. FOCS 2009.
- [2] N. Ailon, B. Chazelle, K. L. Clarkson, D. Liu, W. Mulzer, and C. Seshadhri. Self-improving algorithms. Manuscript at [arXiv:0907.0884](https://arxiv.org/abs/0907.0884), 2009.
- [3] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Self-improving algorithms. In *SODA*, pages 261–270, 2006.
- [4] N. Alon and J. H. Spencer. *The probabilistic method*. Wiley-Interscience, New York, second edition, 2000.
- [5] J. Barbay. Adaptive (analysis of) algorithms for convex hulls and related problems. Available online at <http://www.cs.uwaterloo.ca/~jbarbay/Recherche/Publishing/Publications/#asimuh>, 2008.
- [6] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Delaunay triangulation of imprecise points simplified and extended. In *WADS*, pages 131–143, 2009.
- [7] K. L. Clarkson and C. Seshadhri. Self-improving algorithms for Delaunay triangulations. In *SoCG*, 2008.
- [8] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4(1):387–421, 1989.
- [9] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Comput. System Sci.*, 30(2):209–221, 1985.
- [10] M. Held and J. S. B. Mitchell. Triangulating input-constrained planar point sets. *Inf. Process. Lett.*, 109(1):54–56, 2008.
- [11] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
- [12] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Computing*, 15(1):287–299, 1986.
- [13] M. Löffler and J. Snoeyink. Delaunay triangulations of imprecise points in linear time after preprocessing. In *SoCG*, pages 298–304, 2008.
- [14] M. J. van Kreveld, M. Löffler, and J. S. B. Mitchell. Preprocessing imprecise points and splitting triangulations. In *ISAAC*, pages 544–555, 2008.