Convex Hull of Points Lying on Lines in $o(n \log n)$ Time after Preprocessing[☆]

Esther Ezra[a,1], Wolfgang Mulzer[b,2]

[a] *Courant Institute of Mathematical Sciences, New York University, New York, USA*
[b] *Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany*

Abstract

Motivated by the desire to cope with *data imprecision* [31], we study methods for taking advantage of preliminary information about point sets in order to speed up the computation of certain structures associated with them.

In particular, we study the following problem: given a set L of n lines in the plane, we wish to preprocess L such that later, upon receiving a set P of n points, each of which lies on a distinct line of L, we can construct the convex hull of P efficiently. We show that in quadratic time and space it is possible to construct a data structure on L that enables us to compute the convex hull of any such point set P in $O(n\alpha(n)\log^* n)$ expected time. If we further assume that the points are "oblivious" with respect to the data structure, the running time improves to $O(n\alpha(n))$. The same result holds when L is a set of line segments (in general position). We present several extensions, including a trade-off between space and query time and an output-sensitive algorithm. We also study the "dual problem" where we show how to efficiently compute the $(\leq k)$-level of n lines in the plane, each of which is incident to a distinct point (given in advance).

We complement our results by $\Omega(n \log n)$ lower bounds under the algebraic computation tree model for several related problems, including sorting a set of points (according to, say, their x-order), each of which lies on a given line known in advance. Therefore, the convex hull problem under our setting is *easier* than sorting, contrary to the "standard" convex hull and sorting problems, in which the two problems require $\Theta(n \log n)$ steps in the worst case (under the algebraic computation tree model).

Keywords: data imprecision, convex hull, planar arrangements, geometric data structures, randomized constructions

1. Introduction

Most studies in computational geometry rely on an unspoken assumption: whenever we are given a set of input points, their precise locations are available to us. Nowadays, however, the input is often obtained via sensors from the real world, and hence it comes with an inherent imprecision. Accordingly, an increasing effort is being devoted to achieving a better understanding of data imprecision and to developing tools to cope with it (see, e.g., [31] and the references therein). The notion of imprecise data can be formalized in numerous ways [24, 31, 33]. We consider a particular setting that has recently attracted considerable attention [8, 20, 25, 30, 32, 34]. We are given a set of planar regions, each of which represents an estimate about an input point, and the exact coordinates of the points arrive some time later and need to be processed quickly. This situation could occur, e.g., during a two-phase measuring process: first the sensors quickly obtain a rough estimate of the data, and then they invest considerably more time to find the precise locations. This raises the necessity to preprocess the preliminary (imprecise) locations of the points, and store them in an appropriate data structure, so that when the exact measurements of the points arrive we can efficiently compute a pre-specified structure on them. In settings of this kind, we assume that for each input point its corresponding region is known (note that by this assumption we also avoid a point-location overhead). In light of

[☆] A preliminary version appeared as E. Ezra and W. Mulzer. *Convex Hull of Imprecise Points in o(nlogn) Time after Preprocessing* in Proc. 27th SoCG, pp. 11–20, 2011

[1] Supported by a PSC-CUNY Research Award.
[2] Supported in part by NSF grant CCF-0634958, NSF CCF 083279, and a Wallace Memorial Fellowship in Engineering.

the applications, this is a reasonable assumption, and it can be implemented by, e.g., encoding this information in the ordering of P.

Related work

Data imprecision. Previous work has mainly focused on computing a triangulation for the input points. Held and Mitchell [25] were the first to consider this framework, and they obtained optimal bounds for preprocessing disjoint unit disks for point set triangulations, a result that was later generalized by van Kreveld *et al.* [30] to arbitrary disjoint polygonal regions. For *Delaunay* triangulations, Löffler and Snoeyink [34] obtained an optimal result for disjoint unit disks (see also [20, 32]), which was later simplified and generalized by Buchin *et al.* [8] to *fat*[3] and possibly intersecting regions. If n is the number of input regions, the preprocessing phase typically takes $O(n \log n)$ time and yields a linear size data structure; the time to find the structure on the exact points is usually linear or depends on the complexity (and the fatness) of the input regions.

Since the convex hull can be easily extracted from the Delaunay triangulation in linear time, the same bounds carry over. However, once the regions are not necessarily fat, the techniques in [8, 34] do not yield the aforementioned bounds anymore. In particular, if the regions consist of lines or line segments, one cannot hope (under certain computational models) to construct the Delaunay triangulation of P in time $o(n \log n)$, regardless of preprocessing (see [22] and Section 4). Nevertheless, if we are less ambitious and just wish to compute the *convex hull* of P, we can achieve better performance, as our main result shows.

Convex hull. Computing the convex hull of a planar n-point set is perhaps the most fundamental problem in computational geometry, and there are many algorithms available [6, 37]. All these algorithms require $\Theta(n \log n)$ steps, which is optimal in the algebraic computation tree model [5]. However, there are numerous ways to exploit additional information to improve this bound. For example, if the points are sorted along any fixed direction, Graham's scan takes only linear time [6]. If we know that there are only h points on the hull, the running time reduces to $O(n \log h)$ [1, 28]. If the points constitute the vertices of a given polygonal chain, the complexity again reduces to linear [36]. Our work shows another setting in which additional information can be used to circumvent the theoretic lower bound.

Another somewhat related problem (albeit conceptually different) is the *kinetic convex hull* problem, where we are given n points which move *continuously* in the plane, and the goal is to maintain their convex hull over time. Kinetic data structures have been introduced by Basch *et al.* [4] and received considerable attention in follow-up studies (see, e.g., [2] and the references therein). When the trajectories of the points are lines, our problem can be interpreted as a (perhaps, extended and intricate) variant of the kinetic convex hull problem. Indeed, if the goal is to preprocess the linear trajectories such that the convex hull can be reported efficiently at any given time t, our algorithm applies (in which case the exact set of points P consists of their positions at time t) and yields a relatively simple solution. Nevertheless, our problem is more intricate than the kinetic convex hull problem for linear trajectories, as in our scenario there is no continuous motion that enables us to have a better control on the exact set of points (once they arrive).

Our results. We show that under a mild assumption (see Section 2.2) we can preprocess the input lines L such that given any set P of points, each of which lies on a distinct line of L, the convex hull $\mathrm{CH}(P)$ can be computed in expected time $O(n\alpha(n))$, where $\alpha(\cdot)$ is the (slowly growing) inverse Ackermann function [41, Chapter 2.1]; the expected running time is $O(n\alpha(n) \log^* n)$ without this assumption. Our data structure has quadratic preprocessing time and storage, and the convex hull algorithm is based on a batched randomized incremental construction similar to Seidel's tracing technique [40]. As part of the construction, we repeatedly trace the *zone* of (the boundary of) an intermediate hull in the *arrangement* of the input lines (see below for the definitions). The fact that the complexity of the zone is only $O(n\alpha(n))$ [7, 41], and that it can be computed in the same asymptotic time bound (after having the arrangement at hand), is a key property of our solution. The analysis also applies when L is a set of line segments, and yields the same result.

We also show that the analogous problem in which we just wish to sort the points according to their x-order imposes algebraic computation trees of depth $\Omega(n \log n)$. Hence, in our setting convex hull computation is strictly

[3] A planar region o is said to be fat if there exist two concentric disks, $D \subseteq o \subseteq D'$, such that the ratio between the radii of D' and D is bounded by some constant.

easier than sorting, contrary to the "standard" (unconstrained) model, in which both problems are equivalent in terms of hardness (see, e.g., [6]). Our results can be extended with similar bounds to several related problems, such as determining the width and diameter of P, as well as time-space trade-offs and designing an output-sensitive algorithm. Unfortunately, already for the closest pair problem a preprocessing of the regions is unlikely to decrease the query time to $o(n \log n)$, demonstrating once again the delicate nature of our setting.

In Section 3 we study a generalization of the problem under the dual setting. Specifically, we wish to preprocess a planar n-point set P such that given an integer k and a set L of lines, each of which is incident to a distinct point of P, we can find the "($\leq k$)-level" in the arrangement of L efficiently. We show a randomized construction whose expected running time is $O(n\alpha(n) + nk)$ under a mild assumption, and $O(n\alpha(n) \log^* n + nk)$ without this assumption. As above, our data structure has quadratic preprocessing time and storage. This improves over the $O(n \log n + nk)$ time algorithms in the traditional model [10, 23], as long as $k = o(\log n)$. Our approach is a non-trivial extension of the technique presented in Section 2, incorporated with the algorithms of Chan [10] and Everett *et al.* [23], as well as the Clarkson-Shor technique [16].

The quadratic preprocessing time and storage might seem disappointing. However, a related lower bound by Ali Abam and de Berg [2] from the study of kinetic convex hulls (albeit providing a weaker evidence) suggests that quadratic space might be necessary, and that only relatively weak time-space trade-offs (as in Section 2.3) are possible in this model (see the discussion in Section 2.3 for further details). Given the hardness of related problems, and the fact that previous approaches fail for "thin" regions, it still seems remarkable that improved bounds are even possible.

2. Convex Hulls

Preliminaries. The input at the preprocessing stage is a set L of n lines in the plane. A *query* to the resulting data structure consists of any point set P such that each point lies on a distinct line in L, and for every point we are given its corresponding line. For simplicity, and without loss of generality, we assume that both L and P are in *general position* (see, e.g., [6, 41]). We denote by $\mathrm{CH}(P)$ the convex hull of P, and by $E(P)$ the edges of $\mathrm{CH}(P)$. We represent the vertices of $\mathrm{CH}(P)$ in clockwise order, and we direct each edge $e \in E(P)$ such that $\mathrm{CH}(P)$ lies to its right. Given a subset $Q \subset P$, a point $p \in P \setminus Q$, and an edge $e \in E(Q)$, we say that e is in *conflict* with p if p lies to the left of the line supported by e. The set of all points in $P \setminus Q$ in conflict with e is called the *conflict list* C_e of e, and its cardinality is called the *conflict size* c_e of e.

In what follows we denote the *arrangement* of L by $\mathcal{A}(L)$, defined as the decomposition of the plane into *vertices*, *edges* and *faces* (also called *cells*), each being a maximal connected set contained in the intersection of at most two lines of L and not meeting any other line. The *complexity* of a face f in $\mathcal{A}(L)$ is the number of edges incident to f. The *zone* of a curve γ consists of all faces that intersect γ, and the complexity of the zone is the sum of their complexities.

2.1. The Construction

Preprocessing. We construct in $O(n^2)$ time (and storage) the *arrangement* $\mathcal{A}(L)$ of L, and produce its *vertical decomposition*, that is, we erect an upward and a downward vertical ray through each vertex v of $\mathcal{A}(L)$ until they meet some line of L (not defining v), or else extend to infinity.

Queries. Given an exact point set $P = \{p_1, \ldots, p_n\}$ as described above, we obtain $\mathrm{CH}(P)$ through a *batched* randomized incremental construction. Let $P_1 \subseteq P_2 \subseteq \cdots \subseteq P_{\log^* n} = P$ be a sequence of subsets, where P_{k-1} is a random sample of P_k of size $z_{k-1} := \min\{\lfloor n/\log^{(k-1)} n \rfloor, n\}$, for $k = 2, \ldots, \log^* n$.[4] This sequence of subsets is called a *gradation*. The idea is to construct $\mathrm{CH}(P_1)$, $\mathrm{CH}(P_2)$, ..., $\mathrm{CH}(P_{\log^* n})$ one by one, as follows. First, we have $|P_1| = O(n/\log n)$, so it takes $O(n)$ time to find $\mathrm{CH}(P_1)$, using, e.g., Graham's scan [6]. Then, for $k = 2, \ldots, \log^* n$, we incrementally construct $\mathrm{CH}(P_k)$ by updating $\mathrm{CH}(P_{k-1})$. This basic technique was introduced by Seidel [40] and it has later been exploited by several others [13, 19, 38].

To construct $\mathrm{CH}(P_k)$ from $\mathrm{CH}(P_{k-1})$, we use the data structure from the preprocessing to quickly construct the conflict lists of the edges in $E(P_{k-1})$ with respect to P_k. In the standard Clarkson-Shor randomized incremental

[4] Here, $\log^{(i)} n$ is the ith iterated logarithm: $\log^{(0)} n = n$ and $\log^{(k)} n = \log(\log^{(k-1)} n)$. The standard notation $\log^* n$ is the smallest k such that $\log^{(k)} n \leq 1$.

construction [16] it takes $O(n \log n)$ time to maintain the conflict lists. However, once we have the arrangement $\mathcal{A}(L)$ at hand, this can be done significantly faster.

In fact, we use a refinement of the conflict lists: we shoot an upward vertical ray from each point on the upper hull of P_{k-1}, and a downward vertical ray from each point on the lower hull. Furthermore, we erect vertical walls through the leftmost and the rightmost points of $\mathrm{CH}(P_{k-1})$. This partitions the complement of $\mathrm{CH}(P_{k-1})$ into vertical slabs $S(e)$, for each edge $e \in E(P_{k-1})$, and two boundary slabs $S(v_l)$, $S(v_r)$, associated with the respective leftmost and rightmost vertices v_l and v_r of $\mathrm{CH}(P_{k-1})$. The *refined conflict list* of e, C^*_e, is defined as $C^*_e := (P_k \setminus P_{k-1}) \cap S(e)$. We add to this collection the sets $C^*_{v_l} := (P_k \setminus P_{k-1}) \cap S(v_l)$ and $C^*_{v_r} := (P_k \setminus P_{k-1}) \cap S(v_r)$, which we call the refined conflict lists of v_l and v_r, respectively. Note that $C^*_e \subseteq C_e$, for every $e \in E(P_{k-1})$. Moreover, $C^*_{v_l}$ (resp., $C^*_{v_r}$) is contained in $C_{e_1} \cup C_{e_2}$, where $e_1, e_2 \in E(P_{k-1})$ are the two respective edges emanating from v_l (resp., v_r); see Figure 1(a). We now state a key property of the conflict lists C_e (this property is fairly standard and follows from related studies [13, 16, 38]):

Lemma 2.1. *Let Q be a planar m-point set, r a positive integer satisfying $1 \le r \le m$, and $R \subseteq Q$ a random subset of size r. Suppose that $f(\cdot)$ is a monotone non-decreasing function, so that $f(x)/x^c$ is decreasing, for some constant $c > 0$. Then*

$$\mathbf{Exp}\Big[\sum_{e \in E(R)} f(c_e)\Big] = O\Big(r \cdot f(m/r)\Big),$$

where the constant of proportionality depends on c, and c_e is the number of points $p \in Q \setminus R$ in conflict with $e \in E(R)$. □

In other words, the above lemma implies that, on average, the size of the conflict list of a fixed edge $e \in E(R)$ is m/r (this can easily be seen by setting $f(\cdot)$ to the identity function, and obtaining an overall linear size).

Constructing the refined conflict lists. We next present how to construct the refined conflict lists at the k-th round of the algorithm. We first construct, in a preprocessing step, the refined conflict lists $C^*_{v_l}$, $C^*_{v_r}$ in overall $O(z_k)$ time. We call these points the *extreme* points, and for the sake of the analysis, we eliminate these points from P_k for the time being, and continue processing them only at the final step of the construction—see below.

Let $\mathrm{UH}(P_{k-1})$ be the upper hull of P_{k-1}, and let $\mathrm{LH}(P_{k-1})$ be its lower hull. Having these structures at hand, we construct the zones of $\mathrm{UH}(P_{k-1})$ and $\mathrm{LH}(P_{k-1})$ in $\mathcal{A}(L)$. This takes overall $O(n\alpha(n))$ time, using the vertical decomposition of $\mathcal{A}(L)$ and the fact that the zone complexity of a convex curve in a planar arrangement of n lines is $O(n\alpha(n))$; see Bern *et al.* [7] and Sharir and Agarwal [41, Theorem 5.11].

As soon as we have the zones as above, we can determine for each line $\ell \in L$ the edges $e \in E(P_{k-1})$ that ℓ intersects (if any). Let L_1 be the lines that intersect $\mathrm{CH}(P_{k-1})$, and put $L_2 := L \setminus L_1$. (At this stage of the analysis, we ignore all lines corresponding to points in P_k that were eliminated at the time we processed the extreme points.)

Next, we wish to find, for each point $p \in P_k \setminus P_{k-1}$ the edges in $E(P_{k-1})$ in conflict with p. If p lies inside $\mathrm{CH}(P_{k-1})$, there are no conflicts. Otherwise, we efficiently find an edge $e_p \in E(P_{k-1})$ visible from p, whence we search for the slab $S(e^*_p)$ containing p—see below.

Let us first consider the points on the lines in L_1. Fix a line $\ell \in L_1$, let $p \in P$ be the point on ℓ, and let q_1, q_2 be the intersections between ℓ and the boundary of $\mathrm{CH}(P_{k-1})$. The points q_1, q_2 subdivide ℓ into two rays ρ_1, ρ_2, and the line segment $\overline{q_1 q_2}$. By convexity, $\overline{q_1 q_2} \subseteq \mathrm{CH}(P_{k-1})$ and the rays ρ_1, ρ_2 lie outside $\mathrm{CH}(P_{k-1})$. Hence, if p lies on $\overline{q_1 q_2}$, it must be contained in $\mathrm{CH}(P_k)$. Otherwise, p sees an edge of $E(P_{k-1})$ that meets one of the rays ρ_1, ρ_2, and we thus set e_p to be this edge (which can be determined in constant time); see Figure 1(b).

We next process the lines in L_2. Note that all points on the lines in L_2 conflict with at least one edge in $E(P_{k-1})$, since no line in L_2 meets $\mathrm{CH}(P_{k-1})$. To find these edges we determine for each $\ell \in L_2$ a vertex p_ℓ on the boundary of $\mathrm{CH}(P_{k-1})$ that is extreme for ℓ.[5] This can be done in total time $O(n)$ by ordering $E(P_{k-1})$ and L_2 according to their slopes (the latter being performed during preprocessing), and then merging these two lists in linear time. Next, fix such a line $\ell \in L_2$, and let $p \in \ell$ be a query point, then p must see one of the two edges in $E(P_{k-1})$ incident to p_ℓ (which can be determined in constant time given p_ℓ), and we thus set e_p to be the corresponding edge; see Figure 1(c).

We are now ready to determine, for each point $p \in P_k$ outside $\mathrm{CH}(P_{k-1})$, the slab $S(e^*_p)$ that contains it (note that e^*_p must be vertically visible from p). If e_p is vertically visible from p, we set $e^*_p := e_p$. Otherwise, we walk along

[5] By this we mean that p_ℓ is extremal in the direction of the outer normal of the halfplane that is bounded by ℓ and contains $\mathrm{CH}(P_{k-1})$.

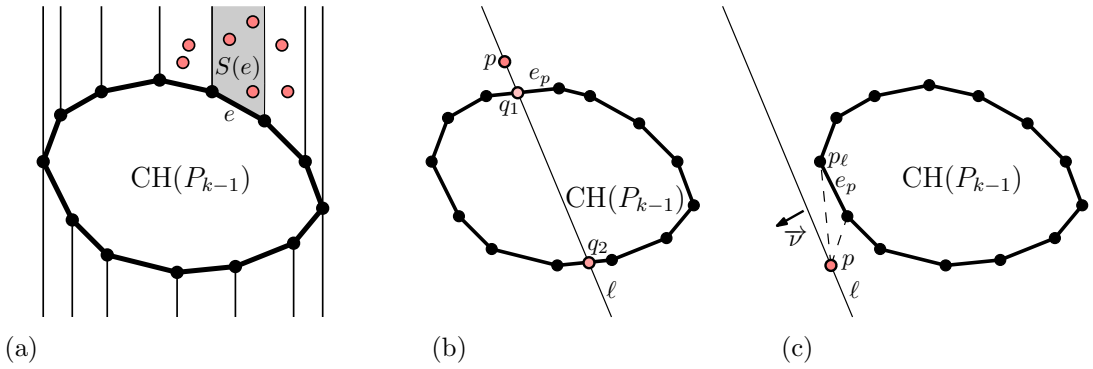

Figure 1: (a) The conflict list C_e of the edge $e \in E(P_{k-1})$ contains all the lightly-shaded points, whereas the refined conflict list $C^*(e)$ has only those points in the vertical slab $S(e)$; (b–c) The edge e_p of $E(P_{k-1})$ is visible to p when (b) ℓ intersects $\mathrm{CH}(P_{k-1})$, or (c) ℓ does not meet $\mathrm{CH}(P_{k-1})$. In this case p_ℓ is an extreme vertex for the direction $\overrightarrow{\nu}$, and the two dashed lines depict the visibility lines between p and the two respective endpoints of e_p.

(the boundary of) $\mathrm{CH}(P_{k-1})$, starting from e_p and progressing in the appropriate direction (uniquely determined by p and e_p), until the slab containing p is found. Using cross pointers between the edges and the points, we can easily compute C^*_e for each $e \in E(P_{k-1})$. By construction, all traversed edges are in conflict with p, and thus the overall time for this procedure is proportional to the total size of the conflict lists C_e. Recalling that $c_e = |C_e|$, we obtain

$$\mathbf{Exp}\Big[\sum_{e \in E(P_{k-1})} c_e\Big] = O(z_k) = O(n),$$

by Lemma 2.1 with $f : m \mapsto m$. This concludes the construction of the refined conflict lists.

Computing $\mathrm{CH}(P_k)$. We next describe how to construct the upper hull of P_k, the analysis for the lower hull is analogous. Let $\langle e_1, \ldots, e_s \rangle$ be the edges along the upper hull of P_{k-1}, ordered from left to right. For each e_i, we sort the points in $C^*_{e_i}$ according to their x-order, using, e.g., merge sort. We apply the same procedure for the extreme points. We then concatenate the sorted lists $C^*_{v_l}, C^*_{e_1}, C^*_{e_2}, \ldots, C^*_{e_s}, C^*_{v_r}$, and merge the result with the vertices of the upper hull of P_{k-1}. Call the resulting list Q, and use Graham's scan to find the upper hull of Q in time $O(|Q|)$. This is also the upper hull of P_k. Applying once again Lemma 2.1 with $f : m \mapsto m \log m$, and putting $c^*_e := |C^*_e|$, $c^*_{v_l} := |C^*_{v_l}|$, $c^*_{v_r} := |C^*_{v_r}|$, and $A > 0$ an absolute constant, the overall expected running time of this step is bounded by

$$\begin{aligned}
&\mathbf{Exp}[A \cdot (c^*_{v_l} \log c^*_{v_l} + c^*_{v_r} \log c^*_{v_r} + \sum_{e \in E(P_{k-1})} c^*_e \log c^*_e)] \\
&\leq \mathbf{Exp}[3A \cdot \sum_{e \in E(P_{k-1})} c_e \log c_e] = O\left(z_k \log\left(z_k / z_{k-1}\right)\right) \\
&= O\left((n/\log^{(k)} n) \log\left(\log^{(k-1)} n / \log^{(k)} n\right)\right) = O(n),
\end{aligned}$$

because by definition $C^*_e \subseteq C_e$, so $c^*_e \leq c_e$, and $c^*_{v_l} \leq c_{e_1} + c_{e_2}$ for two edges e_1, e_2 of $E(P_{k-1})$ (and similarly for $c^*_{v_r}$). In total, we obtain that the expected time to construct $\mathrm{CH}(P_k)$ given $\mathrm{CH}(P_{k-1})$ is $O(n\alpha(n))$, and since there are $\log^* n$ iterations, the total running time is $O(n\alpha(n) \log^* n)$.

We note that the analysis proceeds almost verbatim when L is just a set of *line segments* in the plane. In this case, we preprocess the lines containing the input segments, and proceed as in the original problem. We have thus shown:

Theorem 2.2. *Using $O(n^2)$ space and time, we can preprocess a set L of n lines in the plane (given in general position), such that given any point set P with each point lying on a distinct line in L, we can construct* $\mathrm{CH}(P)$ *in expected time $O(n\alpha(n) \log^* n)$. The same result holds if L is a set of n line segments whose supporting lines are in general position.*

Remark. An inspection of the proof of Theorem 2.2 shows that the total expected conflict size, over all iterations k, is only $O(n)$.

2.2. Better Bounds for Oblivious Points

We now present an improved solution under the *obliviousness model*, where we assume that the points are oblivious to the random choices during the preprocessing step. Specifically, this implies that an adversary cannot pick the point set P in a malicious manner, as it is not aware of the random choices at the preprocessing step. This fairly standard assumption has appeared in various studies (see, e.g., [1, 11]). In the discussion at the end of this section we describe this issue in more detail.

Preprocessing. We now construct a gradation $L_1 \subseteq L_2 \subseteq \cdots \subseteq L_{1+\log\log n} = L$ of the lines during the preprocessing phase, where the set sizes decrease geometrically. Specifically, $|L_1| = y_1 = \lceil n/\log n \rceil$, and for $k = 2, \ldots, 1 + \log\log n$, L_{k-1} is a random subset of L_k of size

$$|L_{k-1}| = y_{k-1} := \left\lceil \frac{y_k}{2} \right\rceil = \left\lceil \frac{1}{2} \cdot \frac{n}{2^{\log\log n - k + 1}} \right\rceil. \tag{1}$$

We construct each arrangement $\mathcal{A}(L_k)$ in $O(n^2 2^{2k-2}/\log^2 n)$ time, for a total $O(n^2)$ time over all gradation steps.

Query. Given an exact input P, we first follow the gradation produced at the preprocessing stage, and generate the corresponding gradation $P_1 \subseteq P_2 \subseteq \ldots \subseteq P_{1+\log\log n} = P$, where $P_k = P \cap L_k$, for all k. By the obliviousness assumption, each P_{k-1} is an unbiased sample of P_k, so Lemma 2.1 applies (see once again the discussion below). Moreover, the key observation is that in order to obtain $\mathrm{CH}(P_k)$ from $\mathrm{CH}(P_{k-1})$, it suffices to confine the search to the arrangement $\mathcal{A}(L_k)$ instead of the entire arrangement $\mathcal{A}(L)$ as in Section 2.1. Thus, we first construct $\mathrm{CH}(P_1)$ in $O(n)$ time as before. Next, to obtain $\mathrm{CH}(P_k)$ from $\mathrm{CH}(P_{k-1})$, we construct the zones of $\mathrm{UH}(P_{k-1})$ and $\mathrm{LH}(P_{k-1})$ in $\mathcal{A}(L_k)$ in $O(y_k \alpha(y_k))$ time, and then compute the refined conflict lists just as in Section 2.1. The overall expected time to produce these lists is $O(y_k)$, totaling $O(n)$ over all steps. As in Section 2.1, the expected time (of the final step) to compute $\mathrm{UH}(P_k)$ and $\mathrm{LH}(P_k)$ is $O\left(y_k \log\left(y_k/y_{k-1}\right)\right) = O\left(y_k\right)$, by (1). Thus, the expected running time at the kth step is dominated by the zone construction, so the overall expected running time is $O\left(\sum_{k=1}^{1+\log\log n} y_k \alpha(y_k)\right) = O\left(n\alpha(n)\right)$, as is easily verified. Thus,

Theorem 2.3. *Using $O(n^2)$ space and time, we can preprocess a set L of n lines in the plane, such that for any point set P with each point lying on a distinct line of L, we can construct* $\mathrm{CH}(P)$ *in expected time $O(n\alpha(n))$ assuming obliviousness.*

Discussion. The issue captured by the obliviousness assumption is: how much does the adversary know about the preprocessing phase? If the adversary manages to obtain the coin flips performed during the preprocessing stage, then this enables a malicious choice of the input. This phenomenon is particularly striking in the case of *hashing*: if the adversary knows the random choice of the hash function, a bad set of inputs can hash all keys to a single slot, completely destroying the hash table. On the other hand, if the adversary is *oblivious* to the hash function, the expected running time per operation is only $O(1)$; see, e.g., [18, Chapter 11].

In our model we encounter a similar phenomenon. Even though the impact is not as disastrous as for hashing, assuming obliviousness for the adversary can improve our running time by a factor of $O(\log^* n)$. To illustrate the effect of obliviousness in our setting, consider the scenario illustrated in Figure 2. In this case, we have a set L of lines and a random subset $L' \subseteq L$. The adversary can pick the point set P so that $P' := P \cap L'$ is a biased sample of P in a sense that violates the properties of Lemma 2.1. In particular, the total number of conflicts between the edges of $\mathrm{CH}(P')$ and P may become quadratic, which makes the random incremental construction inefficient. Nevertheless, if the adversary is oblivious with respect to the sample, the points in P' behave as an unbiased sample of P.

2.3. Extensions and Variants

Diameter- and width-queries. Given $\mathrm{CH}(P)$, we can easily compute the *diameter* (i.e., a pair of points with maximum Euclidean distance) and the *width* of P (a strip of minimal width containing all the points in P) in linear time (see, e.g., [37, Chapter 4]). Hence,

Corollary 2.4. *Using $O(n^2)$ space and time, we can preprocess a set L of n lines in the plane, such that given a point set P with each point lying on a distinct line of L, the diameter or width of P can be found in expected time $O(n\alpha(n)\log^* n)$. The expected running time becomes $O(n\alpha(n))$ assuming obliviousness.*

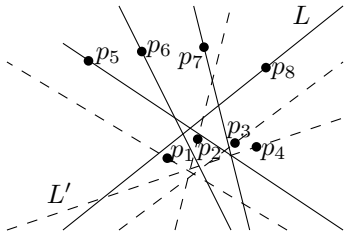

Figure 2: Illustrating the obliviousness assumption. The set L contains all the lines in the figure. The set L' of the dashed lines depicted in the figure is a random subset of L. If the adversary knows this random subset L', it can place the points $p_1, \ldots, p_8$ as illustrated in the figure, thereby constructing $\mathrm{CH}(p_1, \ldots, p_4)$ first. Then each edge on the upper hull $\mathrm{UH}(p_1, \ldots, p_4)$ is in conflict with each of the remaining points $p_5, \ldots, p_8$.

A trade-off between space and query time. Our data structure can be generalized to support a trade-off between preprocessing time (and storage) and the query time, using a relatively standard grouping technique [2, 9], described as follows.

Preprocessing. Let $1 \leq m \leq n$ be a parameter, and, without loss of generality, assume that n/m is an integer. We partition L into m subsets $L_1, \ldots, L_m$ of size n/m each, and construct the arrangements $\mathcal{A}(L_k)$, for $k = 1, \ldots, m$, in overall time and storage $O(n^2/m)$ (cf. [2, 9]).

Query. Given an exact input P, we first construct $\mathrm{CH}(P_k)$, where P_k is the subset of points on the lines in L_k, $k = 1, \ldots, m$, in $O((n/m)\alpha(n/m)\log^*(n/m))$ (assuming obliviousness it is $O((n/m)\alpha(n/m))$) expected time, for a total expected time of $O(n\alpha(n/m)\log^*(n/m))$ (resp., $O(n\alpha(n/m))$) over all these subsets. Having $\mathrm{CH}(P_k)$ at hand for all k, we merge $\mathrm{UH}(P_1), \ldots, \mathrm{UH}(P_m)$ in $O(n \log m)$ time [18], thereby producing a list Q of points sorted according to their x-order. We then use Graham's scheme to construct the upper hull of Q (and thus of P) in $O(|Q|)$ time. We produce the lower hull of P in an analogous manner. We have thus shown:

Corollary 2.5. *Fix $1 \leq m \leq n$. In total $O(n^2/m)$ time and space, we can preprocess a set L of n lines in the plane, such that given a point set P with each point lying on a distinct line of L, we can construct* $\mathrm{CH}(P)$ *in expected time $O(n(\log m + \alpha(n/m)\log^*(n/m)))$. The running time becomes $O(n(\log m + \alpha(n/m)))$ assuming obliviousness.*

Note that for small values of m, Corollary 2.5 in fact yields an improvement over Theorem 2.2. Specifically, by setting $m := 2^{\alpha(n)}$, we have that the space and preprocessing requirement in Theorem 2.2 can be lowered to $O(n^2/2^{\alpha(n)})$, while the expected query time remains $O(n\alpha(n)\log^*(n))$.

Discussion. As noted in the introduction, the bounds in Corollary 2.5 are somewhat disappointing. However, the study by Ali Abam and de Berg [2] might provide (albeit, weak) evidence that these bounds are unlikely to be improved. Indeed, they have studied the *kinetic sorting problem*, where we are given a set of n points moving continuously on the real line, and the goal is to maintain a structure on them so that at any given time the points can be sorted efficiently. Ali Abam and de Berg [2] showed that even when the trajectories of these points are just linear functions, then under the comparison graph model (see [2] for the definition) one cannot answer a query faster than $cn \log m$ time using less than $c'n^2/m$ preprocessing time and storage, for appropriate absolute constants $c, c' > 0$. As discussed in [2], this may indicate that better trade-offs for the kinetic convex hull problem seem unlikely. Nevertheless, it may not provide a rigorous proof, as the analysis for the kinetic sorting problem strongly relies on the one-dimensionality of the points, and does not work for points in the plane, at least under the context of the proofs given in [2]. Still, we have chosen to present those details in this paper, as we tend to believe that bounds of this kind could also apply to our problem (which is even more difficult than the kinetic convex hull problem, as described in the introduction), and that a rigorous analysis could stem from the approach in [2]. This would imply that the trade-off bounds given in Corollary 2.5 are nearly optimal.

An output-sensitive algorithm. Our algorithm can be made sensitive to the size h of the convex hull by adapting a technique of Ali Abam and de Berg [2] that uses *gift wrapping queries*. The setting for queries of this kind is as

follows. Let Q be a point set, given an arbitrary point p (not necessarily from Q) and a line ℓ through p, such that all points of Q lie on the same side of ℓ, report a point $q \in Q$ that is hit first when ℓ is rotated around p (say, in clockwise direction).

A search on the value of h. Since the output size h is not given in advance, we perform a search on its actual value, over at most $\log^* n - 1$ iterations, in the query step, and apply all tested values at the preprocessing step, as described below. The tested values of h are chosen in the following manner. Let h_i be the value of h at the ith round. Initially, $h_1 = 1$, and put $h_i = 2^{(i-1)}$, for $i \geq 2$, where $2^{(\cdot)}$ is the power-tower function. We continue the search as long as $h_i \leq \log n$; let t be the number of rounds thus obtained. By construction $t \leq \log^* n - 1$. When $h_i > \log n$, we stop the search and resort to the bound in Theorem 2.2—see below.

Preprocessing. At each round $i = 1, \ldots, t$, we set a parameter m_i to be

$$m_i := \max\left\{1, \frac{n}{h_i \log h_i}\right\},$$

and partition L into m_i roughly equal subsets $L_1^{(i)}, \ldots, L_{m_i}^{(i)}$. We then proceed in a similar manner as described earlier for the trade-off between space and query time. That is, for each $k = 1, \ldots, m_i$, we construct the arrangement $\mathcal{A}(L_k^{(i)})$ in overall time and storage $O(n^2/m_i)$.

The total time and storage consumed over all rounds i is thus

$$O\left(n^2 + \sum_{i=1}^{t} n h_i \log h_i\right) = O(n^2),$$

since the sum over the rounds i is dominated by the last term, which is $O(n \log n \log\log n)$.

Query. Given a point set P with each point lying on a distinct line of L, we construct $\mathrm{CH}(P)$ in an output-sensitive manner, as follows.

At the ith round, let $P_k^{(i)} := P \cap L_k^{(i)}$, for $k = 1, \ldots, m_i$. Construct $\mathrm{CH}(P_1^{(i)})$, $\mathrm{CH}(P_2^{(i)})$, …, $\mathrm{CH}(P_{m_i}^{(i)})$, as in Section 2.1. This takes total time $O(n\alpha(n/m_i) \log^*(n/m_i)) = O(n\alpha(h_i) \log^*(h_i))$ (resp. $O(n\alpha(n/m_i)) = O(n\alpha(h_i))$ assuming obliviousness).

The primitive operation we would like to obtain is a gift wrapping query on P. To this end, we perform standard gift wrapping queries for each subset $P_k^{(i)}$ in $O(\log(n/m_i))$ time (see, e.g., [37]). This yields a set of m_i candidates, from which we produce the final answer to the query. In total, a gift wrapping query takes $O(m_i \log(n/m_i)) = O(n/h_i)$ steps.

We now attempt to construct $\mathrm{CH}(P)$. We begin with a gift wrapping query for the leftmost vertex p of P and the vertical line ℓ_p passing through p. This yields a pair (p', ℓ'), where p' is the first point hit by ℓ, and ℓ' is the line through p and p'. We continue until (i) we hit p again, or (ii) we have performed h_i gift wrapping queries. This results in a running time of

$$O(h_i \cdot m_i \log(n/m_i)) = O\left(h_i \cdot \frac{n}{h_i}\right) = O(n).$$

The round *succeeds* if we reach p. Otherwise it fails, and we proceed to round $i + 1$. After $t \leq \log^* n - 1$ unsuccessful rounds (i.e., if $h_i > \log n$), we compute $\mathrm{CH}(P)$ directly via Theorem 2.2.

It is easy to verify that the actual number of rounds that we need is at most $O(\log^* h)$. Combining the bounds above, it follows that the overall query time is $O(n\alpha(h)(\log^* h)^2)$ (or $O(n\alpha(h) \log^* h)$ assuming obliviousness), as asserted. We have thus shown:

Corollary 2.6. *In total $O(n^2)$ time and space, we can preprocess a set L of n lines in the plane, such that given a point set P with each point lying on a distinct line of L, $\mathrm{CH}(P)$ can be found in expected time $O(n\alpha(h)(\log^* h)^2)$, where h is the output size. The expected running time becomes $O(n\alpha(h) \log^* h)$ assuming obliviousness.*

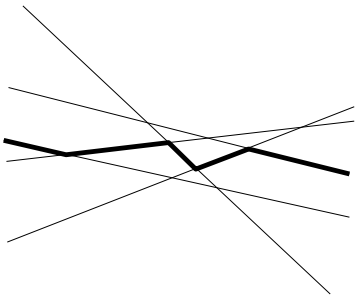

Figure 3: The 2-level in an arrangement of lines.

3. Levels in Arrangements

Preliminaries. Let L be a set of n lines in the plane (in general position). Given a point p, the *level* of p with respect to L is the number of lines in L intersected by the open downward vertical ray emanating from p. For an integer $k \geq 0$, the *k-level* of the arrangement $\mathcal{A}(L)$, denoted by $\mathrm{lev}_k(L)$, is the closure of all edges of $\mathcal{A}(L)$ whose interior points have level k with respect to L. It is a monotone piecewise-linear chain. In particular, $\mathrm{lev}_0(L)$ is the so-called "lower envelope" of L; see, e.g., [41, Chapter 5.4] and Figure 3. The ($\leq k$)*-level* of $\mathcal{A}(L)$, denoted by $\mathrm{lev}_{\leq k}(L)$, is the complex induced by all cells of $\mathcal{A}(L)$ lying on or below the k-level, and thus its edge set is the union of $\mathrm{lev}_i(L)$ for $i = 0, \ldots, k$; its overall combinatorial complexity is $O(nk)$ (see, e.g., [16, 41]).

In what follows we denote by $V_q(M)$ (resp., $V_{\leq q}(M)$) the set of vertices of $\mathrm{lev}_q(M)$ (resp., $\mathrm{lev}_{\leq q}(M)$), where $q \geq 0$ is an integer parameter and M is a set of lines in the plane. It is easy to verify that the combinatorial complexity of $\mathrm{lev}_q(M)$ is at most $O(1 + |V_q(M)|)$ (see once again [41]). Throughout this section, we use the *Vinogradov*-notation: $f \ll g$ means $f = O(g)$ and $f \gg g$ means $f = \Omega(g)$. In addition, we write $\mathbf{Exp}_X[\cdot]$ to emphasize that we take the expectation with respect to the random choice of X (the other variables are considered constant).

The best currently known bound for the worst-case complexity of $\mathrm{lev}_k(L)$ is $O(nk^{1/3})$ [21]. Nevertheless, since the overall combinatorial complexity of, say, $\mathrm{lev}_{\leq 2k}(L)$ is only $O(nk)$ [16], it follows that the *average* size of $V_i(L)$, for each $i = k, \ldots, 2k$, is only $O(n)$. Specifically, we have (see also [23] for a similar property):

Claim 3.1. *Let $\tilde{k}$ be a random integer in the range $\{k, \ldots, 2k\}$. Then, for any subset $S \subseteq L$, we have*

$$\mathbf{Exp}_{\tilde{k}}[|V_{\tilde{k}}(S)|] \ll |S|.$$

Proof. The claim follows from the observation that the total size of $V_{\leq 2k}(S)$ is $O(|S|k)$, and each vertex appears in exactly two consecutive levels of $\mathcal{A}(S)$. □

The problem. In the sequel we study the following problem. We are given a set $P = \{p_1, \ldots, p_n\}$ of n points in the plane (in general position), and we would like to compute a data structure such that, given any set $L = \{\ell_1, \ldots, \ell_n\}$ of n lines satisfying $p_i \in \ell_i$, for $i = 1, \ldots, n$, and any parameter $k \geq 0$, we can efficiently construct $\mathrm{lev}_{\leq k}(L)$. This is a natural generalization of the problem studied in Section 2.1. Indeed, let us apply the standard duality transformation, where a line $l : \ y = ax + b$ is mapped to the point $l^* = (a, -b)$, and a point $p = (c, d)$ is mapped to the line $p^* : \ y = cx - d$ (see, e.g., [6, Chapter 8]). Then $\mathrm{lev}_0(L)$ in the "primal" plane is mapped to the (upper) convex hull of the points L^* in the "dual" plane. Everett *et al.* [23] showed that $\mathrm{lev}_{\leq k}(L)$ can be constructed in $O(n \log n + nk)$ time, and that this time bound is worst-case optimal (see also [10]). We show:

Theorem 3.2. *Using $O(n^2)$ space and time, we can preprocess a set P of n points in the plane, such that given a set L of lines with each line incident to a distinct point of P, $\mathrm{lev}_{\leq k}(L)$ can be computed in expected time $O(n\alpha(n)(\log^* n - \log^* k) + nk)$. The expected running time becomes $O(n\alpha(n) + nk)$ assuming obliviousness.*

Theorem 3.2 improves the "standard" bound of $O(n \log n + nk)$ for any $k = o(\log n)$. We combine ideas from Chan's algorithm for constructing ($\leq k$)-levels in arrangements of planes in $\mathbb{R}^3$ [10] with the technique of Everett *et al.* [23]. The preprocessing phase is fairly simple, but the details of the query processing and its analysis are more intricate. We begin with an overview of the approach, and then describe the query step and its analysis in more detail.

An overview of the algorithm. The main ingredients of the algorithm are as follows.

Preprocessing. Compute the arrangement $\mathcal{A}(P^*)$ of the lines dual to the points in P (and produce its vertical decomposition) in $O(n^2)$ time and storage.

Query. We are given a set of lines L as above, and an integer $k \geq 0$. If $k \geq \log n$ we use the algorithm of Everett *et al.* [23] to report $\mathrm{lev}_{\leq k}(L)$ in $O(n \log n + nk) = O(nk)$ time. Otherwise, we compute a gradation $L_1 \subseteq L_2 \subseteq \cdots \subseteq L_{\log^* n - \log^* k + 1} \subseteq L$ of L. The sizes of the subsets L_i are similar to those presented in Section 2.1 for the dual plane, but as soon as the number of lines in a subset of the gradation exceeds $\lceil n/k \rceil$, we complete the sequence in a single step by choosing the next subset to be the entire set L. As in Section 2.1, we set $|L_1| := \lceil n/\log n \rceil$.

We choose a random integer $\tilde{k} \in \{k, \ldots, 2k\}$. Then, at the first iteration, we construct $\mathrm{lev}_{\leq \tilde{k}}(L_1)$ in $O(nk)$ time, using the algorithm in [23]. At each of the following iterations i, we construct $\mathrm{lev}_{\leq \tilde{k}}(L_i)$ from $\mathrm{lev}_{\leq \tilde{k}}(L_{i-1})$ (at the final step, we construct $\mathrm{lev}_{\leq \tilde{k}}(L)$ from $\mathrm{lev}_{\leq \tilde{k}}(L_{\log^* n - \log^* k + 1})$). As observed above, the random choice of $\tilde{k}$ guarantees that the expected complexity of each $\mathrm{lev}_{\tilde{k}}(L_i)$ is only linear in $|L_i|$, [6] for each $i = 2, \ldots, \log^* n - \log^* k + 1$, which is crucial for the analysis. Finally, we eliminate from $\mathrm{lev}_{\leq \tilde{k}}(L)$ all portions lying above the (actual) k-level, in order to obtain the final structure $\mathrm{lev}_{\leq k}(L)$.

To construct $\mathrm{lev}_{\leq \tilde{k}}(L_i)$ from $\mathrm{lev}_{\leq \tilde{k}}(L_{i-1})$, we would like to proceed as follows. We compute $\mathrm{UH}(\mathrm{lev}_{\leq \tilde{k}}(L_{i-1}))$ and subdivide it into semi-unbounded (in the negative y-direction) trapezoidal cells. The first goal is to find for each such cell Δ the set of lines $C_\Delta \subseteq L_i$ which are in *conflict* with Δ (that is, $\Delta \cap \ell \neq \emptyset$, for each $\ell \in C_\Delta$). This goal is achieved by mapping $\mathrm{UH}(\mathrm{lev}_{\leq \tilde{k}}(L_{i-1}))$ to the dual plane and walking along its zone in $\mathcal{A}(P^*)$. The dual of $\mathrm{UH}(\mathrm{lev}_{\leq \tilde{k}}(S))$ is a concave chain γ (the lower envelope of the lines dual to the vertices of $\mathrm{UH}(\mathrm{lev}_{\leq \tilde{k}}(S))$), where each vertex v of $\mathrm{UH}(\mathrm{lev}_{\leq \tilde{k}}(S))$ is mapped to an edge v^* of γ and each edge e is mapped to a vertex e^* of γ. Moreover, a line $\ell \in L$ below a vertex v of $\mathrm{UH}(\mathrm{lev}_{\leq \tilde{k}}(S))$ is mapped to a point ℓ^* (on some line of P^*) above the corresponding edge v^* of γ. As is easily verified, such a line ℓ intersects $\mathrm{UH}(\mathrm{lev}_{\leq \tilde{k}}(S))$. Otherwise, if ℓ lies above all the vertices of $\mathrm{UH}(\mathrm{lev}_{\leq \tilde{k}}(S))$, then $\ell \cap \mathrm{UH}(\mathrm{lev}_{\leq \tilde{k}}(S)) = \emptyset$, and this implies that ℓ^* lies below γ in the dual plane. See Figure 4(a)–(b).

Having the lists C_Δ at hand, we construct for each Δ the structure $\mathrm{lev}_{\leq \tilde{k}}(L_i)$ clipped to Δ by (i) constructing $\mathrm{lev}_{\tilde{k}}(C_\Delta)$ (clipped to Δ); (ii) clipping each line $\ell \in C_\Delta$ to its portion that lies below $\mathrm{UH}(\mathrm{lev}_{\tilde{k}}(C_\Delta) \cap \Delta)$; (iii) constructing the arrangement of these portions within Δ (as observed in [23], the actual level of these portions in $\mathcal{A}(L_i)$ does not exceed $2\tilde{k} - 1$); and (iv) eliminating from the arrangement just computed all portions lying above $\mathrm{lev}_{\tilde{k}}(C_\Delta) \cap \Delta$. Finally, we glue the resulting structures together and report $\mathrm{lev}_{\leq \tilde{k}}(L_i)$.

However, it would be too expensive to process each conflict list C_Δ individually. Therefore, a crucial ingredient of the algorithm is to consider *blocks* instead of just individual cells. Specifically, we gather contiguous cells into blocks and process them all together. This partition is the key to reducing the number of cells considered in the update step; see Figure 5. The bulk of the analysis lies in a careful balancing between the block sizes and their overall number, and in particular showing that blocks with large conflict lists are scarce.

3.1. Query Processing

We now describe the query process and its analysis in more detail. We first follow a gradation as described in the overview, and then proceed to the update step.

The update step. From now on we fix an iteration $i > 1$, and, with a slight abuse of notation, put $S := L_{i-1}$ and $L := L_i$. Let $p := |S|/|L|$. By definition, S is a random sample of L of size $\lceil n/\log^{(i-1)} n \rceil = p|L|$. Given $\mathrm{lev}_{\leq \tilde{k}}(S)$, we first construct $\mathrm{UH}(\mathrm{lev}_{\leq \tilde{k}}(S))$.

Claim 3.3. *The overall expected time to construct* $\mathrm{UH}(\mathrm{lev}_{\leq \tilde{k}}(S))$ *is* $O(|S|)$.

Proof. Using easy manipulations on the DCEL representing $\mathrm{lev}_{\leq \tilde{k}}(S)$, we can first locate a vertex v of $\mathrm{lev}_{\tilde{k}}(S)$, and then proceed to its neighboring topmost vertex (say, to its left) by walking along its corresponding adjacent edge. We then continue progressing in this manner to the left. The vertices to the right of v are explored analogously. Thus we can extract the sequence of vertices (and edges) along $\mathrm{lev}_{\tilde{k}}(S)$, ordered from left to right. By Claim 3.1, its expected size is $O(|S|)$. Then we use Graham's scan on the resulting set of vertices. □

[6] We use the same value of $\tilde{k}$ throughout the entire process, since the expected complexity of the $\tilde{k}$-level remains linear in each iteration i. By linearity of expectation, the overall expected size of the various $\tilde{k}$-levels is linear in $\sum_{i=2}^{\log^* n - \log^* k + 1} |L_i|$.

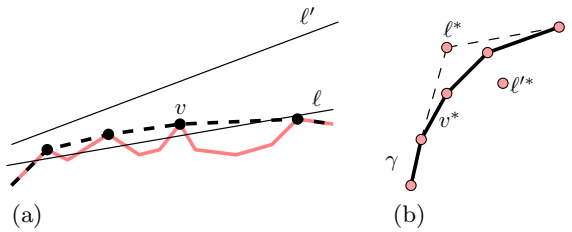

Figure 4: (a) The $\tilde{k}$-level of $\mathcal{A}(S)$ is depicted by the lightly-shaded polygonal line in the figure, and $\mathrm{UH}(\mathrm{lev}_{\leq\tilde{k}}(S))$ is depicted by the dashed line. The line ℓ' passes above $\mathrm{UH}(\mathrm{lev}_{\leq\tilde{k}}(S))$, where ℓ passes below v and thus meets $\mathrm{UH}(\mathrm{lev}_{\leq\tilde{k}}(S))$. (b) The dual scene of (a). The concave chain γ is the dual of $\mathrm{UH}(\mathrm{lev}_{\leq\tilde{k}}(S))$. The line ℓ is mapped to the point ℓ^*, where the pair of the dashed lines depict the visibility lines of ℓ^* to γ. The line ℓ' is mapped to the point ℓ'^* lying below γ.

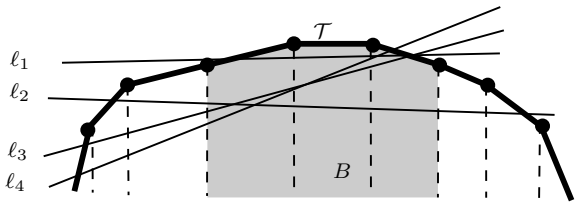

Figure 5: The block B is depicted by the lightly-shaded region. The line ℓ_1 does not intersect any neighboring block, whereas ℓ_3 and ℓ_4 also meet the left neighbor of B. The line ℓ_2 meets both neighbors.

Next, we shoot vertical rays from each vertex of the hull $\mathrm{UH}(\mathrm{lev}_{\leq\tilde{k}}(S))$ in the negative y-direction. This results in a collection $\mathcal{T}_{\tilde{k},S}$ of semi-unbounded trapezoidal cells covering $\mathrm{UH}(\mathrm{lev}_{\leq\tilde{k}}(S))$, and hence also $\mathrm{lev}_{\leq\tilde{k}}(L)$, as is easily verified (see, e.g., [35] for similar arguments). We group the cells in $\mathcal{T}_{\tilde{k},S}$ into $O(|\mathcal{T}_{\tilde{k},S}|(p/\tilde{k}))$ semi-unbounded vertical strips, each of which consists of $\tilde{k}/p$ contiguous cells. Such a vertical strip is called a *block*. Every block is bounded by a convex chain from above, and by two vertical walls, one to its left and the other to its right. Let $\mathcal{B}$ be the set of all blocks.

We say that a line $\ell \in L$ is *in conflict* with a cell $\Delta \in \mathcal{T}_{\tilde{k},S}$, if $\Delta \cap \ell \neq \emptyset$. The conflict list C_Δ is then the set of all lines $\ell \in L$ in conflict with Δ, and we put $c_\Delta := |C_\Delta|$. We similarly define conflict lists C_B and conflict sizes c_B for each block $B \in \mathcal{B}$. Our next goal is to determine the conflict lists C_B for each block.

Lemma 3.4. *We can construct the conflict lists C_B, $B \in \mathcal{B}$, in overall time $O(n\alpha(n) + |V_{\tilde{k}}(S)| + \sum_{B\in\mathcal{B}} c_B)$.*

Proof. First, we determine for each line $\ell \in L$ one trapezoid $\Delta_\ell \in \mathcal{T}_{\tilde{k},S}$ such that $\ell \in C_{\Delta_\ell}$, if such a Δ_ℓ exists. This is done by a walk in the dual plane, as described in the overview above and illustrated in Figure 4. Specifically, we dualize $\mathrm{UH}(\mathrm{lev}_{\leq\tilde{k}}(S))$ to a concave chain γ. Using a similar technique as in Section 2.1, we walk along the zone of γ in $\mathcal{A}(P^*)$ in order to determine, for each point ℓ^* corresponding to a line $\ell \in L$, its orientation with respect to γ. When ℓ^* lies above γ, we find an edge $v^*_{\ell^*}$ of γ that is visible from ℓ^*. Using the corresponding vertex v_ℓ in the primal plane, we can determine a cell Δ_ℓ that is intersected by ℓ.

Next, we determine for each such line ℓ a block B that conflicts with it, namely the block that contains Δ_ℓ. We then find all blocks B' with $\ell \in C_{B'}$ through a bidirectional walk from B. That is, we can determine if ℓ intersects the next block by checking whether ℓ intersects any of its walls (otherwise, ℓ intersects its convex chain). See Figure 5.

The bound on the running time now follows using similar considerations as in Section 2.1. □

Our next goal is to determine the $(\leq \tilde{k})$-level clipped to B, for each $B \in \mathcal{B}$. To this end, we use a variant of the technique of Everett *et al.* [23].

Lemma 3.5. *Let $B \in \mathcal{B}$. The $(\leq \tilde{k})$-level of L clipped to B can be constructed in time $O(c_B \log c_B + (m_B + c_B)\log^2 k + a_B)$, where $c_B := |C_B|$, $m_B := |V_{\tilde{k}}(C_B) \cap B|$, and a_B is the number of vertices of $\mathcal{A}(L)$ below $\mathrm{UH}(\mathrm{lev}_{\tilde{k}}(C_R) \cap B)$.*

Proof. We apply the algorithm of Cole *et al.* [17] in order to construct $\mathrm{lev}_{\tilde{k}}(C_B) \cap B$ in time $O(c_B \log c_B + (m_B + c_B)\log^2 \tilde{k})$. Note that this algorithm returns $\mathrm{lev}_{\tilde{k}}(C_B) \cap B$ as an x-monotone polygonal chain ζ ordered from left to right.[7] Next, we determine for each line $\ell \in C_B$ its first and last intersections w_1, w_2 with ζ (if they exist). Clearly, the portion of ℓ below $\mathrm{UH}(\zeta)$ is either (i) the line segment w_1w_2 (if both intersections exist); (ii) a ray with an endpoint at w_1 (if w_1 is the only intersection with $\mathrm{UH}(\zeta)$) or (iii) the full line ℓ clipped to B (if it lies fully below ζ).

These intersections can easily be determined in $O(m_B)$ time by walking along ζ and recording for each line ℓ the first and last vertices of ζ that are incident to ℓ (if they exist); at the representation of ζ, we also store the incident lines within each vertex. A line that is not encountered during this process, does not meet ζ, and we can easily check whether it lies below ζ. As observed above, each of these portions (clipped to B) is either the (full) line ℓ, a ray, or a line segment. Let C'_B be the resulting set of these portions; by construction, $c'_B := |C'_B| \leq c_B$. Having this collection at hand, the computation of $\mathrm{lev}_{\leq\tilde{k}}(C_B) \cap B$ is almost straightforward. Indeed, we use an optimal line segment intersection algorithm [12, 16] in order to compute the arrangement of C'_B in time proportional to $c'_B \log c'_B + a_B \ll c_B \log c_B + a_B$, where a_B is the number of intersections between the elements of C'_B. Note that some of these intersections may lie above the $\tilde{k}$-level, as they are only guaranteed to be contained in $\mathrm{UH}(\zeta)$. Thus, at the final step of the construction we eliminate such portions of the arrangement. This produces $\mathrm{lev}_{\leq\tilde{k}}(C_B) \cap B = \mathrm{lev}_{\leq\tilde{k}}(L) \cap B$. A key observation is the fact that all these portions are actually contained in $\mathrm{lev}_{\leq(2\tilde{k}-1)}(C_B) \cap B$—see below. □

Finally, we glue all the resulting structures together and report $\mathrm{lev}_{\leq\tilde{k}}(L)$.

3.2. The Analysis

We phrase our analysis below for a random subset S of L with $|S| = p|L|$, for some $p \in (0, 1)$, as the value of p varies at each iteration of the algorithm (as well as the final step). We begin with the following key lemma that bounds the total size of the large conflict sets. The proof is postponed to Appendix A:

Lemma 3.6. *Let $\tilde{k}$, L, S, $\mathcal{T}_{\tilde{k},S}$, C_Δ, c_Δ, p be defined as above. Then, for any sufficiently large constant $\beta \geq 1$, we have*

$$\mathbf{Exp}_S\left[\sum_{\substack{\Delta\in\mathcal{T}_{\tilde{k},S}\\ c_\Delta\geq\beta\tilde{k}/p^2}} c_\Delta(\log c_\Delta + \log^2\tilde{k} + \log(1/p))\right] \leq |L|e^{-\Theta(\beta)(\tilde{k}/p)}. \tag{2}$$

Remarks. (1) The bound in Lemma 3.6 holds for any integer $\tilde{k} \in \{k \ldots 2k\}$. In particular, the analysis does not assume a linear complexity bound on any of the levels of $\mathcal{A}(S)$ (and $\mathcal{A}(L)$); see Appendix A for further details.
(2) It is easy to verify that the bound in Lemma 3.6 can be rewritten when we apply the summation over all blocks. That is,

$$\mathbf{Exp}_S\left[\sum_{\substack{B\in\mathcal{B}\\ \Delta\in B, c_\Delta\geq\beta\tilde{k}/p^2}} c_\Delta(\log c_\Delta + \log^2\tilde{k} + \log(1/p))\right] \leq |L|e^{-\Theta(\beta)(\tilde{k}/p)}. \tag{3}$$

Bounding the expected running time. Adding the bounds in Lemmas 3.4 and 3.5, the running time to construct $\mathrm{lev}_{\leq\tilde{k}}(L)$ from $\mathrm{lev}_{\leq\tilde{k}}(S)$ is asymptotically upper-bounded by

$$n\alpha(n) + |V_{\tilde{k}}(S)| + \sum_{B\in\mathcal{B}} c_B(\log c_B + \log^2 k) + m_B \log^2 k + a_B \tag{4}$$

We bound each summand in turn. By Claim 3.1, we have $\mathbf{Exp}_{\tilde{k}}[|V_{\tilde{k}}(S)|] \ll |S| \leq |L|$.

Claim 3.7. *We have:*

$$\mathbf{Exp}_{S,\tilde{k}}\left[\sum_{B\in\mathcal{B}} c_B(\log c_B + \log^2\tilde{k})\right] \ll |L|\left(\log(1/p) + \log^2 k\right).$$

[7] The algorithm of Cole *et al.* proceeds with a rotational sweep in the dual plane that keeps $\tilde{k}$ points of the input to the left of the sweep-line. In order to find only those vertices of the $\tilde{k}$-level which lie inside B, we need to identify the appropriate initial orientation for this line, but this is easily done by inspecting the intersection of C_B with the left boundary of B and using a linear time selection algorithm [18, Chapter 9].

Proof. We say that a line $\ell \in C_B$ is *spanning* for a block $B \in \mathcal{B}$ if ℓ intersects both the left and the right walls of B, otherwise it is non-spanning. Note that every line can be non-spanning for at most two blocks. Let $\beta \geq 1$ be a sufficiently large constant. We say that a block $B \in \mathcal{B}$ is *light* if it has at most $\beta\tilde{k}/p^2$ spanning lines and if $c_B \leq \beta\tilde{k}^2/p^3$. Otherwise, B is called *heavy*.

We split the summation, as follows:

$$\sum_{B\in\mathcal{B}} c_B\left(\log c_B + \log^2 \tilde{k}\right) = \sum_{\substack{B\in\mathcal{B}\\ B \text{ is light}}} c_B\left(\log c_B + \log^2 \tilde{k}\right) + \sum_{\substack{B\in\mathcal{B}\\ B \text{ is heavy}}} c_B\left(\log c_B + \log^2 \tilde{k}\right). \tag{5}$$

Let us first consider the sum over the light blocks. Let B be a light block. By definition, we have $\log c_B \ll \log(1/p) + \log k$. Furthermore, write $c_B = c_B^s + c_B^n$, where c_B^s is the number of spanning lines in C_B, and c_B^n is the number of non-spanning lines in C_B. Observe that

$$\sum_{\substack{B\in\mathcal{B}\\ B \text{ is light}}} c_B^s \ll |\mathcal{B}|(\tilde{k}/p^2),$$

since each light block can have only $O(\tilde{k}/p^2)$ spanning lines, and that

$$\sum_{\substack{B\in\mathcal{B}\\ B \text{ is light}}} c_B^n \ll |L|,$$

since the total number of non-spanning lines is at most $2|L|$, over all blocks in $\mathcal{B}$. It follows that

$$\begin{aligned}\sum_{\substack{B\in\mathcal{B}\\ B \text{ is light}}} c_B\left(\log c_B + \log^2 \tilde{k}\right) &\ll \left(\sum_{\substack{B\in\mathcal{B}\\ B \text{ is light}}} c_B^s + c_B^n\right)\left(\log(1/p) + \log^2 \tilde{k}\right)\\ &\ll \left(|\mathcal{B}|(\tilde{k}/p^2) + |L|\right)\left(\log(1/p) + \log^2 \tilde{k}\right).\end{aligned}$$

Now we bound $|\mathcal{B}|(\tilde{k}/p^2)$. Since each block contains $p/\tilde{k}$ contiguous cells, we have $|\mathcal{B}| \ll |\mathcal{T}_{\tilde{k},S}|(p/\tilde{k})$. Furthermore, because the number of cells is bounded by the number of vertices on the $\tilde{k}$-level, we get

$$\mathbf{Exp}_{S,\tilde{k}}[|\mathcal{T}_{\tilde{k},S}|] \ll \mathbf{Exp}_{S,\tilde{k}}[|V_{\tilde{k}}(S)|] \ll \mathbf{Exp}_S[|S|] = |L|p,$$

using Claim 3.1 and the definition of p as $|S|/|L|$. Thus,

$$\mathbf{Exp}_{S,\tilde{k}}[|\mathcal{B}|(\tilde{k}/p^2)] \ll \mathbf{Exp}_{S,\tilde{k}}[|\mathcal{T}_{\tilde{k},S}|/p] \ll |L|.$$

Therefore,

$$\mathbf{Exp}_{S,\tilde{k}}\left[\sum_{\substack{B\in\mathcal{B}\\ B \text{ is light}}} c_B(\log c_B + \log^2 \tilde{k})\right] \ll |L|\left(\log(1/p) + \log^2 k\right).$$

To bound the sum over the heavy blocks in (5), observe that by definition a heavy block B must contain a cell Δ with $c_\Delta > \beta\tilde{k}/p^2$: either there are more than $\beta\tilde{k}/p^2$ spanning lines, in which case all the cells in B have this property, or $c_B > \beta\tilde{k}^2/p^3$, in which case the claim follows from the fact that B contains only $\tilde{k}/p$ cells. Let $\Delta^* \in B$ be the cell that maximizes c_Δ for $\Delta \in B$. Clearly, we have $c_{\Delta^*} > \beta\tilde{k}/p^2$ and $c_B \leq (\tilde{k}/p)c_{\Delta^*}$. Hence,

$$\begin{aligned}\sum_{\substack{B\in\mathcal{B}\\ B \text{ is heavy}}} c_B\left(\log c_B + \log^2 \tilde{k}\right) &\ll \sum_{\substack{B\in\mathcal{B}\\ \Delta\in B, c_\Delta \geq \beta\tilde{k}/p^2}} (\tilde{k}/p)\cdot c_\Delta\left(\log(c_\Delta\tilde{k}/p) + \log^2 \tilde{k}\right)\\ &\ll (\tilde{k}/p) \sum_{\substack{B\in\mathcal{B}\\ \Delta\in B, c_\Delta \geq \beta\tilde{k}/p^2}} c_\Delta\left(\log c_\Delta + \log^2 \tilde{k} + \log(1/p)\right).\end{aligned}$$

By (3), the expectation (over S) of the latter sum is at most

$$(\tilde{k}/p)|L|e^{-\Theta(\beta)(\tilde{k}/p)} \ll |L|,$$

for β sufficiently large. □

Remark. In the analysis of Lemma 3.7 concerning the bound for the heavy blocks, each such block B may consist of both heavy cells (that is, cells Δ with $c_\Delta \geq \beta\tilde{k}/p^2$) and light ones. At first glance, one may suspect that the overall contribution of the light cells should have the bound $O(|L|(\log(1/p) + \log^2 k))$, as obtained in the case for light blocks. Nevertheless, since these cells belong to a heavy block, the actual bound is smaller, and in fact follows from the property that the number of heavy blocks is eventually much smaller than the number of light blocks (this property is an easy consequence of Lemma 3.6).

Claim 3.8. *We have:* $\sum_{B\in\mathcal{B}} m_B \log^2 \tilde{k} \ll |L|k$.

Proof. Recall that $m_B = |V_{\tilde{k}}(C_B) \cap B| = |V_{\tilde{k}}(L) \cap B|$. Using Dey's bound on the size of the $\tilde{k}$-level [21], it follows that $\sum_{B\in\mathcal{B}} m_B = |V_{\tilde{k}}(L)| \ll |L|k^{1/3}$. The claim is now immediate. □

Claim 3.9. *We have:* $\sum_{B\in\mathcal{B}} a_B \ll |L|k$.

Proof. Everett *et al.* [23] showed that no element in C'_B contains a point which lies above $\mathrm{lev}_{2\tilde{k}-1}(C_B) \cap B$. Since all sets C'_B are clipped to B, for each $B \in \mathcal{B}$, it follows that all portions of the various arrangements that we construct, over all $B \in \mathcal{B}$, lie within $\mathrm{lev}_{\leq 2\tilde{k}-1}(L)$. Hence,

$$\sum_{B\in\mathcal{B}} a_B \ll |L|\tilde{k} \ll |L|k.$$

□

We thus conclude:

Corollary 3.10. *The total expected running time for the ith iteration is*

$$O\left(n\alpha(n) + |L_i|\left(k + \log\left(\frac{|L_i|}{|L_{i-1}|}\right)\right)\right).$$

Proof. This follows by substituting the bounds from Claims 3.7–3.9 into (4), by using that $\log^2 k = O(k)$, and by remembering that we set $L = L_i$, $S = L_{i-1}$ and $p = |S|/|L|$. □

Note that

$$\sum_{i=2}^{\log^* n - \log^* k + 1} |L_i|k = \sum_{i=2}^{\log^* n - \log^* k + 1} nk/\log^{(i)} n \ll nk,$$

since the sequence $\{1/\log^{(i)} n\}_{i=\log^* n - \log^* k}^{2}$ decreases faster than any geometric sequence. Moreover, for all but the last iteration, we have $|L_i|\log(|L_i|/|L_{i-1}|) \ll |L_i|\log^{(i)} n \ll n$. At the last iteration, we have $|S| \geq n/k$, so $\log(|L|/|S|) \leq \log k$, and thus

$$\sum_{i=2}^{\log^* n - \log^* k + 1} |L_i|\log^{(i)} n + |L|\log k \ll n(\log^* n - \log^* k + \log k).$$

It thus follows that the overall expected running time is $O(n\alpha(n)(\log^* n - \log^* k) + nk)$. It is easy to verify that when $k = 0$ we obtain the same asymptotic time bound as in Theorem 2.2.

A faster algorithm under the obliviousness assumption. Similar to Section 2.2, the expected running time can be improved to $O(n(\alpha(n)+k))$ assuming obliviousness. As before, we now compute a gradation during the preprocessing phase: $P_1 \subseteq P_2 \subseteq \cdots \subseteq P_{1+\log\log n} = P$ with $|P_1| \ll n/\log n$ and $|P_i| = 2|P_{i-1}|$, and we compute each of the arrangements $\mathcal{A}(P_i^*)$ in the dual plane.

The algorithm for processing a set of lines L, with each line containing exactly one point, is just as above, with two major differences: first, we compute the gradation for L by using the precomputed gradation for P. Second, during the ith iteration we use $A(P_i^*)$ instead of $A(P^*)$ to determine the zone of γ. Using similar considerations as in Section 2.2, the bound in Corollary 3.10 now becomes $O(|L_i|(\alpha(n)+k))$, because $\log(|L_i|/|L_{i-1}|) = 1$. Summing over the various iterations i and the final step yields the bound $O(n(\alpha(n)+k))$, as asserted. The total storage requirement remains $O(n^2)$. This at last concludes the proof of Theorem 3.2.

4. Lower Bounds

In this section we study problems where preprocessing $\mathcal{A}(L)$ is unlikely to decrease the query time to $o(n\log n)$ (at least under some computational models).

Delaunay triangulations. It has already been observed in [8, 30] that for some sets L, even when we have $\mathcal{A}(L)$ precomputed, there are point sets, with each point lying on a distinct line, such that their Delaunay triangulation cannot be constructed in $o(n\log n)$ time (albeit sometimes one can obtain better bounds if each point lies on a fat region given in advance [8, 34]). This lower bound holds in the classic algebraic computation tree model [3, Chapter 16], and it essentially comes from a construction due to Djidjev and Lingas [22]. Specifically, they showed that when the points are sorted in just a single direction, one cannot compute their Delaunay triangulation in less than $\Omega(n\log n)$ time. Thus, if L is a set of vertical lines, we can only anticipate the x-order of the points (received later), from which the lower bound follows. Note that this lower bound also implies that no speedup is possible for computing the Euclidean minimum spanning tree (EMST), since the Delaunay triangulation can be constructed in linear time once the EMST is known [14, 29].

Closest Pairs. Finding the closest pair in a point set is somewhat easier than the Delaunay triangulation problem (since the latter has an edge between the closest pair [6]), but is often harder than computing convex hulls (except perhaps when the model of computation provides the floor function as well as a source of randomness, see, e.g., [27]). Formally, the problem is defined as follows: given a set $L = \{\ell_1, \ldots, \ell_n\}$ of lines in the plane, compute a data structure such that given any point set $P = \{p_1, \ldots, p_n\}$ with $p_i \in \ell_i$ for $i = 1, \ldots, n$, we can quickly find a pair $(p_i, p_j) \in P \times P$ of distinct points that minimizes $\|p_i - p_j\|$. Incorporating the lower bound by Djidjev and Lingas [22], we show the following:

Proposition 4.1. *There exists a set $L = \{\ell_1, \ldots, \ell_n\}$ of lines in the plane, such that for any point set P with each point lying on a distinct line of L, finding the closest pair in P (after preprocessing L) requires $\Omega(n\log n)$ operations under the algebraic computation tree model.*

Proof. Consider the problem Fuzzy-2-Separation: for a sequence $x_1, \ldots, x_n$ in $\mathbb{R}$, output No, if there exists a pair $1 \le i < j \le n$ with $|x_i - x_j| \le 1$, and Yes, if for each pair $1 \le i < j \le n$ we have $|x_i - x_j| \ge 2$. In all other cases the answer is arbitrary.

Claim 4.2. *Any algebraic decision tree for the problem* Fuzzy-2-Separation *has depth $\Omega(n\log n)$.*

Proof. This follows from a straightforward application of the technique of Ben-Or [5]. The only somewhat non-standard feature is the need to deal with fuzziness. Let

$$W_1 = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid \forall\, 1 \le i < j \le n : |x_i - x_j| > 1\},$$

and let

$$W_2 = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid \forall\, 1 \le i < j \le n : |x_i - x_j| \ge 2\}.$$

Let T be a decision tree for Fuzzy-2-Separation, and let $W = T^{-1}(\text{Yes}) \subseteq \mathbb{R}^n$ be the set of inputs that lead to a leaf in T labeled Yes. By definition, we have $W_1 \supseteq W \supseteq W_2$. It now follows that W has at least $n!$ different connected

components, since the $n!$ inputs $x_\pi = (2\pi(1), 2\pi(2), \ldots, 2\pi(n))$ for any permutation π of $\{1, \ldots, n\}$ are all contained in W_2 and reside in different connected components of W_1 (see [3, Theorem 16.20] for this standard technique). Hence, Ben-Or's result [5] implies that T has depth $\Omega(\log n!) = \Omega(n \log n)$. □

The reduction from Fuzzy-2-Separation to closest pair queries is almost straightforward. For $i = 1, \ldots, n$, let ℓ_i be the horizontal line $\ell_i : y = i/n$, and let $L = \{\ell_1, \ldots, \ell_n\}$. Thus, the only information we can precompute from L is exactly this order. Given an instance $(x_1, \ldots, x_n)$ of Fuzzy-2-Separation, we map each x_i to a point $p_i = (x_i, i/n) \in \ell_i$, and then find the closest pair in the resulting point set. If the distance of the closest pair is greater than 2, our algorithm outputs Yes, otherwise it outputs No. Clearly, the overhead for this reduction is linear. We are now left to show the correctness of the reduction. Indeed, if $|x_i - x_j| \geq 2$, for every pair of indices $1 \leq i < j \leq n$, then clearly $\|p_i - p_j\| \geq \sqrt{4 + 1/n^2} > 2$, and this in particular applies for the closest pair of points. Otherwise, if there exists a pair $1 \leq i < j \leq n$ with $|x_i - x_j| < 1$, then $\|p_i - p_j\| \leq \sqrt{1 + 1} < 2$ (and this also upper bounds the distance between the closest pair), so the reduction reports the correct answer on all mandatory Yes and No instances, as asserted. □

Convex hull in three dimensions. Returning to the convex hull problem, we next study its extension to three dimensions. That is, given a set $H = \{h_1, h_2, \ldots, h_n\}$ of n planes in $\mathbb{R}^3$, we would like to compute a data structure, so that for any point set $P = \{p_1, \ldots, p_n\}$ with $p_i \in h_i$, $i = 1, \ldots, n$, we can construct $\mathrm{CH}(P)$ quickly. Since the complexity of the convex hull in both $\mathbb{R}^2$ and $\mathbb{R}^3$ is only linear, and since there are several algorithms that construct the convex hull (in both cases) in the same asymptotic running time (see, e.g., [6, 16]), one may ask if a three-dimensional convex hull query can be answered in $o(n \log n)$ time as well. Using the well-known lifting transformation [41], one can quickly derive a lower bound from the result about Delaunay triangulations mentioned above, but below we also give simple direct reduction (which follows immediately from a result of Seidel [39]).

Proposition 4.3. *There is a set $H = \{h_1, \ldots, h_n\}$ of planes in $\mathbb{R}^3$, such that for any point set P with each point lying on a distinct plane of H, constructing $\mathrm{CH}(P)$ (after preprocessing H) requires $\Omega(n \log n)$ operations under the algebraic computation tree model.*

Proof. Let h_i be the plane defined by the equation $z = i$, for $i = 1, \ldots, n$, and let $H = \{h_1, \ldots, h_n\}$. We give a reduction from planar convex hulls to computing three-dimensional convex hulls of point sets, where each plane in H contains precisely one such point. Let $P = \{p_1, \ldots, p_n\}$ be a set of points in the plane, and, for $i = 1, \ldots n$, let $\widehat{p_i} := (p_{ix}, p_{iy}, i)$, that is, the point obtained by lifting p_i to h_i. As observed by Seidel [39, Section IV], to compute the planar convex hull $\mathrm{CH}(P)$, it suffices to perform a convex hull query for $\widehat{P} = \{\widehat{p_1}, \widehat{p_2}, \ldots, \widehat{p_n}\}$ and then project the result onto the xy-plane. It is shown in [39] that once we have $\mathrm{CH}(\widehat{P})$ at hand, the time to project it onto the xy-plane (and then extract the actual planar convex hull) is only $O(n)$. Thus the overhead of the reduction is linear, as is easily verified. The result now follows from the standard $\Omega(n \log n)$ lower bound for planar convex hulls in the algebraic computation tree model (see, e.g., [5]). □

Sorting. Interestingly, a similar approach also shows that *sorting* requires $\Omega(n \log n)$ operations under the algebraic computation tree model. We have a set $L = \{\ell_1, \ldots, \ell_n\}$ of n lines in the plane, and we wish to compute a data structure such that for any set $P = \{p_1, \ldots, p_n\}$ of points with $p_i \in \ell_i$, $i = 1, \ldots, n$, we can quickly sort these points according to their x-order.

Proposition 4.4. *There exists a set $L = \{\ell_1, \ldots, \ell_n\}$ of lines in the plane, such that for any point set P with each point lying on a distinct line of L, sorting P according to its x-order (after preprocessing L) requires $\Omega(n \log n)$ operations under the algebraic computation tree model.*

Proof. Let $X = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}$. For $i = 1, \ldots, n$, let ℓ_i be the line $\ell_i : y = i$, and let $L = \{\ell_1, \ldots, \ell_n\}$. We now lift each x_i on ℓ_i, and obtain the point $p_i := (x_i, i)$, $i = 1, \ldots, n$; let P denote this set of points. It is now easy to see that the x-order of P yields the sorted order for the numbers in X, and that this reduction has a linear running time. □

5. Concluding remarks

Note that Proposition 4.4, which has a straightforward proof, has an intriguing implication emphasizing a main contribution of this paper: while the "standard" planar convex hull and sorting problems are basically equivalent in

terms of hardness (e.g., [6]), in our setting convex hull queries are in fact *easier*. This improvement stems from the "output-sensitive nature" of convex hulls: points inside the hull are irrelevant to the computation, and the information provided by L, combined with our update technique, allows us to quickly discard those non-extremal points, and not further process them in following iterations. In our setting the two problems become equivalent if the input points are in convex position. Then, Proposition 4.4 does not apply, since the points are sorted along two directions, and having the order according to one of them immediately implies the order according to the other.

Our study raises several open problems. The first one is whether the $\log^* n$ factor in the query time bound is indeed necessary for both convex hull and $(\leq k)$-level queries. We conjecture it to be an artifact of the technique and that the actual running times are $O(n\alpha(n))$ and $O(n(\alpha(n)+k))$ for the two respective problems (as in the obliviousness model). Another problem concerns the case of convex hulls for points restricted to three-dimensional *lines*. In this case, the lower bound in Section 4 does not apply. Moreover, if the lines are *parallel*, a simple variant of our approach yields expected query time $O(n \log\log n)$ with polynomial preprocessing and storage. Is there a better bound? What happens in the general case?

Acknowledgments. The authors wish to thank Maarten Löffler for suggesting the problem and for interesting discussions, and Boris Aronov and Timothy Chan for helpful discussions.

We would like to thank the anonymous referees for their careful reading of the paper and for numerous insightful comments that improved the quality of the paper.

References

[1] P. Afshani, J. Barbay, T.M. Chan, Instance-optimal geometric algorithms, in: Proc. 50th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS), pp. 129–138.
[2] M. Ali Abam, M. de Berg, Kinetic sorting and kinetic convex hulls, Comput. Geom. Theory Appl. 37 (2007) 16–26.
[3] S. Arora, B. Barak, Computational complexity: A Modern Approach, Cambridge University Press, 2009.
[4] J. Basch, L.J. Guibas, J. Hershberger, Data structures for mobile data, J. Algorithms 31 (1999) 1–28.
[5] M. Ben-Or, Lower bounds for algebraic computation trees, in: Proc. 16th Annu. ACM Sympos. Theory Comput. (STOC), pp. 80–86.
[6] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational geometry: algorithms and applications, Springer-Verlag, Berlin, third edition, 2008.
[7] M. Bern, D. Eppstein, P. Plassmann, F. Yao, Horizon theorems for lines and polygons, in: Discrete and computational geometry (New Brunswick, NJ, 1989/1990), volume 6 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 1991, pp. 45–66.
[8] K. Buchin, M. Löffler, P. Morin, W. Mulzer, Preprocessing imprecise points for Delaunay triangulation: Simplified and extended, Algorithmica 61 (2011) 674–693.
[9] T.M. Chan, Output-sensitive results on convex hulls, extreme points, and related problems, Discrete Comput. Geom. 16 (1996) 369–387.
[10] T.M. Chan, Random sampling, halfspace range reporting, and construction of $(\leq k)$-levels in three dimensions, SIAM J. Comput. 30 (2000) 561–575.
[11] T.M. Chan, Dynamic coresets, Discrete Comput. Geom. 42 (2009) 469–488.
[12] B. Chazelle, H. Edelsbrunner, An optimal algorithm for intersecting line segments in the plane, J. ACM 39 (1992) 1–54.
[13] B. Chazelle, W. Mulzer, Computing hereditary convex structures, Discrete Comput. Geom. 45 (2011) 796–823.
[14] F. Chin, C.A. Wang, Finding the constrained Delaunay triangulation and constrained Voronoi diagram of a simple polygon in linear time, SIAM J. Comput. 28 (1999) 471–486.
[15] V. Chvátal, The tail of the hypergeometric distribution, Discrete Math. 25 (1979) 285–287.
[16] K.L. Clarkson, P.W. Shor, Applications of random sampling in computational geometry. II, Discrete Comput. Geom. 4 (1989) 387–421.
[17] R. Cole, M. Sharir, C.K. Yap, On k-hulls and related problems, SIAM J. Comput. 16 (1987) 61–77.
[18] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to algorithms, MIT Press, Cambridge, MA, third edition, 2009.
[19] O. Devillers, Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems, Internat. J. Comput. Geom. Appl. 2 (1992) 97–111.
[20] O. Devillers, Delaunay triangulation of imprecise points: Preprocess and actually get a fast query time, J. Comput. Geom. (JoCG) 2 (2011) 30–45.
[21] T.K. Dey, Improved bounds for planar k-sets and related problems, Discrete Comput. Geom. 19 (1998) 373–382.
[22] H. Djidjev, A. Lingas, On computing Voronoi diagrams for sorted point sets, Internat. J. Comput. Geom. Appl. 5 (1995) 327–337.
[23] H. Everett, J.M. Robert, M. Van Kreveld, An optimal algorithm for computing $(\leq K)$-levels, with applications, Internat. J. Comput. Geom. Appl. 6 (1996) 247–261.
[24] D. Guibas, L. Salesin, J. Stolfi, Epsilon geometry: building robust algorithms from imprecise computations, in: Proc. 5th Annu. ACM Sympos. Comput. Geom. (SoCG), pp. 208–217.
[25] M. Held, J.S.B. Mitchell, Triangulating input-constrained planar point sets, Inform. Process. Lett. 109 (2008) 54–56.
[26] W. Hoeffding, Probability inequalities for sums of bounded random variables, J. Amer. Statist. Assoc. 58 (1963) 13–30.
[27] S. Khuller, Y. Matias, A simple randomized sieve algorithm for the closest-pair problem, Inform. and Comput. 118 (1995) 34–37.
[28] D.G. Kirkpatrick, R. Seidel, The ultimate planar convex hull algorithm?, SIAM J. Comput. 15 (1986) 287–299.

[29] R. Klein, A. Lingas, A linear-time randomized algorithm for the bounded Voronoi diagram of a simple polygon, Internat. J. Comput. Geom. Appl. 6 (1996) 263–278.
[30] M.J. van Kreveld, M. Löffler, J.S.B. Mitchell, Preprocessing imprecise points and splitting triangulations, SIAM J. Comput. 39 (2010) 2990–3000.
[31] M. Löffler, Data Imprecision in Computational Geometry, Ph.D. thesis, Utrecht University, 2009.
[32] M. Löffler, W. Mulzer, Triangulating the square and squaring the triangle: quadtrees and Delaunay triangulations are equivalent, in: Proc. 22nd Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA), pp. 1759–1777.
[33] M. Löffler, J. Phillips, Shape fitting on point sets with probability distributions, in: Proc. 17th Annu. European Sympos. Algorithms (ESA), pp. 313–324.
[34] M. Löffler, J. Snoeyink, Delaunay triangulation of imprecise points in linear time after preprocessing, Comput. Geom. Theory Appl. 43 (2010) 234–242.
[35] J. Matoušek, Reporting points in halfspaces, Comput. Geom. Theory Appl. 2 (1992) 169–186.
[36] D. McCallum, D. Avis, A linear algorithm for finding the convex hull of a simple polygon, Inform. Process. Lett. 9 (1979) 201–206.
[37] F.P. Preparata, M.I. Shamos, Computational Geometry - An Introduction, Springer, 1985.
[38] E.A. Ramos, On range reporting, ray shooting and k-level construction, in: Proc. 15th Annu. ACM Sympos. Comput. Geom. (SoCG), pp. 390–399.
[39] R. Seidel, A Method for Proving Lower Bounds for Certain Geometric Problems, Technical Report TR84-592, Cornell University, Ithaca, NY, USA, 1984.
[40] R. Seidel, A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons, Comput. Geom. Theory Appl. 1 (1991) 51–64.
[41] M. Sharir, P.K. Agarwal, Davenport-Schinzel sequences and their geometric applications, Cambridge University Press, New York, NY, USA, 1995.

Appendix A. Levels in Arrangements

Proof of Lemma 3.6: We actually consider the sum over all $\Delta \in \mathcal{T}_{\tilde{k},S}$ with $c_\Delta \geq 2\beta'\tilde{k}/p$, where $\beta' > 1$ is a constant to be fixed shortly. Then the lemma follows by choosing $\beta \geq 2\beta'$. In what follows, with a slight abuse of notation, we denote β' by β. For every vertex v of $\mathcal{A}(L)$, let C_v be the set of lines intersecting the (open) downward vertical ray emanating from v, and put $c_v := |C_v|$. Every vertex of $\mathrm{lev}_{\tilde{k}}(S)$ bounds at most two cells in $\mathcal{T}_{\tilde{k},S}$, and for every $\Delta \in T_{\tilde{k},S}$ any line in C_Δ passes under at least one vertex of Δ. Thus, we have $c_\Delta \leq 2\max\{c_{v_1}, c_{v_2}\}$, where v_1, v_2 are the two vertices of Δ. We thus have:

$$\sum_{\substack{\Delta\in\mathcal{T}_{\tilde{k},S}\\ c_\Delta\geq 2\beta\tilde{k}/p^2}} c_\Delta(\log c_\Delta + \log^2\tilde{k} + \log(1/p)) \ll \sum_{\substack{v\in V_{\tilde{k}}(S)\\ c_v\geq\beta\tilde{k}/p^2}} c_v(\log c_v + \log^2\tilde{k} + \log(1/p)).$$

Now, let v be a vertex of $\mathcal{A}(L)$ at level $c_v \geq (\tilde{k}/p^2) - 1$, and let ℓ_1, ℓ_2 be the two lines defining v. The vertex v appears in $V_{\tilde{k}}(S)$ precisely if (i) ℓ_1 and ℓ_2 are in S; and (ii) S contains $\tilde{k} - 1$ or $\tilde{k}$ lines below v. Thus,

$$\begin{aligned}\Pr[v \in V_{\tilde{k}}(S)] &= \Pr[\{\ell_1,\ell_2\} \subseteq S \wedge |S \cap C_v| \in \{\tilde{k}-1, \tilde{k}\}] \\ &= \left(\binom{|L|-2}{p|L|-2}\Big/\binom{|L|}{p|L|}\right)\cdot \Pr\left[|S\cap C_v| \in \{\tilde{k}-1,\tilde{k}\} \mid \{\ell_1,\ell_2\}\subseteq S\right].\end{aligned}$$

Conditioned on S containing $\{\ell_1, \ell_2\}$, the sample $S' := S \setminus \{\ell_1, \ell_2\}$ is a random $(p|L| - 2)$-sample from the set $L' := L \setminus \{\ell_1, \ell_2\}$. Hence, $|S' \cap C_v|$ follows a hypergeometric distribution, so Hoeffding's bound [15, 26] implies that

$$\begin{aligned}\Pr[|S\cap C_v| \in \{\tilde{k}-1,\tilde{k}\} \mid \{\ell_1,\ell_2\}\subseteq S] &\leq \Pr[|S'\cap C_v|/|S'| \leq \tilde{k}/|S'|] \\ &= \Pr[|S'\cap \overline{C_v}|/|S'| \geq 1-\tilde{k}/|S'|] \leq \left(\left(\frac{1-c_v/|L'|}{1-\tilde{k}/|S'|}\right)^{1-\tilde{k}/|S'|}\left(\frac{c_v/|L'|}{\tilde{k}/|S'|}\right)^{\tilde{k}/|S'|}\right)^{|S'|},\end{aligned}$$

recalling that $c_v = |C_v|$ denotes the number of lines below v. Now note that

$$p|L'| = p(|L|-2) = |S| - 2p = |S'| + 2 - 2p.$$

Thus, writing $c_v = t\cdot\tilde{k}/p$, for some appropriate $t \geq \beta/p$, we get

$$\Pr[|S\cap C_v| \in \{\tilde{k}-1,\tilde{k}\} \mid \{\ell_1,\ell_2\}\subseteq S] \leq \left(\Big(\frac{1-t\tilde{k}/(|S'|+2(1-p))}{1-\tilde{k}/|S'|}\Big)^{1-\tilde{k}/|S'|}\cdot\Big(\frac{t\tilde{k}/(|S'|+2(1-p))}{\tilde{k}/|S'|}\Big)^{\tilde{k}/|S'|}\right)^{|S'|}.$$

To simplify this, we first observe that

$$1 - \frac{t\tilde{k}}{|S'| + 2(1-p)} \le 1 - \frac{t\tilde{k}}{|S'| + 2} \le 1 - \frac{t\tilde{k}}{|S'| + |S'|/3} = 1 - \frac{3t\tilde{k}}{4|S'|},$$

since we can assume S' is large enough so that $2 \le |S'|/3$. Therefore,

$$\frac{1 - t\tilde{k}/(|S'| + 2(1-p))}{1 - \tilde{k}/|S'|} \le \frac{1 - (3t/4)\tilde{k}/|S'|}{1 - \tilde{k}/|S'|} = 1 - \frac{(3t/4 - 1)\tilde{k}/|S'|}{1 - \tilde{k}/|S'|}.$$

For the other term, we calculate

$$\frac{t\tilde{k}/(|S'| + 2(1-p))}{\tilde{k}/|S'|} = t \cdot \frac{|S'|}{|S'| + 2(1-p)} \le t.$$

Therefore, we can bound the probability as

$$\begin{aligned}\Pr[|S \cap C_v| \in \{\tilde{k} - 1, \tilde{k}\} \mid \{\ell_1, \ell_2\} \subseteq S] &\le \left(\left(1 - \frac{(3t/4 - 1)\tilde{k}/|S'|}{1 - \tilde{k}/|S'|}\right)^{1 - \tilde{k}/|S'|} t^{\tilde{k}/|S'|}\right)^{|S'|} \\ &\le \exp\left(-(3t/4 - 1 - \log t)\tilde{k}\right) \le \exp\left(-t\tilde{k}/2\right),\end{aligned}$$

for $t \ge \beta/p$ large enough. We next observe that

$$\binom{|L| - 2}{p|L| - 2} \Big/ \binom{|L|}{p|L|} = \frac{(|L| - 2)!}{(p|L| - 2)!(|L| - p|L|)!} \cdot \frac{(p|L|)!(|L| - p|L|)!}{|L|!} = \frac{p|L|}{|L|} \cdot \frac{p|L| - 1}{|L| - 1} \le p^2$$

in order to conclude that

$$\Pr[v \in V_{\tilde{k}}(S)] = \left(\binom{|L| - 2}{p|L| - 2} \Big/ \binom{|L|}{p|L|}\right) \cdot \Pr\left[|S \cap C_v| \in \{\tilde{k} - 1, \tilde{k}\} \mid \{\ell_1, \ell_2\} \subseteq S\right] \le p^2 \cdot \exp\left(-t\tilde{k}/2\right). \tag{A.1}$$

Now we can finally bound the expectation as follows:

$$\begin{aligned}&\mathbf{Exp}\left[\sum_{\substack{v \in V_{\tilde{k}}(S) \\ c_v \ge \beta\tilde{k}/p^2}} c_v(\log c_v + \log^2 \tilde{k} + \log(1/p))\right] \\ &= \sum_{v \in \mathrm{lev}_{\ge \beta\tilde{k}/p^2}(L)} \Pr[v \in V_{\tilde{k}}(S)]\, c_v(\log c_v + \log^2 \tilde{k} + \log(1/p))\end{aligned}$$

(grouping by level, using (A.1), and letting l_c denote the number of vertices in $\mathrm{lev}_c(L)$)

$$\le \sum_{c = \beta\tilde{k}/p^2}^{|L|} l_c\, p^2 e^{-cp/2} c(\log c + \log^2 \tilde{k} + \log(1/p))$$

(bounding the sum by an integral and using $l_c = O(|L|c^{1/3})$ [21])

$$\ll \int_{c = \beta\tilde{k}/p^2}^{\infty} |L| c^{1/3} p^2 e^{-cp/2}\, c(\log c + \log^2 \tilde{k} + \log(1/p))\, \mathrm{d}c$$

(substituting $c = t\tilde{k}/p$ and using $\mathrm{d}c = (\tilde{k}/p)\mathrm{d}t$)

$$= \int_{t = \beta/p}^{\infty} |L| (t\tilde{k}/p)^{1/3} p^2 e^{-t\tilde{k}/2}\, (t\tilde{k}/p)(\log(t/p) + \log^2 \tilde{k} + \log(1/p))(\tilde{k}/p)\, \mathrm{d}t$$

(collecting the terms and simplifying)

$$\ll |L|(\tilde{k}^3/p)\int_{t=\beta/p}^{\infty} t^2 e^{-t\tilde{k}/2}\,\mathrm{d}t$$

(solving the integral)

$$= |L|(\tilde{k}^3/p)(2\cdot(\beta/p)^2\tilde{k}^{-1} + 8(\beta/p)\tilde{k}^{-2} + 16\tilde{k}^{-3})e^{-\beta\tilde{k}/2p}$$

(simplifying)

$$\ll |L|e^{-\Theta(\beta)(\tilde{k}/p)},$$

for β large enough, as desired. □