Computational Geometry: Theory and Applications 109 (2023) 101945

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Subquadratic algorithms for some 3Sum-hard geometric problems in the algebraic decision-tree model ☆

Boris Aronov [a], Mark de Berg [b,*], Jean Cardinal [c], Esther Ezra [d], John Iacono [c,a], Micha Sharir [e]

[a] Tandon School of Engineering, New York University, Brooklyn NY, USA
[b] Eindhoven University of Technology, Eindhoven, Netherlands
[c] Université libre de Bruxelles (ULB), Brussels, Belgium
[d] School of Computer Science, Bar Ilan University, Ramat Gan, Israel
[e] School of Computer Science, Tel Aviv University, Tel Aviv, Israel

ARTICLE INFO

Article history:
Received 20 January 2022
Received in revised form 10 August 2022
Accepted 10 September 2022
Available online 15 September 2022

Keywords:
3Sum-hard problems
Algebraic decision-tree model
Point location
Polynomial partitions
Order type

ABSTRACT

We present subquadratic algorithms in the algebraic decision-tree model for several 3Sum-hard geometric problems, all of which can be reduced to the following question: Given two sets A, B, each consisting of n pairwise disjoint segments in the plane, and a set C of n triangles in the plane, we want to count, for each triangle $\Delta \in C$, the number of intersection points between the segments of A and those of B that lie in Δ. We present solutions in the algebraic decision-tree model whose cost is $O(n^{60/31+\varepsilon})$, for any $\varepsilon > 0$. Our approach is based on a primal-dual range searching mechanism, which exploits the multi-level polynomial partitioning machinery recently developed by Agarwal et al. (2021) [3]. A key step in the procedure is a variant of point location in arrangements, say of lines in the plane, which is based solely on the *order type* of the lines, a "handicap" that turns out to be beneficial for speeding up our algorithm.

1. Introduction

3Sum is a classical problem in computer science, which can be stated as follows (see, e.g., [23]): Given three sets X, Y, Z, each with n numbers, determine if there exists a triple $(x, y, z) \in X \times Y \times Z$ with $x + y + z = 0$. A quadratic-time algorithm is not difficult to obtain and it had been long believed that 3Sum cannot be solved in subquadratic time. This was disproven in [27] and later in [15], but a *substantially* subquadratic algorithm, that is, one running in time $O(n^c)$ for some $c < 2$, remains elusive and there are reasons to believe that such an algorithm may not exist, in the standard real-RAM

☆ Work by B.A. was partially supported by NSF grants CCF-15-40656 and CCF-20-08551, and by grant 2014/170 from the U.S-Israel Binational Science Foundation. Work by M.d.B. was partially supported by the Dutch Research Council (NWO) through Gravitation Grant NETWORKS (project no. 024.002.003). Work by J.C. was partially supported by the F.R.S.-FNRS (Fonds National de la Recherche Scientifique) under CDR Grant J.0146.18. Work by E.E. was partially supported by NSF CAREER under grant CCF:AF-1553354 and by grant 824/17 from the Israel Science Foundation. Work by J.I. was partially supported by Fonds de la Recherche Scientifique FNRS under grant no. MISU F 6001 1. Work by M.S. was partially supported by ISF grant 260/18, by grant 1367/2016 from the German-Israeli Science Foundation (GIF), and by Blavatnik Research Fund in Computer Science at Tel Aviv University.

* Corresponding author.
E-mail addresses: boris.aronov@nyu.edu (B. Aronov), m.t.d.berg@tue.nl (M. de Berg), jcardin@ulb.ac.be (J. Cardinal), ezraest@cs.biu.ac.il (E. Ezra), john@johniacono.com (J. Iacono), michas@tau.ac.il (M. Sharir).

https://doi.org/10.1016/j.comgeo.2022.101945

model (also referred to as the *uniform* model) of computation [35]. We say that a problem is 3SUM-*hard* if 3SUM can be reduced to it in substantially subquadratic (usually near linear) time.

Let A and B be two sets, each consisting of n pairwise disjoint line segments in the plane, and let C be a set of n triangles in the plane. We study the problem of counting, for each triangle $\Delta \in C$, the number of intersection points between the segments of A and those of B that lie inside Δ. We refer to this problem as *within-triangle intersection counting*. This is one of four 3SUM-hard problems (among many others) studied by Chan [15].[1] The other three problems are[2]:

(i) *Intersection of three polygons.* Given three simple n-gons A, B, C in the plane, determine whether $A \cap B \cap C$ is nonempty.
(ii) *Coverage by three polygons.* Given three simple n-gons A, B, C in the plane, determine whether $A \cup B \cup C$ covers a given triangle Δ_0.
(iii) *Segment concurrency.* Given sets A, B, C, each consisting of n pairwise disjoint segments in the plane,[3] determine whether $A \times B \times C$ contains a concurrent triple.

Chan [15] presents slightly subquadratic algorithms for all four problems, whose running time in the uniform model is $O((n^2/\log^2 n)\log^{O(1)}\log n)$. He observes that questions (i)–(iii) can be reduced in near-linear time to within-triangle intersection counting, so it suffices to present an efficient subquadratic solution for the latter problem.

We study the within-triangle intersection-counting problem in the algebraic decision-tree model. In this model only sign tests of polynomials of constant degree that access explicitly the endpoint coordinates of the input segments and vertices of the input triangles count towards the running time. All other operations cost nothing in the model, but are assumed not to access those real parameters specifying the input segments and triangles explicitly. Although originally introduced for establishing lower bounds [9], the algebraic decision-tree model has become a standard model for upper bounds too. By now it has been used in the study of many problems, including the 3SUM-problem itself [14,21,24,27,29] and various 3SUM-hard geometric problems [6,7,21]. One can interpret the decision-tree model as an attempt to isolate and minimize the cost of the part of the algorithm that explicitly accesses the real representation of the input objects, and ignore the cost of the other purely discrete steps. This can provide us with an insight about the problem complexity, which might eventually lead to an improved solution in the uniform model as well. See also the recent work by Chan and Zheng [16], and references therein, for problems where a subquadratic algorithm in the decision-tree model can be turned into a subquadratic algorithm in the uniform model.

We show that within-triangle intersection counting and, hence, also problems (i)–(iii), can be solved in this model with $O(n^{60/31+\varepsilon})$ sign tests, for any $\varepsilon > 0$. Chan [15] also remarks (without providing details) that his algorithm can be implemented in $O(n^{2-\delta})$ time in the algebraic decision-tree model, for some $\delta > 0$ that he left unspecified. (As was communicated to us, with some care one can obtain $\delta \approx 0.01$.) Our algorithm is rather different from Chan's, and gives the concrete, improved value for δ (any positive $\delta < 2/31$), as mentioned above. Our techniques appear to be of independent interest and to have the potential to apply to other problems, as we demonstrate in Section 4.

If the segments in A and B and the triangles in C were all full lines,[4] then determining the existence of a concurrent triple of lines in $A \times B \times C$ (the so-called *concurrency testing* problem) is the dual version of the classical 3SUM-hard *collinearity-testing* problem. In the latter problem we are given three sets of points in the plane and wish to determine whether their Cartesian product contains a collinear triple. Barba et al. [7] studied this question in a restricted version where each of the three point sets is assumed to lie on a constant-degree algebraic curve; they described subquadratic algorithms in the algebraic decision tree model for this case, as well as slightly subquadratic algorithms in the RAM model. Recently, Aronov et al. [6] studied this problem in the algebraic decision-tree model, where only two of the three sets lie on constant-degree algebraic curves; they obtained an algorithm performing roughly $O(n^{28/15})$ sign tests.

The problems studied here can be regarded as other dual versions of collinearity testing, where restrictions of a different kind are imposed. As noted by Chan [15], the additional disjointness properties that are assumed here make the problem simpler than collinearity testing (albeit by no means simple), and its solution appears to have no bearing on the unconstrained collinearity-testing problem itself. In Section 5 we comment on the substantial differences between this work and the work by Aronov et al. [6].

Our technique is based on hierarchical cuttings of the plane, as well as on tools and properties of segment-intersection range searching. We also use the so-called Fredman's trick in algebraic-geometric settings, in which the problem is solved using a primal-dual range searching mechanism involving points and surfaces in $\mathbb{R}^6$. This reduction exploits the very recent multi-level polynomial partitioning technique of Agarwal et al. [3] (see also a similar complementary technique of Matoušek and Patáková [31]). Our range-searching mechanism for points and algebraic surfaces in higher dimensions is a by-product of our analysis, which appears to be broadly applicable to other range-searching applications, and we regard it as a technique of independent interest; see, for example, Proposition 3.2 and its proof.

[1] Chan [15] refers to this problem as "triangle intersection-counting."
[2] The fact that these problems are 3SUM-hard, and the connections between them, are stated in [15].
[3] The segments of one set, say C, need not be pairwise disjoint. Although not explicitly stated, the technique in [15] for the uniform model can also handle this situation.
[4] Disjointness then of course cannot be assumed, unless the lines in each set are parallel, as in the dual version of the 3SUM-hard GEOMBASE problem [23].

Point location in arrangements. An additional key ingredient of our approach involves point location in an arrangement of lines in the plane (or an arrangement of curves, or of hyperplanes in higher dimensions). This is of course a well studied problem with several optimal solutions—see for example the survey by Snoeyink [34]—but we adapt and use techniques that are handicapped by the requirement that each operation that examines the real parameters specifying the lines involves at most *three* input lines. In contrast, the persistent data structure of [33], for example, needs to sort the vertices of the arrangement from left to right, thus requiring comparisons of the x-coordinates of a pair of vertices, which are in general determined by the parameters of *four* input lines. The persistent data-structure method has been used in [6,15] for the study of other 3SUM-hard geometric problems. Here we replace this method with one that uses solely the relative positions of triples of lines, the so-called *order type* of the arrangement. In this new approach each comparison involves only *three* input lines, which eventually leads to improved performance of the algorithm.

In standard settings, separating the order-type computation from the rest of the processing makes no sense, since one can compute the arrangement, which gives the full order-type information, in $O(N^2)$ time [10]. This makes the approach based on the order type noncompetitive, as one can just do point location in the line arrangement, in the uniform model, with $O(N^2)$ preprocessing. Nevertheless, in the applications considered in this paper (see Sections 3 and 4), the input lines have a special representation, which allows us to avoid an explicit construction of their order type and obtain this information implicitly in subquadratic time in the decision-tree model. The rest of the preprocessing, which still takes quadratic time and storage in the uniform model, costs nothing in the decision-tree model.

The problem of determining how the order type of an arrangement can be used in order to construct an efficient point-location data structure has, to the best of our knowledge, never been addressed explicitly. As we believe that this kind of "handicapped" point location will be useful for other applications (some of which are mentioned in Section 4), we present it in some detail in Section 2. We also present extensions of this technique to arrangements of constant-degree algebraic curves in $\mathbb{R}^2$, and to arrangements of planes or hyperplanes[5] in higher dimensions, which is used in the applications in Section 4.

Paper organization. In summary, the remainder of this paper is organized as follows: In Section 2 we construct two different point-location data structures based on the order-type information alone. The algorithm for solving the within-triangle intersection-counting problem in the algebraic decision-tree model, and, consequently, also the other three problems listed at the beginning of this section, is then presented in Section 3. Additional applications of our technique are described in Section 4; they include: (i) counting intersections between two sets of pairwise disjoint circular arcs inside disks, and (ii) minimum distance problems between lines and two sets of points in the plane. We conclude with a discussion in Section 5.

2. Order-type–based point location in arrangements

Order types. An arrangement of non-vertical lines in the plane (and, later, curves in the plane, or hyperplanes in higher dimension) can be described in the following combinatorial fashion. We use the notion of an *order type*, defined for a set L of lines as follows: Given any ordered triple of lines (ℓ_1, ℓ_2, ℓ_3) from L, where both ℓ_2 and ℓ_3 intersect ℓ_1, we record the left-to-right order of the intersections $\ell_1 \cap \ell_2$ and $\ell_1 \cap \ell_3$ along ℓ_1; note that the left-to-right order can also specify that the intersections coincide. The totality of this information gives, for each line in L, the left-to-right order of its intersections with every other line it meets. We also assume the existence of an "infinitely steep" line ℓ_∞, placed sufficiently far to the left, the order of whose intersections with the "normal" lines encodes the (reverse) order of their slopes. This information is dual to the perhaps more familiar notion of an order type for a set of points in the plane [25]. A higher-dimensional analog of this information involves recording the order in which a line that is the intersection of $d-1$ hyperplanes in $\mathbb{R}^d$ meets the remaining hyperplanes that meet but do not contain it. We also assume a suitable analog of the "infinitely steep line," replacing the line by a hyperplane, recursively defined over the dimension.

Back in the plane, the sorted sequences along each line of the intersection points with the other lines are called *local sequences* [26]. This view allows us to extend the definition of the order type to unbounded x-monotone curves, where each pair of curves is assumed to intersect in at most s points, for some constant s. To this end we label each intersection between two curves γ_i, γ_j, with $i < j$, by a triple (i, j, k), where the parameter k indicates that it is the kth leftmost intersection point of γ_i and γ_j. We allow the arrangement of curves to contain degeneracies, such as multiple curves passing through a common point, two or more curves being tangent to each other, or even both occurring simultaneously. Our encoding of the order type must accommodate such events. For example, the ordering of the intersections of γ_i with γ_j and with γ_ℓ must include the possibility that some pair of these intersection points coincide. The kth intersection point (i, j, k) of γ_i and γ_j is further labeled so as to distinguish a proper crossing from a tangency. The order type for a collection of curves then records, for each curve in the collection, the left-to-right order of the labeled intersection points. The order type also includes the vertical order of the curves at $x = -\infty$ (the order in which they intersect a suitably defined infinitely steep line at $x = -\infty$).

[5] For compactness of presentation, we do not single out the case of lines in the plane, and obtain it as a special case of hyperplanes in $d = 2$ dimensions.

The significance of the order type is that (a) it only records information for $(d+1)$-tuples of objects, and (b) it contains enough information that lets us construct the arrangement and preprocess it for fast point location, without having to further access the actual parameters that define the objects. See below for details concerning (b).

The problem we tackle now is the following: Given the order type of an arrangement, preprocess this information into a point-location data structure. The preprocessing stage is not allowed to access the actual geometric description of the objects, such as the coefficients of the equations defining the lines, curves, or hyperplanes, but can only exploit the discrete data given by the order type. A query, in contrast, is allowed to examine the coefficients of the objects it encounters.

We present two solutions for this problem. First, we show that, for d-dimensional hyperplane arrangements, for any $d \geq 2$, the sampling method of Meiser [32] (see also [20]) can be implemented using only order-type information. Second, we show that for arrangements of unbounded x-monotone curves in the plane, a simple variant of the separating-chain method for point location [19,30] can be implemented such that only order-type information is used during the preprocessing.

2.1. Sampling-based approach for hyperplane arrangements

Let H be a set of N non-vertical hyperplanes in $\mathbb{R}^d$, where $d \geq 2$ is a fixed constant. We want to construct a point-location data structure for the arrangement $\mathcal{A}(H)$ induced by H, where we are only given the order type of H. Essentially, we are given, for each intersection line formed by $d-1$ hyperplanes, the order of its intersections with the other hyperplanes not containing it. (Alternatively, we are given, for each simplex σ formed by $d+1$ of the hyperplanes, the vertices of σ sorted by their x_1-coordinate.) We only require H not to contain vertical hyperplanes. In particular, we allow for two hyperplanes to be parallel, for d hyperplanes not to intersect at all or intersect in a line or a higher-dimensional flat, and for more than d hyperplanes to share a point.

We briefly sketch the randomized method first proposed by Meiser [32] and analyzed in detail by Ezra et al. [20] (see also [14]), and show that the order type information is sufficient to construct the data structure.

Before considering the point-location structure, we note that the order type suffices to construct a discrete representation of the arrangement $\mathcal{A}(H)$. In this representation each j-dimensional cell of $\mathcal{A}(H)$, for $j = 1, \ldots, d$, stores the set of all $(j-1)$-dimensional cells that form its relative boundary (and consequently of all cells, of all dimensions, on its boundary), with back pointers from each cell to all higher-dimensional cells that contain it in their relative boundary. This can be done, e.g., by the Folkman–Lawrence topological representation theorem for oriented matroids [22], which, roughly speaking, implies that, given the order type of H, one can construct a combinatorial representation for the arrangement $\mathcal{A}(H)$, consisting of all *sign conditions*. That is, each face f of $\mathcal{A}(H)$ (of any dimension) is encoded by a sign vector $\{-1, 0, +1\}^{|H|}$ representing the *above* (corresponding to $+1$), *below* (-1), or *on* (0) relation of f with respect to each hyperplane in H; see [13] for an inductive proof that such an encoding can be obtained using only order-type information for the planar case, and [12] for its generalization to higher dimensions. Given this property, a naïve actual construction of the combinatorial representation of $\mathcal{A}(H)$ is easy to derive, and is free of charge in the decision-tree model, once the order type of H is computed. When we perform a point-location query we report a pointer to the sign vector of the cell of $\mathcal{A}(H)$ that contains the query point—see below.

Preprocessing. Given the arrangement $\mathcal{A}(H)$ and a fixed $\varepsilon > 0$, we first construct a random sample S of $O(\frac{d^2}{\varepsilon} \log \frac{d}{\varepsilon})$ hyperplanes of H. We then compute a *canonical* triangulation of the arrangement $\mathcal{A}(S)$. To this end, for each face of $\mathcal{A}(S)$ of dimension at least two, we fix a *reference vertex* p of this face, defined as its lexicographically smallest vertex, where each vertex is represented by the lexicographically smallest d-tuple of (the indices of) the hyperplanes that contain it and whose intersection is a single point. Triangulating a face f of $\mathcal{A}(S)$ is done by the *fan* obtained by adding the vertex p to each simplex in the recursively constructed triangulations of the lower-dimensional faces composing the boundary of f and not incident to p. Next, we construct the *conflict list* $L(\Delta)$ for each simplex Δ of the triangulation, of any dimension, defined as the set of hyperplanes of H that *cross* Δ. Here we say that $h \in H$ *crosses* Δ if h intersects Δ but does not fully contain it. $L(\Delta)$ can indeed be constructed using only the order type: Deciding whether a hyperplane $h \in H$ belongs to $L(\Delta)$ amounts to testing whether there exist two vertices u, v of Δ that lie on different sides of h. This test can be implemented using $k+1$ orientation tests, where k is the dimension of Δ, one for each vertex[6] for the $(d+1)$-tuple consisting of h and the d planes defining u. Two vertices u and v lie on different sides of h iff these are opposite orientations.

From standard results on ε-nets [28], a suitable choice of the constant of proportionality in the bound on the size of the sample S guarantees that, with high probability, the conflict list sizes are not larger than εn, for all simplices Δ. We continue resampling until this condition is met (note that this requires $O(1)$ trials in expectation).

It remains to recurse, for each simplex Δ of the triangulation, on the hyperplanes in $L(\Delta)$. If Δ is not full-dimensional, any query point directed to this recursive structure will lie in the affine span of Δ, but we still build a full-dimensional structure for $L(\Delta)$; see [20] for a similar approach to handle cells Δ of any dimension. This leads to a hierarchical data structure in which the number of hyperplanes decreases by a factor of ε at each level. The construction continues until the

[6] Note that, to ensure consistent orientation tests, to represent a vertex v in the arrangement, we pick lexicographically smallest tuple $\langle h_1, h_2, \ldots, h_d \rangle$ of hyperplanes whose intersection is v and *then* we choose the order of the h_i's so that a hyperplane h lies below (above) u iff the sign of $\det[h_1, \ldots, h_d, h]$ is positive (resp., negative).

number of hyperplanes falls below a suitable constant, at which point we simply store the remaining hyperplanes at Δ. Let w be a leaf in this hierarchy. It will be convenient to further preprocess the set $H(w)$ of hyperplanes stored at w into a tree $\mathcal{T}_w$ that allows us to locate a query point in the arrangement $\mathcal{A}(H(w))$. The structure $\mathcal{T}_w$ is simply a ternary tree of depth $|H(w)| = O(1)$, where a node at level j stores the jth hyperplane h_j of $H(w)$, so we can test if a query point is below, on, or above h_j. Observe that each leaf of $\mathcal{T}_w$ corresponds to a unique cell in the arrangement $\mathcal{A}(H(w))$ and, hence, also in $\mathcal{A}(H)$—indeed, the sign with respect to every hyperplane in $H \setminus H(w)$ is determined by the search path to the node w in the hierarchy, and is therefore fixed.

Answering queries. Each point-location query returns the relatively open simplex,[7] of the suitable dimension, in the canonical triangulation of $\mathcal{A}(H)$ that contains the query point q. Queries are answered as follows. First, we locate the (open) simplex Δ of the canonical triangulation of $\mathcal{A}(S)$ containing the query point q. Since d is assumed to be constant, S is also of constant size, and so locating Δ can be done in $O(1)$ time (e.g., by inspecting every simplex of the triangulation). Next, we recurse in the data structure attached to Δ. When we reach a leaf w of the hierarchy, we continue to search in the tree $\mathcal{T}_w$. When we reach a leaf in $\mathcal{T}_w$, we have located q and can report (a pointer to) the sign vector of the cell containing q.

The overall number of these recursive steps is $O(\log n)$, and thus answering a query costs $O(\log n)$ arithmetic operations, where the hidden constant[8] is polynomial in d. As noted, in our applications we only need to determine whether q lies on a hyperplane of H.

The following lemma summarizes the result.

Lemma 2.1. *Let H be a set of n hyperplanes in $\mathbb{R}^d$, where $d \geq 2$ is a constant. Using only the order type of H, we can construct a polynomial-size data structure that guarantees $O(\log n)$-time point-location queries in the arrangement $\mathcal{A}(H)$; the implied constant depends polynomially on d. The preprocessing time and storage of the data structure cost nothing in the decision-tree model.*

2.2. Level-based approach for order-type–based point location in x-monotone curves in the plane

Let $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ be a collection of n unbounded x-monotone constant-degree algebraic curves in the plane, and let $\mathcal{A}(\Gamma)$ denote the arrangement induced by Γ. Let $s = O(1)$ denote the maximum number of intersections between any pair of curves of Γ. We assume that Γ does not contain vertical lines. Note that we do allow multiple curves to pass through the same point. Recall that the order type of Γ gives us the following information:

- For each curve $\gamma \in \Gamma$, the left-to-right order of the intersection points of γ with the other curves of Γ; in case of tangencies between curves, the intersection point is labeled as such.
- The vertical order of the curves at $x = -\infty$.

As already mentioned, each intersection point p is labeled by a triple[9] (i, j, k) of indices, where γ_i and γ_j are the pair of curves that intersect at p, and $1 \leq k \leq s$ is the *index* of p, meaning that p is the kth leftmost intersection point of the two curves. Note that the order type tells us whether the kth leftmost intersection point of γ_i and γ_j lies to the left or to the right of, or coincides with the k'th leftmost intersection point of γ_i and $\gamma_{j'}$, for any quintuple of indices i, j, k, j', k'. (Observe that the quintuple involves only three curves.) Here too we are not concerned with the actual construction of the order type—we simply assume it is given to us in advance. Such a construction, in a special context that arises in our applications, is considered when we discuss these applications, in Sections 3 and 4.

Recall that in the query phase we *do* have access to an explicit description of the curves. We assume a model of computation in which the following operations can be performed in $O(1)$ time by the query algorithm:

- Given a query point q and a curve γ_j, decide whether q lies above, on, or below γ_j.
- Given a query point q and an intersection point v that is labeled (i, j, k), decide whether the x-coordinate of q is smaller than, equal to, or larger than, the x-coordinate of v.

Executing these basic operations is rather easy for lines (where we always have $k = 1$). When Γ contains higher-degree curves, however, executing the second operation is more involved. More concretely, comparing q to an intersection point v, labeled as (i, j, k), amounts to testing whether a certain quantified Boolean predicate P is satisfied. This predicate P depends on the real parameters specifying γ_i and γ_j and on the coordinates of q. It involves $O(k)$ quantified variables that

[7] For our applications, as well as for the techniques in [14,20] on which we rely, the information that we actually want is whether the query point q lies on *any* of the hyperplanes in H—this is provided by the point location.

[8] The value of this constant depends on the storage allocated to the structure. For example, spending $n^{2d \log d + O(d)}$ on storage guarantees query cost of $O(d^4 \log n)$ [20].

[9] Actually, a 4-tuple, to distinguish proper crossings from tangencies. To simplify the notation we will ignore the fourth component and pretend we work with triples.

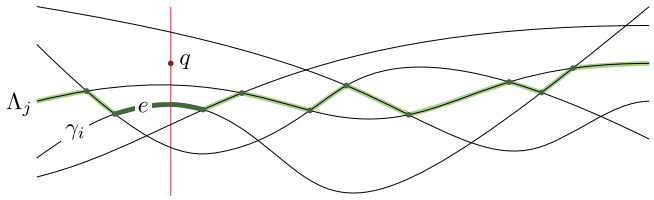

Fig. 1. A binary search with the x-coordinate of the query point q in the set of x-coordinates of the vertices of level Λ_j (in green) gives us the edge e (in dark green) of Λ_2 intersecting the vertical line through q. Comparing q to the curve γ_i containing e then tells us whether q is above, below, or on Λ_j. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

represent the k intersection points of γ_i and γ_j to the left of, and including, v, and consists of polynomial equalities and inequalities, whose number depends on k, of constant degree (which depends on the degree of the curves of Γ). Still, since k and the degrees of the curves are constant, the predicate P has constant complexity. Hence, its validity can be tested in $O(1)$ time [8].

Preprocessing. Our point-location data structure is based on the separating-chain method for planar maps, due to Lee and Preparata [30], which was later refined by Edelsbrunner et al. [19]. For the case of unbounded x-monotone curves, the separating-chain method is especially easy to implement, since we can simply use the levels in the arrangement as separating chains. This allows us to carry out the preprocessing using only order-type information, as explained next.

Observe that a doubly-connected edge list (DCEL) representation [10] of the arrangement $\mathcal{A}(\Gamma)$ can be constructed in $O(n^2)$ time in the real-RAM model, and at no cost in our model, using only the given order-type information, without further accessing the real parametric representation of the curves. Specifically, the order type gives us the local sequences of intersection points along each curve, and, assuming for the moment that there are no concurrent triples, we can identify, for each intersection point, its four incident edges. Using this data we can trace the boundary of each 2-face of the arrangement, and consequently obtain the DCEL structure (observing that each face is x-monotone in this setup). If more than two arcs meet at a vertex v, we also need to know the circular order of the incident curves around v, which we can deduce from the order of the curves at $x = -\infty$ and from the indices of v along each curve, as we have assumed that all the crossings are proper (by our general position assumption described earlier).

Let Λ_j, for $j = 0, \ldots, n-1$, denote the jth level in the arrangement $\mathcal{A}(\Gamma)$; refer to Fig. 1. In other words, Λ_j is the closure of the set of points that lie on the curves of Γ and have exactly j curves passing below them. Note that the levels can easily be extracted from the DCEL of $\mathcal{A}(\Gamma)$, as the rule for constructing a level is to follow it from left to right, switching at each vertex to the other curve forming that vertex, when the intersection is a proper crossing. (This latter rule has to be modified, in an easy manner, when more than two curves are incident to v or when a tangency is involved.) We store each level Λ_j as a sorted sequence of its vertices and edges, in left-to-right order, where each vertex is represented as a triple (i', j', k'), as explained above—when more curves are incident to the vertex, any one of the representing triples suffices—and each edge is represented by the index of its defining curve.

Answering queries. To answer a query, we perform a binary search on the levels $\Lambda_0, \ldots, \Lambda_{n-1}$. At each step of this primary binary search we need to decide whether the query point q lies above, on, or below a level Λ_j. We can do this by a secondary binary search, this time on the x-coordinates of the vertices of Λ_j. This gives us an edge e of Λ_j intersecting the vertical line through q. By comparing q to the curve $\gamma_i \in \Gamma$ containing e, we can determine the position of q relative to Λ_j. If q lies on Λ_j then we are done, otherwise we continue the primary binary search. When the query algorithm has finished, we have either identified an edge (or vertex) of $\mathcal{A}(\Gamma)$ containing q, or an edge immediately above (or below) q. Since the DCEL gives us, for each edge e of $\mathcal{A}(\Gamma)$, the two adjacent faces of $\mathcal{A}(\Gamma)$, we can now answer the query, returning the (DCEL pointer to the) face, edge or vertex containing q.

The cost of the search is $O(\log^2 n)$, where the constant of proportionality depends on the degree of the curves of Γ.

Note that, unlike the technique in the preceding subsection, here we simply report the (pointer to the) face containing q, as provided by the DCEL, rather than a pointer to the sign vector.

Lemma 2.2. *Let Γ be a set of n unbounded x-monotone constant-degree algebraic curves in the plane in general position. Using only the order-type information of Γ, we can construct a data structure that uses $O(n^2)$ storage and that allows us to answer point-location queries in the arrangement $\mathcal{A}(\Gamma)$ in $O(\log^2 n)$ time.*

Note that the $O(\log n)$ query time of the procedure described in Section 2.1 for $d = 2$ is faster than the $O(\log^2 n)$ time of the procedure presented here for curves in the plane. However, the preceding sampling-based method does not extend to non-straight curves, since there is no obvious way to extend the notion of a canonical triangulation to the case of curves. The only viable way of doing this seems to use the standard vertical-decomposition technique. Unfortunately (for us), constructing the vertical decomposition requires that we compare the x-coordinates of vertices defined by different, unrelated pairs of curves. Such a comparison involves *four* input curves and it cannot be resolved from the order-type

information alone. For lines in the plane, however, the technique from Section 2.1 does yield the improved logarithmic query time.

Remarks. (1) We have assumed that the given curves are algebraic of constant degree, but the same machinery applies if the curves are arbitrary, provided we have constant-time black-box routines that perform the two basic operations: Testing whether a point lies above, below, or on a curve, and testing whether a point lies to the left or to the right of a specific intersection point of two curves.
(2) It is tempting to apply fractional cascading [18] to reduce the query time to $O(\log n)$. This is problematic in our context, however, because to implement fractional cascading, we must be able to merge suitable sorted subsequences of the sequences of vertices of different levels in the arrangement. Such a merge requires comparing the x-coordinates of two vertices on different levels, which is not possible using order type only (see the discussion above).

3. The algorithm for within-triangle intersection counting

Our input consists of two sets A, B, each of n pairwise disjoint segments in the plane, and of a set C of n triangles in the plane, with no segment of A overlapping a segment of B. It will be convenient to make the following no-degeneracy assumptions: among the segments of A, B, and edges of triangles of C, no two share a supporting line, and no endpoint of one segment lies on another. Before we present our algorithm, we explain how to eliminate such degeneracies using a suitable preprocessing step.

First, all endpoints of a segment of A lying on a segment of B (or vice versa) are computed, which can be done in $O(n \log n)$ time in the uniform model, since the segments in A (and, similarly, those in B) are pairwise disjoint. The segments with an endpoint on another segment are then slightly extended to eliminate the degeneracy. Note that this does not change the number of within-triangle intersections. We also compute which endpoints of segments in $A \cup B$ lie on an edge of C. Since these edges are not pairwise disjoint, this is slightly more difficult, but using standard range-searching techniques we can still do it in $O(n^{4/3+\varepsilon})$ time in the uniform model. Again, we extend the relevant segments from $A \cup B$ to eliminate the degeneracy.

Next we identify the vertices of a triangle $\Delta \in C$ that lie on a segment from $A \cup B$. This can again be done in $O(n \log n)$ times in the uniform model. We then slightly perturb Δ to eliminate the degeneracy. We do this by slightly moving the relevant triangle vertex in such a way that the perturbed triangle contains the original triangle Δ. Thus, no within-triangle intersections are lost.

It remains to get rid of situations where two segments or edges share a supporting line. Such degeneracies can easily be detected in $O(n \log n)$ time in the uniform model. We first eliminate the degeneracies involving a triangle edge, by slightly expanding each triangle $\Delta \in C$ so that we do not loose any within-triangle intersections. Finally we eliminate the collinearity of segments in $A \cup B$, by slightly shifting these segments. Note that we do not loose any within-triangle intersections here, since we assumed that the segments in $A \cup B$ do not overlap.[10]

From now on we assume that degeneracies mentioned above do not occur. Note that we do allow a triple of segments (one from A, one from B, and an edge of a triangle from C) to be concurrent; this is what happens in the special case of concurrency testing.

A roadmap of the algorithm. As our approach is fairly involved, we start with a sketch of our procedure. The outline focuses on the simpler *segment concurrency* problem, where C is a set of (not necessarily disjoint) segments, rather than proper triangles, and the goal is to determine whether there is a triple $(a, b, c) \in A \times B \times C$ of concurrent segments. To further simply the description, we assume that the segments in C are actually full lines.

We fix a parameter $g \ll n$ and put $r := n/g$ (g will be the subproblem size when we apply the so-called "Fredman's trick" below). We construct a $(1/r)$-cutting $\Xi(A)$ for the segments of A, and another such cutting $\Xi(B)$ for the segments of B. Since the segments of A are pairwise disjoint, we can construct $\Xi(A)$ so that it has size $O(r)$, and similarly for $\Xi(B)$ (see [11] and Fig. 2). We overlay the two cuttings and obtain a planar decomposition Ξ. While the complexity of Ξ is $O(r^2)$, any line of C crosses only $O(r)$ of its cells, since it crosses only $O(r)$ cell boundaries in each cutting.

Consider a two-dimensional cell σ of Ξ (lower-dimensional cells are easier to handle). Define $A_\sigma \subseteq A$ and $B_\sigma \subseteq B$ to be the sets of those segments that cross σ. Note that $|A_\sigma|, |B_\sigma| \leq n/r = g$. We will preprocess $A_\sigma \cup B_\sigma$ into a data structure that supports efficient queries, each specifying a line c and asking whether c passes through an intersection point of a segment of A_σ and a segment of B_σ. We pass to the dual plane, obtain sets A^*_σ and B^*_σ of at most g points (dual to the lines containing the segments) each. (Observe that "short" segments, which have an endpoint inside σ, require special treatment; see below.) The query is a point c^* and the task is to determine whether c^* is collinear with a pair of points $(a^*, b^*) \in A^*_\sigma \times B^*_\sigma$. For $a \in A_\sigma$ and $b \in B_\sigma$ we define $\gamma_{a,b}$ to be the line that passes through a^* and b^*, and let Γ_σ denote the collection of these lines. The query with c^* then reduces to point location in the arrangement $\mathcal{A}(\Gamma_\sigma)$, where we only need to know whether c^* lies on any of the lines.

[10] With a suitable modification of what it means to "count intersections between segments of A and B within each triangle of C," our preprocessing can be modified to deal with such overlaps.

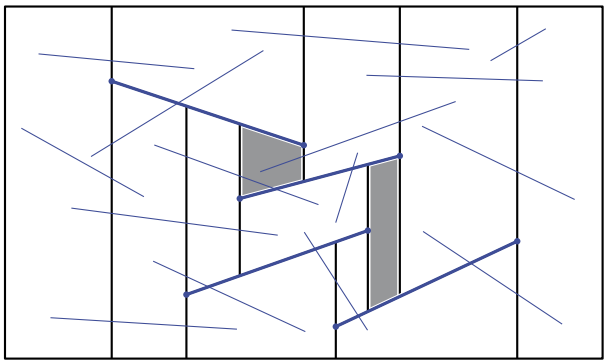

Fig. 2. A cutting of a set of segments (shown in blue). Each (open) trapezoid in the cutting is intersected by at most n/r segments.

We cannot perform this task explicitly in an efficient manner in the uniform model, since the complexity of $\mathcal{A}(\Gamma_\sigma)$ is $O(g^4)$ and we have $O(r^2) = O(n^2/g^2)$ such arrangements, of overall size $O(n^2 g^2)$. We can do it, though, in the algebraic decision-tree model, in an implicit manner, using the so-called *Fredman's trick*; see [27] for a simpler yet representative application of Fredman's trick, as well as [6,7] for geometric applications of the trick. Concretely, we apply the order-type–based machinery of Section 2 to construct $\mathcal{A}(\Gamma_\sigma)$ and preprocess it for fast point location. More precisely, we first construct the order type of Γ_σ: this involves, for each triple of lines γ_{a_1,b_1}, γ_{a_2,b_2}, γ_{a_3,b_3}, determining the ordering of their intersection points along each of these lines. We express this test, in a straightforward manner, as the sign test of some 12-variate constant-degree polynomial $G(a_1, a_2, a_3; b_1, b_2, b_3)$.

We map the triple (b_1, b_2, b_3) to a point in a six-dimensional parametric space, and (a_1, a_2, a_3) to an algebraic surface ψ_{a_1,a_2,a_3} in this space, which is the locus of all triples (b_1, b_2, b_3) with $G(a_1, a_2, a_3; b_1, b_2, b_3) = 0$. We now need to locate the points (b_1, b_2, b_3) in the arrangement of the surfaces ψ_{a_1,a_2,a_3}, from which all the sign tests can be resolved, at no extra cost in the algebraic decision-tree model, thereby yielding the desired order type. The subsequent construction of the arrangement $\mathcal{A}(\Gamma_\sigma)$, and its preprocessing for fast point location, using the machinery in Section 2, also cost nothing in our model.

To make this process efficient, we group together all the points (b_1, b_2, b_3), for b_1, b_2, b_3 in the same cell σ, over all cells of $\Xi(B)$, into one global set P, and group the surfaces ψ_{a_1,a_2,a_3}, for a_1, a_2, a_3 in the same cell of $\Xi(A)$, into another global set Ψ. We have $|P|$, $|\Psi| = O(r) \cdot O(g^3) = O(ng^2)$, since there are only $O(r)$ cells of $\Xi(A)$ (resp., of $\Xi(B)$) from which the triples (a_1, a_2, a_3) (resp., (b_1, b_2, b_3)) are drawn.

Using the recent machinery of Agarwal et al. [3], or the alternative technique of Matoušek and Patáková [31], we can perform this batched point location in 6-space in time

$$O\left(|P|^{6/7+\varepsilon}|\Psi|^{6/7+\varepsilon} + |P|^{1+\varepsilon} + |\Psi|^{1+\varepsilon}\right) = O\left((ng^2)^{12/7+2\varepsilon}\right),$$

for any $\varepsilon > 0$. Full details of this step are given in Section 3.1.

Searching with the dual points c^* takes $O\left(\frac{n^2}{g}\log g\right)$ time, because we have n query lines c, each line crosses $O(r) = O(n/g)$ cells σ, and each point location with c^* in each of the encountered arrangements takes $O(\log g)$ time, by Lemma 2.1. Balancing (roughly) this cost with the preprocessing cost, we choose $g = n^{2/31}$, and obtain the total subquadratic running time $O(n^{2-2/31+\varepsilon}) = O(n^{60/31+\varepsilon})$.

Quite a few issues were glossed over in this overview. Since the segments of A and of B are bounded, a cell σ may contain endpoints of these segments, making the passage to the dual plane more involved. The same applies in the original within-triangle intersection-counting problem, where the triangles of C may have vertices or more than one bounding edge that lie in or meet σ. We thus need to handle the presence of such 'short' segments and/or 'short' triangles. Moreover, we need to count intersection points within each triangle, and the number of cells in the overlay of the cuttings Ξ_A, Ξ_B that a triangle can fully contain is much larger than $O(r)$. All these issues require considerably more careful handling, detailed below. The overall runtime of the resulting algorithm remains $O(n^{60/31+\varepsilon})$, for any $\varepsilon > 0$.

Hierarchical cuttings. This ingredient is needed for counting intersection points in cells that are fully contained inside a query triangle. The application of hierarchical cuttings to our problem significantly reduces the query time—see below. Fix a parameter $g \ll n$ and put $r := n/g$. We construct a *hierarchical* $(1/r)$*-cutting* $\Xi(A)$ for the segments of A, which is a hierarchy of $(1/r_0)$-cuttings, where r_0 is some sufficiently large constant. The top-level cutting $\Xi_1(A)$ is constructed for A. Since the segments of A are pairwise disjoint, we can construct $\Xi_1(A)$ so that it consists of only $O(r_0)$ trapezoids—for concreteness, we write this bound as tr_0, for some absolute constant t—each of which is crossed by at most n/r_0 segments of A, which comprise the so-called *conflict list* of the cell σ, denoted as A_σ. The construction time of $\Xi_1(A)$, in the real-RAM model, is $O(n \log r_0) = O(n)$. See [11, Theorem 1] for details.

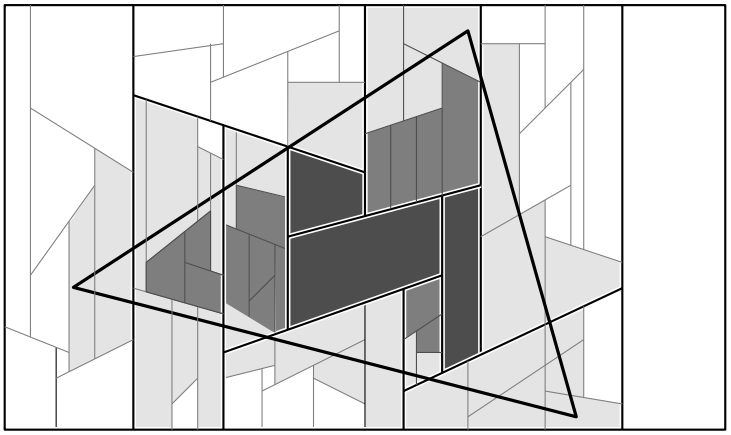

Fig. 3. A hierarchical cutting and its interaction with a triangle. The dark gray cells are the ones inside the triangle at the top level of the hierarchy; the medium gray cells are the ones inside the triangle at the second level (and whose parent cells are not inside the triangle). The light gray cells will be refined and handled at lower levels, since they intersect the triangle boundary.

For each cell σ of $\Xi_1(A)$, we clip the segments in its conflict list A_σ to within σ and apply the cutting-construction step recursively to this set, clipping also the cells of the new cutting to within σ and ignoring cells, or portions thereof, that lie outside σ (see below for a comment regarding the complexity of the clipped cells). We denote the union of all the resulting $(1/r_0)$-cuttings by $\Xi_2(A)$. We continue recursively in this manner, until we reach a level s at which every cell is crossed by at most n/r segments. We thus obtain a hierarchy of cuttings $\Xi_1(A), \Xi_2(A), \ldots, \Xi_s(A)$, for some index $s = O(\log r)$. We denote the collective hierarchy as $\boldsymbol{\Xi}(A)$. Since we stop the recursion as soon as $n/r_0^s \le n/r$, the overall number of cells of all the levels is $O((tr_0)^s) = O(r^{1+\varepsilon})$, for any prespecified $\varepsilon > 0$, for a suitable choice of $r_0 = r_0(\varepsilon)$. Technically, the trapezoids in the cutting are relatively open, and the cutting also includes one- and zero-dimensional cells; as the latter are easier to deal with, we will focus below on the two-dimensional cells of the cutting. At any level j of the hierarchy, the cells of $\Xi_j(A)$ are pairwise disjoint. As these cells partition the plane, each intersection point between a segment of A and a segment of B lies in precisely one cell of a suitable dimension of each level. See Fig. 3 for an illustration.

We apply a similar hierarchical construction to B, and let $\boldsymbol{\Xi}(B) = \{\Xi_j(B)\}_{j \le s}$ denote the resulting hierarchical cutting, which has analogous properties. (We assume for simplicity that the highest index s is the same in both hierarchies.)

We now overlay $\boldsymbol{\Xi}(A)$ with $\boldsymbol{\Xi}(B)$. More precisely, at each level j of the hierarchy, we overlay the cells of $\Xi_j(A)$ with the cells of $\Xi_j(B)$. We denote the jth level overlay as Ξ_j, and the entire hierarchical overlay structure as $\boldsymbol{\Xi} = \{\Xi_j\}_{j \le s}$. Since each of $\Xi_j(A)$ and $\Xi_j(B)$ consists of at most $(tr_0)^j$ cells, the number of cells of Ξ_j is at most $O((tr_0)^{2j})$. Since we have $r_0^s \approx r$ (up to a factor of r_0), it follows that the overall complexity of all the overlays is $O(r^{2+2\varepsilon})$, provided that we choose r_0, as above, to be sufficiently large, as a function of ε.

As already mentioned, for simplicity of exposition, we ignore lower-dimensional faces of the cuttings, and regard each of the overlays Ξ_j as a decomposition of the plane into pairwise openly disjoint convex polygons, each of complexity linear in $j \le s = O(\log r)$ (indeed, due to the clipping, each convex polygon is the intersection of at most j trapezoids). Each cell σ of the overlay is identified by the pair (τ, τ'), where τ and τ' are the respective cells of $\Xi_j(A)$ and $\Xi_j(B)$ whose intersection is σ; we simply label $\sigma = \tau \cap \tau'$ as (τ, τ'). Each bottom-level cell σ of the final overlay Ξ_s is crossed by at most $n/r = g$ segments of A and by at most $n/r = g$ segments of B.

Classifying the segments and triangles. Let $\sigma = (\tau, \tau')$ be a cell of Ξ_j, for any level j of the hierarchy. Call a segment e of A *long* (resp., *short*) *within* σ if e crosses σ and neither of its endpoints lies in σ (resp., at least one endpoint lies in σ). Let A_σ^l (resp., A_σ^s) denote the set of long (resp., short) segments of A within σ. Apply analogous definitions and notations to the segments of B. Denote by C_σ (resp., $C_\sigma^{(0)}$) the set of triangles with at least one edge that crosses σ (resp., that fully contain σ). Call a triangle $\Delta \in C_\sigma$ *long* (resp., *short*) in σ if σ does not (resp., does) contain a vertex of Δ, and denote by C_σ^l (resp., C_σ^s) the set of long (resp., short) triangles in C_σ.

For each triangle $\Delta \in C$, each of its edges crosses only $O((tr_0)^j)$ cells of Ξ_j. Indeed, as such an edge crosses from one cell of Ξ_j to an adjacent cell, it does so by crossing the boundary of either a cell of $\Xi_j(A)$ or a cell of $\Xi_j(B)$, and the total number of such crossings is $O((tr_0)^j)$. In particular, the edge crosses at most $O(r^{1+\varepsilon})$ cells of the final overlay Ξ_s. It follows that $\sum_{\sigma \in \boldsymbol{\Xi}} |C_\sigma^l| \le \sum_{\sigma \in \boldsymbol{\Xi}} |C_\sigma| = O(nr^{1+\varepsilon})$, but clearly $\sum_{\sigma \in \boldsymbol{\Xi}} |C_\sigma^s|$ is only $O(n \log r)$. In contrast, Δ can fully contain many more cells of Ξ_s, perhaps almost all of them, but the hierarchical nature of the construction allows us to deal with a much smaller number of such interior cells, by collecting them at higher levels of the hierarchy, as illustrated in Fig. 3; see below for details.

The algorithm: a quick review. The high-level structure of the algorithm is as follows. (This expands, and puts in more concrete form, the 'roadmap' overview given earlier.) We construct the hierarchies $\boldsymbol{\Xi}(A) = \{\Xi_j(A)\}_{j \ge 1}$ and $\boldsymbol{\Xi}(B) = \{\Xi_j(B)\}_{j \ge 1}$.

For each cell τ of $\Xi_j(A)$, we compute its conflict list A_τ, which, as we recall, is the set of all segments of A that cross τ. Similarly, we compute the conflict list $B_{\tau'}$ for each cell τ' of $\Xi_j(B)$, which contains all segments of B that cross τ'. We then form the hierarchical overlay $\boldsymbol{\Xi} = \{\Xi_j\}_{j\geq 1}$, and for each cell $\sigma = (\tau, \tau')$ of any overlay Ξ_j, we compute the subset A_σ of the segments of A_τ that cross σ, and the subset B_σ of the segments of $B_{\tau'}$ that cross σ. We partition A_σ into the subsets A^l_σ and A^s_σ of long and short segments (within σ), respectively, and apply an analogous partition to B_σ. The additional overall cost for constructing these sets, over all hierarchical levels, is $O(r^{2+\varepsilon} \cdot n/r) = O(nr^{1+\varepsilon}) = O(n^{2+\varepsilon}/g)$. (The cost at the bottom level dominates the entire cost over all levels.)

We then trace each triangle $c \in C$ through the cells of each overlay $\Xi_j \in \boldsymbol{\Xi}$ that are crossed by its edges, and form, for each cell σ of the overlay, the list C_σ of triangles of C with at least one edge that crosses σ. Cell-triangle interactions in which the triangle fully contains the cell will be handled using the hierarchical structure of the partition—see Fig. 3 and below. We partition C_σ into the subsets C^l_σ and C^s_σ, as defined earlier. As we show below, we can handle, in a much simpler way, the short triangles of C^s_σ, as well as the triangles of C^l_σ all three of whose edges cross σ, simply because the overall number of such triangle-cell interactions is small. We therefore focus on the triangles of C^l_σ that have only one or two edges crossing σ. For triangles with two crossing edges we use a standard two-level data structure, where at each level we consider only one crossing edge. This lets us assume, without loss of generality, that each triangle in C^l_σ is a halfplane. Each of these halfplanes can be represented by its bounding line, that is the line supporting the appropriate crossing edge of the triangle. We flesh out the details below.

We also assume, for now, that all the segments of A_σ and of B_σ are long in σ. This is the hard part of the analysis, requiring the involved machinery presented below. After handling this case, we will address the much simpler situations that involve short segments and/or short triangles (or triangles with three edges crossing σ, as well as triangles that fully contain cells). The cost of handling short segments or short triangles within cells is smaller, even in the uniform model, since the overall number of short objects within cells is smaller.

Handling the long segments. We preprocess each level j of the overlay, to compute, for each of its cells $\sigma = (\tau, \tau')$, the number of intersection points between the (long) segments of A^l_σ and those of B^l_σ (which, due to the clipping, lie in σ). This is a standard procedure that involves computing the number of pairs of segments from $A^l_\sigma \times B^l_\sigma$ whose intersection points with the boundary of σ interleave (these are precisely the pairs of intersecting segments), and can be implemented to take $O((|A^l_\sigma| + |B^l_\sigma|)\log^2(|A^l_\sigma| + |B^l_\sigma|))$ time [17].[11] We store the resulting count at σ.

Consider a two-dimensional cell σ, a segment $a \in A^l_\sigma$, a segment $b \in B^l_\sigma$, and a triangle $\Delta \in C_\sigma$. By assumption, Δ has only one edge c or two edges c_1, c_2 crossing σ. When a and b intersect inside σ, the intersection lies in Δ if and only if the triple (a, b, c), or each of the triples (a, b, c_1), (a, b, c_2), has a prescribed orientation, reflecting the condition that the point $a \cap b$ lies on the side of c (or the sides of c_1, c_2) containing Δ. This orientation (or pair of orientations) can be positive, negative, or zero, depending on the relative order of the slopes of a, b, and c (or of c_1 and c_2), and on whether Δ lies to the left or to the right of c (or of c_1, c_2).

For each halfplane c^+ that represents a triangle $\Delta \in C_\sigma$ (the halfplane contains Δ and is bounded by the line supporting the single (relevant) edge c of Δ that crosses σ), we want either (i) to represent the set of pairs $(a, b) \in A^l_\sigma \times B^l_\sigma$ that have a prescribed orientation of the triple (a, b, c), as the disjoint union of complete bipartite graphs (*bicliques*), or (ii) to count the number of such pairs. The subtask (i) arises in cases where Δ has two edges crossing σ and is needed for the first level of the data structure, which we query with the first crossing edge of Δ. The subtask (ii) arises in the second level of the structure, which we query with the second crossing edge of Δ, and in cases where only one edge of Δ crosses σ.

As described above, in $O\left((|A^l_\sigma| + |B^l_\sigma|)\log^2(|A^l_\sigma| + |B^l_\sigma|)\right)$ time, we count the number of intersections within σ. As a matter of fact, with a simple modification of the procedure, we can, within the same time bound, represent the set of all pairs of segments $(a, b) \in A^l_\sigma \times B^l_\sigma$ that intersect each other (inside σ) as the disjoint union of bicliques[12] of total size $O\left((|A^l_\sigma| + |B^l_\sigma|)\log^2(|A^l_\sigma| + |B^l_\sigma|)\right)$. This follows from standard planar segment-intersection range searching machinery [17]. In what follows we focus on just one such graph, and to simplify the presentation we denote it as $A^l_\sigma \times B^l_\sigma$, with a slight abuse of notation.

Preparing for Fredman's trick. We use the infrastructure developed by Aronov et al. [6], but adapt it to the order-type context. We preprocess A and B into a data structure that we will then search with the points dual to the lines supporting the edges of the triangles of C. For each $a \in A$, $b \in B$, we define $\gamma_{a,b}$ to be the line that passes through a^* and b^*, where a^* (resp., b^*) is the point dual to a (resp., b). By our general position assumption, $a^* \neq b^*$, so $\gamma_{a,b}$ is well defined. Let Γ_0 denote the set of these n^2 lines. Our goal in task (ii) is to count, for each cell σ of any of the overlays, for each point c^* dual to an edge of a triangle $\Delta \in C_\sigma$, the number of lines of Γ_0 that lie above c^*, the number of lines that are incident to c^*, and the number of lines that lie below c^*. In task (i), we want to represent each of these sets of lines as the disjoint union of a small number of precomputed canonical sets. This calls for preprocessing the arrangement $\mathcal{A}(\Gamma_0)$ into a suitable point-location

[11] Counting these intersections costs only $O\left((|A^l_\sigma| + |B^l_\sigma|)\log(|A^l_\sigma| + |B^l_\sigma|)\right)$ time, but in order to represent them efficiently one needs to pay an extra logarithmic factor, and therefore we stick with the bound $O\left((|A^l_\sigma| + |B^l_\sigma|)\log^2(|A^l_\sigma| + |B^l_\sigma|)\right)$. See also the discussion below.

[12] To simplify the terminology, we refer to the combined size of the vertex sets of a biclique as the *size* of the graph.

data structure, which we will then search with each $c^* \in C^*$, and retrieve the desired data from the outcome of each query. (That is, the compact representation of these sets of lines is derived from the point-location data structure, see, e.g., [2], for this standard step.)

As in, e.g., [6], a naïve implementation of this approach will be too expensive, as there are too many lines. Instead, we return to the hierarchical partitions $\Xi(A)$, $\Xi(B)$, and Ξ, and apply the following approach to each of the cells $\sigma = (\tau, \tau')$ of the bottom level Ξ_s. To this end, define $\Gamma_\sigma := \{\gamma_{a,b} \mid (a,b) \in A_\sigma \times B_\sigma\}$. In principle, we want to construct the separate arrangements $\mathcal{A}(\Gamma_\sigma)$, over the cells σ, preprocess each of them into a point-location data structure, and search, for each triangle $\Delta \in C$, in the structures that correspond to the cells of Ξ that are either crossed by (at most) one or two edges of Δ, or fully contained in Δ. This is also too expensive if implemented naïvely, so we use instead Fredman's trick (see below), combined with the machinery developed in Section 2.

Preparing for the search with the triangles of C. We first observe that, for each triangle $\Delta \in C$, finding the cells σ (at any level of the hierarchy) that Δ fully contains (but does not contain their parent cell from the previous level) is easy and inexpensive. We go over the hierarchy of the overlays Ξ_j. At the root we find, by brute force, all the (constantly many) cells of Ξ_1 that Δ fully contains, and add their intersection counts to our output counter. We then recurse, in the same manner, in the at most tr_0 cells of Ξ_1 that Δ crosses. Thus the number of cells we visit is at most $O(r_0^2) \cdot \left(1 + tr_0 + (tr_0)^2 + \cdots + (tr_0)^s\right) = O(r^{1+\varepsilon})$, so the overall cost of this step[13] is $O\left(nr^{1+\varepsilon}\right) = O\left(n^{2+\varepsilon}/g\right)$.

We therefore focus, for each triangle Δ of C, only on the cells that it crosses (at every level of the hierarchy), and restrict the analysis for now to cells at which Δ is long, with at most two of its edges crossing the cell (as promised, cells crossed by all three edges and cells containing a vertex will be processed separately, later). Repeating most of the analysis just given, the number of these cells is $O(r^{1+\varepsilon})$ (with a smaller constant of proportionality, since we now do not have the factor $O(r_0^2)$, as above).

Constructing $\mathcal{A}(\Gamma_\sigma)$ in the decision-tree model. Consider the step of constructing $\mathcal{A}(\Gamma_\sigma)$ for some fixed bottom-level cell σ. Following the technique in Section 2, we perform this step using only the order type of Γ_σ, and we begin by considering the task of obtaining the order-type information itself. That is, we want to determine, for each ordered triple $(\gamma_{a_1,b_1}, \gamma_{a_2,b_2}, \gamma_{a_3,b_3})$ of lines of Γ_σ, whether the point $\gamma_{a_1,b_1} \cap \gamma_{a_2,b_2}$ lies to the left or to the right of, or coincides with the point $\gamma_{a_1,b_1} \cap \gamma_{a_3,b_3}$. Let $G(a_1,a_2,a_3;b_1,b_2,b_3)$ denote the 12-variate polynomial (of constant degree) whose sign determines the outcome of the above comparison. (The immediate expression for G is a rational function, which we turn into a polynomial by multiplying it by the square of its denominator, without affecting its sign; our general position assumption ensures that no denominator vanishes.)

Once the signs of all expressions $G(a_1,a_2,a_3;b_1,b_2,b_3)$ are determined, we can apply Lemma 2.1. Recall that it yields a preprocessing that constructs a discrete representation of the arrangement (say, in the DCEL format [10]), and turns this representation into an efficient point-location data structure, and this latter part of the preprocessing can be carried out at no cost in the algebraic decision-tree model. Next we describe how to determine the signs of all expressions $G(a_1,a_2,a_3;b_1,b_2,b_3)$ in an efficient manner.

We search the hierarchical overlay structure Ξ with each triangle $\Delta \in C_\sigma$. We may assume that Δ is long in σ and that only one or two edges of Δ cross σ; the other cases have been or will be handled separately. Assuming further that there is only one such edge c, locating the dual point c^* in $\mathcal{A}(\Gamma_\sigma)$ takes $O(\log g)$ time, as shown in Section 2 (noting that Γ_σ consists of only g^2 lines). With suitable preprocessing, locating c^* gives us, for free in our model, the three sets of the lines that pass above c^*, are incident to c^*, or pass below c^*. The case where two edges of Δ cross σ is handled using a two-level version of the structure; see below for details. The point-location cost now goes up to $O(\log^2 g)$.

Consider then the step of computing the order type of the lines of Γ_σ, that is, of computing the sign of $G(a_1,a_2,a_3;b_1,b_2,b_3)$, for every triple of segments $a_1,a_2,a_3 \in A_\sigma$ and every triple of segments $b_1,b_2,b_3 \in B_\sigma$. To this end, we play Fredman's trick. We fix a bottom-level cell τ of $\Xi(A)$. For each triple $(a_1,a_2,a_3) \in A_\tau^3 := A_\tau \times A_\tau \times A_\tau$, we define the surface[14]

$$\psi_{a_1,a_2,a_3} := \{(b_1,b_2,b_3) \in \mathbb{R}^6 \mid G(a_1,a_2,a_3;b_1,b_2,b_3) = 0\},$$

and denote by Ψ the collection of these surfaces, over all cells τ. We have

$$N := |\Psi| = O((n/g)^{1+\varepsilon} \cdot g^3) = O(n^{1+\varepsilon} g^2).$$

Similarly, we let P denote the set of all triples (b_1,b_2,b_3), for $b_1,b_2,b_3 \in B_{\tau'}^3 := B_{\tau'} \times B_{\tau'} \times B_{\tau'}$, over all cells τ' of $\Xi(B)$. We have $M := |P| = O(n^{1+\varepsilon} g^2)$. These bounds pertain to the bottommost level of the hierarchy; they are smaller at levels

[13] It is for making this step efficient that we use hierarchical partitions. A single-shot partition would have forced the query to visit up to $\Theta(r^2)$ such cells, which would make it too expensive.

[14] Note that we can have $a_1 = a_2$ for instance. Such cases can be handled in a similar manner as the general case, except that we now have to work with a slightly different polynomial (e.g., the case where we only have $a_1 = a_2$ would result in a 10-variate polynomial). In what follows, and for the sake of simplicity, we concentrate merely on the general case, which determines the final time complexity of the algorithm.

of smaller indices. Note that we can also dualize the setup, mapping triples (a_1, a_2, a_3) to points, and triples (b_1, b_2, b_3) to surfaces.

We apply a batched point-location procedure to the points of P and the surfaces of Ψ. The output of this procedure is a collection of bicliques of $P \times \Psi$, so that, for each such subgraph $P_\alpha \times \Psi_\alpha$, $G(a_1, a_2, a_3; b_1, b_2, b_3)$ has a fixed sign for all $(b_1, b_2, b_3) \in P_\alpha$ and all $(a_1, a_2, a_3) \in \Psi_\alpha$, see, e.g., [5,17] for the use of such structures in similar contexts. This tells us the desired signs of $G(a_1, a_2, a_3; b_1, b_2, b_3)$, for every pair of triples $(a_1, a_2, a_3) \in A^3_\tau$, $(b_1, b_2, b_3) \in B^3_{\tau'}$, over all pairs of cells $\sigma = (\tau, \tau') \in \Xi(A) \times \Xi(B)$ (with a nonempty intersection), and these signs give us the orientation (i.e., the order of the intersection points) of every triple of lines $\gamma_{a,b}$. That is, we obtain the order type of the lines of Γ_σ. As remarked in Section 2, we may assume that this also includes the sorting of the lines at $x = -\infty$, but, for the sake of concreteness, we will address this simpler task (which admits a more efficient procedure) in some detail later on.

Remark. This shuffling of the pairs (a_1, b_1), (a_2, b_2), (a_3, b_3) into the triples (a_1, a_2, a_3), (b_1, b_2, b_3) and the treatment of the first triple as defining a surface in $\mathbb{R}^6$ and of the second triple as defining a point in $\mathbb{R}^6$ is the realization of Fredman's trick in our context.

3.1. The batched point-location step

We now spell out the details of the batched point-location procedure. It involves points and surfaces in a six-dimensional parametric space, and proceeds by using the recent multilevel polynomial partitioning technique of Agarwal et al. [3, Corollary 4.8].[15] Specialized to our context, it asserts the following result.

Theorem 3.1 *(A specialized version of Agarwal et al. [3, Corollary 4.8]). Given a set Ψ of N constant-degree algebraic surfaces in $\mathbb{R}^6$, a set P of M points in $\mathbb{R}^6$, and a parameter δ, with $0 < \delta < 1/6$, there are finite collections $\Omega_0, \ldots, \Omega_6$ of semi-algebraic cells in $\mathbb{R}^6$ with the following properties.*

- *For each index i, each cell $\omega \in \Omega_i$ is a connected semi-algebraic set of constant complexity.*
- *For each index i and each $\omega \in \Omega_i$, at most $\frac{N}{4|\Omega_i|^{1/6-\delta}}$ surfaces from Ψ cross ω (meaning, as in the planar setup, that they intersect ω but do not fully contain it), and at most $\frac{M}{4|\Omega_i|^{1-\delta}}$ points from P are contained in ω.*
- *The cells partition $\mathbb{R}^6$, in the sense that $\mathbb{R}^6 = \bigsqcup_{i=0}^{6} \bigsqcup_{\omega\in\Omega_i} \omega$, where $\sqcup$ denotes disjoint union.*
- *The sizes of the collections $\Omega_0, \ldots, \Omega_6$ are bounded by a function of δ, and not of $|P|$ and $|\Psi|$.*

The sets in $\Omega_0, \ldots, \Omega_6$ can be computed in $O(N + M)$ expected time, where the constant of proportionality depends on δ, by a randomized algorithm. For each i and for every set $\omega \in \Omega_i$, the algorithm returns a semi-algebraic representation of ω, a reference point inside ω, the subset of surfaces of Ψ that cross ω, the subset of surfaces that fully contain ω (for lower-dimensional cells ω), and the subset of points of P that are contained in ω.

We compute the partition of Theorem 3.1, for a suitable choice of δ, and find, for each $\psi \in \Psi$, the sets $\omega \in \Omega_i$, over all $i = 0, \ldots, 6$, that it crosses, and those that it fully contains. For each i and $\omega \in \Omega_i$, let $P_{i,\omega}$ denote the set of points of P in ω, and let $\Psi_{i,\omega}$ denote the set of surfaces of Ψ that cross ω. We form three bicliques $P_{i,\omega} \times \Psi^0_{i,\omega}$, $P_{i,\omega} \times \Psi^+_{i,\omega}$, and $P_{i,\omega} \times \Psi^-_{i,\omega}$, where $\Psi^0_{i,\omega}$ is the set of surfaces that fully contain ω (and thus also $P_{i,\omega}$), and $\Psi^+_{i,\omega}$ (resp., $\Psi^-_{i,\omega}$) is the set of surfaces for which ω lies fully in their positive (resp., negative) side, that is, the side at which the corresponding values of G are positive (resp., negative). As the parameters of the partition are all constant, the overall size of the vertex sets of these graphs is $O(M + N)$.

For each i and ω, we also have a recursive subproblem that involves $P_{i,\omega}$ and the subset $\Psi_{i,\omega}$ of the surfaces that cross ω. Putting $r_i := |\Omega_i|$, for $i = 0, \ldots, 6$, we have, for each i and ω,

$$|P_{i,\omega}| \le \frac{M}{4r_i^{1-\delta}} \quad \text{and} \quad |\Psi_{i,\omega}| \le \frac{N}{4r_i^{1/6-\delta}}.$$

To handle each recursive subproblem, we pass to the dual 6-space, with the roles of a_1, a_2, a_3 and of b_1, b_2, b_3 swapped (as already noted, such a swap is justified by the complete symmetry of the setup between the parameters a_1, a_2, a_3 and b_1, b_2, b_3), and apply, using Theorem 3.1, a similar partitioning. (We now denote the resulting collections as Ω^*_i and their respective sizes as r^*_i.) We obtain a second collection of bicliques, still of overall size $O(M + N)$, now with a somewhat larger constant of proportionality, and a new set of recursive subproblems. Each of these subproblems can be labeled by the pairs (k, ω) and (ℓ, ω^*), where k is the index i of the primal collection Ω_i containing ω, and ℓ is the index j of the dual collection Ω^*_j containing ω^* (and therefore $k, \ell = 0, \ldots, 6$).

[15] Alternatively, we could use the partitioning technique of Matoušek and Patáková [31], which in a sense is dual to that of [3].

For each quadruple $((k,\omega),(\ell,\omega^*))$, the corresponding primal subproblem involves at most $\frac{M}{4r_k^{1-\delta}}$ points and at most $\frac{N}{4r_k^{1/6-\delta}}$ surfaces, which switch their roles when we pass to the dual, so each cell of the resulting dual partition generates a subproblem that involves at most $\frac{M}{16r_k^{1-\delta}(r_\ell^*)^{1/6-\delta}}$ dual surfaces (or primal points) and at most $\frac{N}{16r_k^{1/6-\delta}(r_\ell^*)^{1-\delta}}$ dual points (or primal surfaces).

We keep flipping between the primal and dual setups in this manner, until one of the parameters (number of points or number of surfaces) becomes smaller than some constant threshold n_0, which is chosen to be sufficiently larger than all the (constant) parameters r_k, r_ℓ^*. When this happens, we solve the problem by brute force, where the running time, and the overall size of the resulting collection of bicliques, are both proportional to the value of the other parameter (number of surfaces or number of points, respectively).

The primal-dual recursion is applied whenever $M \le N^6$ and $N \le M^6$. If $M > N^6$ we recurse only in the primal, and if $N > M^6$ we recurse only in the dual. We terminate, as before, when we reach subproblems where one of the parameters M, N becomes at most n_0.

The resulting recursion has the following performance bounds.

Proposition 3.2. *Let $T(M,N)$ denote the maximum possible sum of the sizes of the vertex sets of the bicliques produced by the recursive process described above, over all input sets of at most M points and at most N surfaces. Then we have*

$$T(M,N) = O\left(M^{6/7+\varepsilon}N^{6/7+\varepsilon} + M^{1+\varepsilon} + N^{1+\varepsilon}\right),$$

for any $\varepsilon > 0$, where the constant of proportionality depends on ε. The same asymptotic bound also holds for the cost (in the uniform model) of constructing these graphs.

Proof. We fix ε, and show, using induction on M and N, that

$$T(M,N) \le A\left(M^{6/7+\varepsilon}N^{6/7+\varepsilon} + M^{1+\varepsilon} + N^{1+\varepsilon}\right), \tag{1}$$

for a suitable constant coefficient A that depends on ε. We use $\delta := \varepsilon/2$ in Theorem 3.1; to simplify the calculations a bit, we will work with δ instead of ε, so we put $\varepsilon = 2\delta$.

The base cases are when either $M \le n_0$ or $N \le n_0$. If, say, $M \le n_0$, then we clearly have the 'brute force' bound $T(M,N) \le n_0 N$, which satisfies the bound in (1) if A is chosen sufficiently large. A symmetric treatment holds when $N \le n_0$. Assume then that (1) holds for all pairs M', N' such that $M' \le M$ and $N' \le N$, where at least one of the inequalities is strict, for some parameters M, N (both greater than n_0), and consider an instance with a set P of M points and a set Ψ of N surfaces.

Assume first that $N^{1/6} \le M \le N^6$. We consider one phase of the primal decomposition followed by one phase of the dual decomposition. Fix a pair (k,ω) (in the primal) and a pair (ℓ,ω^*) (in the dual), where $k,\ell = 0,1,\ldots,6$, and follow the notation introduced above. Apply the induction hypothesis to the dual subproblem at ω^*. As argued above, this subproblem involves at most $\frac{M}{16r_k^{1-\delta}(r_\ell^*)^{1/6-\delta}}$ dual surfaces (or primal points) and at most $\frac{N}{16r_k^{1/6-\delta}(r_\ell^*)^{1-\delta}}$ dual points (or primal surfaces). Hence, by the induction hypothesis, the contribution of this subproblem to $T(M,N)$ is at most

$$A\left(\left(\frac{M}{16r_k^{1-\delta}(r_\ell^*)^{1/6-\delta}}\right)^{6/7+2\delta}\left(\frac{N}{16r_k^{1/6-\delta}(r_\ell^*)^{1-\delta}}\right)^{6/7+2\delta} + \left(\frac{M}{16r_k^{1-\delta}(r_\ell^*)^{1/6-\delta}}\right)^{1+2\delta} + \left(\frac{N}{16r_k^{1/6-\delta}(r_\ell^*)^{1-\delta}}\right)^{1+2\delta}\right).$$

We assume that the numbers r_ℓ^* are the same at each primal subproblem. We make this assumption for simplicity and clarity of presentation, but it can be removed with a more careful analysis. Multiplying by the number $r_k r_\ell^*$ of subproblems with the indices k, ℓ, and simplifying the expressions, the contribution is at most

$$A\left(\frac{M^{6/7+2\delta}N^{6/7+2\delta}}{16^{12/7+4\delta}(r_k r_\ell^*)^{13\delta/21-4\delta^2}} + \frac{(r_\ell^*)^{5/6+2\delta/3+2\delta^2}}{16^{1+2\delta}r_k^{\delta-2\delta^2}}M^{1+2\delta} + \frac{r_k^{5/6+2\delta/3+2\delta^2}}{16^{1+2\delta}(r_\ell^*)^{\delta-2\delta^2}}N^{1+2\delta}\right).$$

Recall however that we are in the range $M \le N^6$ and $N \le M^6$, so we have

$$M^{1+2\delta} \le \frac{M^{6/7+2\delta}N^{6/7+2\delta}}{N^{2\delta}} \qquad \text{and} \qquad N^{1+2\delta} \le \frac{M^{6/7+2\delta}N^{6/7+2\delta}}{M^{2\delta}},$$

as is easily checked. The contribution is thus at most

$$AM^{6/7+2\delta}N^{6/7+2\delta}\left(\frac{1}{16^{12/7+4\delta}(r_k r_\ell^*)^{13\delta/21-4\delta^2}} + \frac{(r_\ell^*)^{5/6+2\delta/3+2\delta^2}}{16^{1+2\delta}r_k^{\delta-2\delta^2}}\cdot\frac{1}{N^{2\delta}} + \frac{r_k^{5/6+2\delta/3+2\delta^2}}{16^{1+2\delta}(r_\ell^*)^{\delta-2\delta^2}}\cdot\frac{1}{M^{2\delta}}\right)$$
$$< \frac{A}{49}M^{6/7+2\delta}N^{6/7+2\delta},$$

provided that n_0 (and thus M and N) are sufficiently large. Finally, multiplying this bound by the 49 possible choices of $k, \ell = 0, 1, \ldots, 6$, the resulting bound is at most $AM^{6/7+2\delta}N^{6/7+2\delta}$, thereby establishing the induction step for this range of M and N.

Consider next the case where $M > N^6$. In this case we only work in the primal. After one level of recursion, for a fixed pair (k, ω), we get r_k subproblems, each involving at most $M/(4r_k^{1-\delta})$ points and at most $N/(4r_k^{1/6-\delta})$ surfaces. Applying the induction hypothesis at each of these subproblems, the contribution of each subproblem to $T(M, N)$ is at most

$$A\left(\left(\frac{M}{4r_k^{1-\delta}}\right)^{6/7+2\delta}\left(\frac{N}{4r_k^{1/6-\delta}}\right)^{6/7+2\delta} + \left(\frac{M}{4r_k^{1-\delta}}\right)^{1+2\delta} + \left(\frac{N}{4r_k^{1/6-\delta}}\right)^{1+2\delta}\right).$$

Multiplying by the number r_k of subproblems, and simplifying the expressions, we get at most

$$A\left(\frac{M^{6/7+2\delta}N^{6/7+2\delta}}{4^{12/7+4\delta}r_k^{13\delta/21-4\delta^2}} + \frac{M^{1+2\delta}}{4^{1+2\delta}r_k^{\delta-2\delta^2}} + \frac{r_k^{5/6+2\delta/3+2\delta^2}N^{1+2\delta}}{4^{1+2\delta}}\right).$$

Since $N < M^{1/6}$, the third term is dominated by the second term, provided that n_0 is sufficiently large (recall that we have chosen it to be much larger than the quantities r_k). Using this fact and multiplying by the number, 7, of values of k, we establish the induction step for this range.

The case $N > M^6$ is handled in a fully symmetric manner, except that we only work in the dual.

The running time of the procedure obeys the same asymptotic upper bound, which is a consequence of the fact that the multi-level cells in $\Omega_0, \ldots, \Omega_6$ and their conflict lists can be computed in $O(M + N)$ time. We omit the easy details.

This completes the proof of the lemma. □

Remarks.

1. We spell out all these technical details because the analysis in [3] did not handle batched range searching problems, so we provide it for the sake of completeness, and also in the hope that it would find additional applications in the future.

2. Proposition 3.2 can be extended to any dimension d, with a similar proof, to obtain a primal-dual range searching algorithm involving M points and N surfaces in d dimensions, assuming full symmetry between the points and surfaces, as above. The running time of the algorithm (in the uniform model) is $O(M^{d/(d+1)+\varepsilon}N^{d/(d+1)+\varepsilon} + M^{1+\varepsilon} + N^{1+\varepsilon})$, for any $\varepsilon > 0$.

3.2. Wrapping up

Using a similar and simpler technique, we can sort the lines of each of the arrangements $\mathcal{A}(\Gamma_\sigma)$, over all cells σ, at $x = -\infty$. (Note that this corresponds to sorting them in reverse order of their slopes.) Here each comparison is between a pair of lines, say γ_{a_1,b_1} and γ_{a_2,b_2}, and its outcome is the sign of some constant-degree 8-variate polynomial (more precisely, a rational function turned into a polynomial) $H(a_1, a_2; b_1, b_2)$. Fredman's trick for this setup leads to a batched point-location procedure that involves $O((n/g)^{1+\varepsilon}g^2) = O(n^{1+\varepsilon}g)$ points and $O((n/g)^{1+\varepsilon}g^2) = O(n^{1+\varepsilon}g)$ surfaces in $\mathbb{R}^4$. This task can be accomplished by a somewhat simpler variant of the technique presented above, whose running time bound is subsumed in the above bound.

In summary, the information collected so far allows us to obtain the combinatorial structure of each of the arrangements $\mathcal{A}(\Gamma_\sigma)$, over all cells σ of Ξ, and subsequently construct an order-type–based point-location data structure for each of them, at no extra cost in the algebraic decision-tree model. The overall cost of this phase, in this model, is thus $O\left((n^{1+\varepsilon}g^2)^{12/7+\varepsilon}\right)$, for any $\varepsilon > 0$. By replacing ε by some small multiple thereof, we can write this bound as $O\left((ng^2)^{12/7+\varepsilon}\right)$, for any $\varepsilon > 0$.

Fredman's trick, as applied above, separates the handling of the conflict lists A_τ, over the trapezoids τ of $\Xi(A)$, and the conflict lists $B_{\tau'}$, over the trapezoids τ' of $\Xi(B)$. For a cell $\sigma = (\tau, \tau')$ of Ξ, not all the segments in A_τ necessarily cross σ, so we have to retain (for σ) only those that do cross it, and apply a similar pruning to $B_{\tau'}$. The cost of this filtering step is $O(g)$ for each σ, for an overall cost of $O((n/g)^2 \cdot g) = O(n^2/g)$. This cost is subsumed by the cost of searching with the elements of C, discussed later.

Interpreted in the dual, the filtering step just described filters out all lines $\gamma_{a,b}$ from Γ_σ that pass through a (dual) point whose (primal) segment does not cross σ, but we also need to filter out lines $\gamma_{a,b}$, where the corresponding (long) segments a and b do not meet inside σ (or do not meet at all). Filtering by inspecting all pairs (a, b) would be too expensive in the uniform model, but, fortunately, we can implement this step free of charge in the decision-tree model. Indeed, consider the bicliques in the compact representation of all the long pairs (a, b) that intersect inside σ (as described in the earlier quick overview for handling long segments). Once this biclique decomposition is available, we simply keep in Γ_σ only those lines that correspond to the edges of these graphs, a step that costs nothing in the decision-tree model, since it does not incur any extra comparisons among the input segments. Once this filtration is performed, we can construct the arrangement of the surviving lines, at no extra cost, and use the modified arrangement for the point-location searches with the elements of C, discussed next.

Searching with the triangles of C. We now need to search the structures computed in the preceding phase with the duals of the (lines containing) the edges of the triangles of C.

Each triangle $\Delta \in C$ crosses only $O(r^{1+\varepsilon}) = O(n^{1+\varepsilon}/g)$ cells of Ξ (without fully containing the cell). Recall that, for a cell σ that Δ crosses, we say that Δ is *long* (resp., *short*) in σ if σ does not contain (resp., contains) a vertex of Δ. There are at most three cells σ at the bottom level of the hierarchy, in which Δ is short, and we simply inspect all the g^2 pairs of segments in $A_\sigma \times B_\sigma$, and include those pairs that intersect inside Δ in our output count for Δ, for a total cost of $O(ng^2)$. It therefore suffices to focus on cells in which Δ is long. Δ is not processed at intermediate-level cells at which it is short; it is only processed as a short triangle at the relevant bottom-level cells.

Let σ be a bottom-level cell. It is easy (and standard) to show that there is at most one cell σ that is crossed by all three edges of Δ, so we can handle these cells as the cells where Δ is short, with comparable efficiency. It thus suffices to assume that σ is crossed by only one or two edges of Δ. In the former case we may replace Δ, for the purpose of searching within σ, by the halfplane bounded by the single edge that crosses σ, and in the latter case we may replace Δ by the intersection wedge of the two halfplanes bounded by the two edges that cross σ. In the former situation we replace Δ by the point c^* dual to the line supporting the single crossing edge, and search the point-location structure constructed for $\mathcal{A}(\Gamma_\sigma)$ with c^*. In the latter situation we replace Δ by the pair of points c_1^*, c_2^* dual to the lines supporting the two crossing edges. We prepare a two-level data structure (see, for example, [2]), where each level is based on the above point-location structure for $\mathcal{A}(\Gamma_\sigma)$, except that the first level collects its output as the disjoint union of canonical sets, and the second level counts intersections within the query triangle. We then search the top level with c_1^* and search the resulting substructures of the second level, for each relevant canonical subset, with c_2^*.

There are $O(nr^{1+\varepsilon}) = O(n^{2+\varepsilon}/g)$ triangle-cell crossings, each requiring $O(\log^2 g)$ search time, using (one or two levels of) the above point-location data structure for each such arrangement, for a total of $O\left(\frac{n^{2+\varepsilon}\log^2 g}{g}\right)$ time. Adding the time to construct the order-type–based point-location structure, the total time needed to handle the long segments is $O\left((ng^2)^{12/7+\varepsilon} + \frac{n^{2+\varepsilon}\log^2 g}{g}\right)$. We (nearly) balance the terms in this bound by taking $g = n^{2/31}$, so the cost of this procedure, in the algebraic decision-tree model, is $O(n^{2-2/31+\varepsilon}) = O(n^{60/31+\varepsilon})$, for a slightly larger value of ε.

We next have to handle short segments and short triangles within cells of Ξ, including triangles that have three edges crossing the cell (for the latter we repeat and refine some of the details described earlier). As might be expected, this part is less expensive than the handling of long segments and triangles, as we now show.

Handling short segments. There are two main tasks that we have to implement for short segments: (i) count the number of intersection points that involve a short segment and another segment, at all cells of the overlays at all levels, and (ii) preprocess them so that, for each bottom-level cell σ, we can count, for each query triangle Δ, the number of intersection points with a short segment within σ that lie inside Δ. We start with the first task.

A segment of either A or B can be short in at most two cells, at each level of the hierarchy. For each cell σ at any fixed level j, let n_σ denote the number of short segments (from $A_\sigma \cup B_\sigma$) in σ, so we have $\sum_{\sigma\in\Xi_j} n_\sigma \le 4n$. For each cell σ, the overall number of segments that cross σ is at most $2n/r_0^j$.

Thus, at each cell σ at level j, we count the number of intersections between the n_σ short segments and the at most $2n/r_0^j$ other (long or short) segments. Using an algorithm of Agarwal [1], this takes

$$O\left(n_\sigma^{2/3}(n/r_0^j)^{2/3+\varepsilon} + n_\sigma^{1+\varepsilon} + (n/r_0^j)^{1+\varepsilon}\right)$$

time (also in the uniform model).[16] Recall that the number of cells σ in Ξ_j is $O((tr_0)^{2j})$, where t is an absolute constant. Hence, by using Hölder's inequality, the sum of the above bounds over the cells σ is at most

$$\begin{aligned} &O\left(\left(\sum_\sigma n_\sigma\right)^{2/3}(tr_0)^{2j/3}(n/r_0^j)^{2/3+\varepsilon} + n^{1+\varepsilon} + (tr_0)^{2j}\cdot(n/r_0^j)^{1+\varepsilon}\right)\\ &= O\left(n^{2/3}\cdot\frac{t^{2j/3}n^{2/3}}{r_0^{j\varepsilon}} + n^{1+\varepsilon} + t^{2j}r_0^{(1-\varepsilon)j}n^{1+\varepsilon}\right).\end{aligned}$$

Recalling that $r_0^j \le r_0^s = O(r^{1+\varepsilon}) = O(n^{1+\varepsilon}/g)$, this can be upper bounded by $O(n^{4/3+\varepsilon} + n^{1+\varepsilon}r_0^j) = O(n^{2+\varepsilon}/g)$, for a slightly larger ε (assuming that $g < n^{2/3}$, as indeed will be the case), a cost that is subsumed by that of other steps of the algorithm.

Consider next the second task, of counting the number of intersection points inside a query triangle that involve a short segment, at the bottom-level cells. The overall number of such intersection points is only $O(ng)$, and we compute all of

[16] The actual bound in [1] contains polylogarithmic factors rather than factors of the form n^ε. This, however, does not affect the bounds derived by our analysis—see below.

them by brute force, and distribute them among the cells. For each bottom-level cell σ, let P_σ denote the set of these points in σ, and let C_σ denote, as above, the set of triangles that cross σ, with only one or two crossing edges. To simplify the presentation, we only consider triangle-cell crossings for which the triangle has just one crossing edge, so it behaves as a halfplane in the cell. The case of two crossing edges is handled, as above, via a two-level data structure. Put $M_\sigma := |P_\sigma|$ and $N_\sigma := |C_\sigma|$, for each cell σ, and observe that (i) $\sum_\sigma M_\sigma = O(ng)$, (ii) $\sum_\sigma N_\sigma = O(n^{2+\varepsilon}/g)$, and (iii) $N_\sigma \le n$ for each σ.

Applying the standard machinery for halfspace range counting [2,4], we can count the number of points of P_σ that lie inside (the halfplanes representing) each of the triangles in C_σ, in time $O\left(M_\sigma^{2/3}N_\sigma^{2/3+\varepsilon} + M_\sigma^{1+\varepsilon} + N_\sigma^{1+\varepsilon}\right)$, for each cell σ. Summing this bound over σ, using Hölder's inequality, we get a total of

$$\begin{aligned}
&\sum_\sigma O\left(M_\sigma^{2/3}N_\sigma^{2/3+\varepsilon} + M_\sigma^{1+\varepsilon} + N_\sigma^{1+\varepsilon}\right)\\
&\quad = O\left(n^{1/3+\varepsilon}\right)\cdot\sum_\sigma M_\sigma^{2/3}N_\sigma^{1/3} + O\left(n^{1+\varepsilon}g + n^{2+\varepsilon}/g\right)\\
&\quad = O\left(n^{1/3+\varepsilon}(ng)^{2/3}(n^{2+\varepsilon}/g)^{1/3} + n^{1+\varepsilon}g + n^{2+\varepsilon}/g\right)\\
&\quad = O\left(n^{5/3+\varepsilon}g^{1/3} + n^{1+\varepsilon}g + n^{2+\varepsilon}/g\right).
\end{aligned}$$

This bound is subsumed in the overall bound on the cost of the other steps of the algorithm (again, assuming that g is not too large).

Handling short triangles and triangles with three crossing edges. As we have already noted, the overall cost of this part is $O(ng^2)$. Indeed, each triangle Δ is short in at most three cells, at each level of the hierarchy. However, we need to count intersection points inside a short triangle only at the bottom-level cells where the triangle is short. For each such cell σ, we count for each triangle Δ that is short in σ, by brute force, the number of intersection points inside $\Delta \cap \sigma$. This has a total cost of $O(ng^2)$, well below our overall bound. The same argument applies to triangles with three edges crossing a bottom cell.

We remark that the analysis of these parts of the algorithm, which deal with short segments or triangles, also applies in the uniform model.

Putting it all together. In conclusion, we finally have:

Theorem 3.3. *Let A and B be two sets each consisting of n pairwise disjoint segments in the plane, and let C be a set of n triangles in the plane. We can count, for each triangle $\Delta \in C$, the number of intersection points of segments of A with segments of B that lie inside Δ, in the algebraic decision-tree model, at the subquadratic cost $O(n^{60/31+\varepsilon})$, for any $\varepsilon > 0$.*

Corollary 3.4. *We can solve, in the algebraic decision-tree model, at the cost of $O(n^{60/31+\varepsilon})$, for any $\varepsilon > 0$, each of the problems (i) intersection of three polygons, (ii) coverage by three polygons, and (iii) segment concurrency, as listed in the introduction.*

4. Extensions

In this section we present two additional applications of the paradigm developed in this paper.

4.1. Circular arc intersection counting

We have two sets A, B, each consisting of n pairwise disjoint circular arcs in the plane, and a third set C, consisting of n circles in the plane. Our goal is to count, for each circle $c \in C$, the number of intersection points of an A-arc with a B-arc that lie in the interior of c. Denote by $\bar\gamma$ the circle containing γ, for each arc $\gamma \in \mathcal{A} \cup B$. Put $\bar A := \{\bar a \mid a \in A\}$ and $\bar B := \{\bar b \mid b \in B\}$, for the respective sets of the containing circles.

Using the standard lifting transform, each circle $\bar a \in \bar A$ is lifted to a 'red' plane a^* in $\mathbb{R}^3$, and each circle $\bar b \in \bar B$ is lifted to a 'blue' plane b^* in $\mathbb{R}^3$. For each $\bar a \in \bar A$ and $\bar B \in \bar B$, the line $\lambda_{a,b} := a^* \cap b^*$ intersects the standard paraboloid Π in at most two points, and the lifted images of the at most two intersection points of the arcs supported by a and b form a subset of zero, one, or two of these points. See Fig. 4(i). Let P denote the set of those points on Π that correspond to actual intersection points of an A-arc and a B-arc; we cannot afford to compute P explicitly. We have $|P| \le 2n^2$. Given a circle $c \in C$ (call such circles 'green'), we want to count the number of points of P that lie below or on the plane c^*.

We dualize the setup in $\mathbb{R}^3$, using the standard duality that preserves the above/below relationship, and get a set P^* of at most $2n^2$ 'red-blue' dual planes (all tangent to Π). The goal is now to locate the points dual to the planes of C^* in the arrangement $\mathcal{A}(P^*)$ of the planes of P^*. More precisely, we want to count how many planes pass below (or through) each query point.

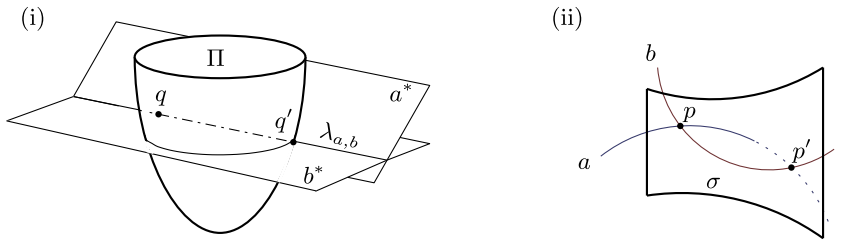

Fig. 4. Illustrating the setup for orientation testing. (i) The planes a^* and b^*, which are lifted images of a pair of intersecting circles $\bar{a} \in \bar{A}$ and $\bar{b} \in \bar{B}$, intersect in a line $\lambda_{a,b}$ that crosses the paraboloid Π twice. (ii) The setup in the plane: p is the projection of q and p' is the projection of q'. The circles $\bar{a}$ and $\bar{b}$ meet at p and p', but only p is relevant, because it is also an intersection of the arcs a and b, and it lies inside the cell σ.

We thus face the problem of point location in a three-dimensional arrangement $\mathcal{A}(P^*)$ of a set P^* of $O(n^2)$ 'red-blue' planes, each determined by a red arc in A and a blue arc in B. Of course, we cannot afford the construction of the full arrangement, so we play Fredman's trick, as in Section 3. That is, we construct a $(1/r)$-cutting Ξ_A for the A-arcs, and a $(1/r)$-cutting Ξ_B for the B-arcs, each of complexity $O(r)$ (which follows since the arcs in each set are pairwise disjoint; the argument of [11] applies here), construct the overlay Ξ of these cuttings, and process each cell σ of Ξ separately. We actually need to construct hierarchical cuttings, as in Section 3, and, at each cell σ of the overlay hierarchy of Ξ, at any level, we also need to count the overall number of intersections of A-arcs and B-arcs that lie inside σ (this information will be needed when σ is fully contained inside a circle of C). As before, we classify each arc of $A \cup B$ at a cell σ as *long* (resp., *short*) if the cell does not contain (resp., contains) an endpoint of the arc. It suffices to focus on long arcs, as short arcs can be handled in much the same way as in Section 3.

As demonstrated in Section 3, this subtask—counting the number of intersections between long arcs inside a give cell σ—is very easy for (long) segments, but is more challenging for circular arcs. Still, using the algorithm of Agarwal et al. [5], this can be done, for the long arcs within each cell σ, in $O(N_\sigma^{3/2+\varepsilon})$ time, for any $\varepsilon > 0$, where $N_\sigma := |A_\sigma| + |B_\sigma|$, and A_σ (resp., B_σ) is the set of (long) arcs of A (resp., B) that cross σ. We have $|A_\sigma|, |B_\sigma| \le g := n/r$, so $N_\sigma \le 2g$.

At each bottom-level cell σ of the hierarchy of Ξ, we need to construct, and preprocess for fast point location, the arrangement $\mathcal{A}(P^*_\sigma)$, where P^*_σ is the set of all dual red-blue planes in $\mathbb{R}^3$ that are determined by an arc of A_σ and an arc of B_σ.

We now use the machinery developed in Section 2. Here we need to perform orientation tests for quadruples of planes in P^*_σ, and Fredman's trick allows us to represent each such test as a sign test of some constant-complexity algebraic predicate G in 24 variables, 12 variables for the parameters of the four circles of A participating in the test, and 12 variables for the parameters of the four circles of B.

In more detail, each plane participating in the orientation test is dual to a point that is the intersection of Π with some line $\lambda_{a,b}$. There are at most two such points, but for such a point to actually materialize, it needs to belong to the two arcs a, b. We assume for now that each of these arcs is long in σ (the other cases are much easier to handle). We first need to distinguish between the two possible points, which differ by the sign of the square root in the solution of the resulting quadratic equation. Once that is done, and the four intersection points participating in the orientation test have been identified (there are up to 16 possible such identifications of the quadruple), the test itself is the sign test of a fixed algebraic expression. However, in order for the sign test to be meaningful, we need to assert that each of the four relevant points p (within the xy-plane) exist and lie in σ (since we assume that our arcs are long in σ, this suffices to ensure that the two arcs do indeed intersect at p). See Fig. 4 for an illustration. Put together, all these constraints define our desired predicate G, which clearly is of constant complexity.

We transform these tests into point-location tests of g^4 points, formed by quadruples of circular arcs of A, in an arrangement of g^4 surfaces, formed by quadruples of circular arcs of B, in $\mathbb{R}^{12}$, or the other way around (since we will also use duality, in which we flip the roles of A and B, so that the dual points are determined by quadruples of arcs of B and the dual surfaces are determined by quadruples of arcs of A). Again, since the arcs are assumed to be long, specifying the three real parameters that define the containing circle, and knowing σ, uniquely identifies the arc. (The intersection of an arc with a cell σ does not have to be connected. If it is not connected, we treat each of its connected components as a separate arc.)

We group together all these points and surfaces, over the $O(r) = O(n/g)$ bottom-level cells of Ξ_A (for the points) and of Ξ_B (for the surfaces) into single respective collections Q, Σ, consisting of $O((n/g)^{1+\varepsilon} g^4) = O(n^{1+\varepsilon} g^3)$ points and surfaces, respectively. (As just mentioned, and as we did in Section 3, we use duality, so we also treat Q as a collection of dual surfaces and treat Σ as a collection of dual points in $\mathbb{R}^{12}$.) Adapting the machinery in Section 3.1 (see the remark at the end of that section), we can solve the latter point-location problem in time

$$O\left((n^{1+\varepsilon} g^3)^{24/13}\right) = O\left(n^{24/13+2\varepsilon} g^{72/13}\right),$$

for any $\varepsilon > 0$. Handling short arcs is simpler and is done analogously to the treatment in Section 3. From the output of this procedure we can construct, using the random sampling machinery in Section 2.1, and at no extra cost in the algebraic

decision-tree model, a data structure for point location in $\mathcal{A}(P^*_\sigma)$, for each bottom-level cell σ of Ξ, where the cost of a query is $O(\log g)$. Arguing as in the preceding section, each circle of C has to perform this search at only $O(n^{1+\varepsilon}/g)$ cells, so the total cost of the point-location searches with the circles of C is $O\left(\frac{n^{2+\varepsilon}\log g}{g}\right)$. Roughly balancing this cost with the preprocessing cost, we choose $g := n^{2/85}$, and the overall cost of the procedure is the subquadratic bound of $O\left(n^{168/85+2\varepsilon}\right)$, for any $\varepsilon > 0$. Scaling ε, we obtain the following theorem:

Theorem 4.1. *Given sets A, B, each of n pairwise disjoint circular arcs in the plane, and a set C of n circles in the plane, we can count, for each circle $c \in C$, the number of intersection points of an A-arc with a B-arc that lie in the interior of c, in $O\left(n^{2-2/85+\varepsilon}\right) = O\left(n^{168/85+\varepsilon}\right)$ time, for any $\varepsilon > 0$, in the algebraic decision-tree model.*

4.2. Points and lines in the plane: minimum distance problems

In the problem studied in this subsection we have two sets A, B, each of n points in the plane, and a third set C of n lines in the plane, and the goal is to determine whether there exists a triple $(a, b, c) \in A \times B \times C$, such that c contains a point x that satisfies some property involving $\mathsf{dist}(x, a)$ and $\mathsf{dist}(x, b)$. For a concrete example, to be expanded below, given a prescribed parameter $t > 0$, determine whether any line $c \in C$ contains a point whose sum of distances to its nearest neighbor in A and its nearest neighbor in B is at most t. Equivalently, determine whether any $c \in C$ intersects any ellipse of major axis t whose pair of foci lie in $A \times B$.

The problem, in detail. Let A and B be two sets, each of n points in the plane, and let C be a set of n lines in the plane. Consider predicates of the form (where a and b are points, c is a line, and t is a real number)

$$\pi(a, b, c; t) := \exists x \in c \mid F(\mathsf{dist}(x, a), \mathsf{dist}(x, b)) \le t, \tag{2}$$

where F is a constant-degree bivariate piecewise algebraic function that is monotone increasing in both variables, and dist is the Euclidean distance. Typical examples are $F(u, v) = u + v$, $F(u, v) = \max\{u, v\}$, or $F(u, v) = u^2 + v^2$. Our goal is to determine whether there exists a triple $(a, b, c) \in A \times B \times C$ such that $\pi(a, b, c; t)$ holds. For example, when $F(u, v) = \max\{u, v\}$, the goal is to determine whether there exists a line of C that contains a point that lies at distance at most t from a point of A and from a point of B. Similarly, when $F(u, v) = u + v$, the goal is to determine whether there exists a line $c \in C$ that intersects any 'bichromatic' ellipse of major axis t that is spanned by a focus in A and a focus in B. Problems of this kind are special instances of facility location, where we want to determine whether there exists a line of C that contains a point whose distance from A and distance from B satisfy some property. Alternatively, we can aim at reporting all lines of C with this property.

Problems of this kind arise in instances where the points of A contain one type of resource (e.g., milk) and those of B contain another type of resource (e.g., honey), and we want to find on each element $c \in C$, say a road, a location from which access to a type-A resource and to a type-B resource is cheap, in a specific sense according to the measure we use (maximum distance, sum of distances, and the like).

A more ambitious goal (but perhaps not that much more) would be to find the minimum value of t for which there exist $(a, b, c) \in A \times B \times C$ such that $\pi(a, b, c; t)$ holds, or for which every c has a pair (a, b) such that $\pi(a, b, c; t)$ holds. For example, for $F(u, v) = u + v$, find the smallest major axis of a bichromatic ellipse of this kind that is crossed by some line of C, or find the smallest major axis of a bichromatic ellipse so that every line of C crosses such an ellipse.

We will consider here only the former setup, in which t is prespecified. It seems likely that the problem of optimizing t could be solved using parametric search.

The problems studied here can be generalized in several ways, for example by replacing the lines of C by constant-degree algebraic curves, or by replacing the Euclidean distance by more general distance functions, but we will deal only with the problem as formulated above.

For each triple $(a, b, c) \in A \times B \times C$, eliminate x from the expression in (2) that determines $\pi(a, b, c; t)$, to obtain a semi-algebraic region $G(a, b; t)$ in the dual plane (in which lines are represented as points), of constant complexity, so that $c \in G(a, b; t)$ if and only if $\pi(a, b, c; t)$ holds.

The algorithm. We present a solution for the above problem, that runs in (strictly) subquadratic time in the algebraic decision-tree model.[17] We remark that the problem can be solved in quadratic time in the uniform model, as follows from the algorithm that we derive; see a comment below to this effect.

By the preceding discussion, we face the problem of point location (of the points dual to the lines of C) in the planar arrangement $\mathcal{A}(\mathcal{G})$ of the set $\mathcal{G}$ of the 'red-blue' regions $G(a, b; t)$, each being a semi-algebraic set of constant complexity, and determined by a pair of a red point $a \in A$ and a blue point $b \in B$. As with the previous problems, we cannot afford the construction of the full arrangement, so we play Fredman's trick, similar to, but in a somewhat different context than, the technique in Section 3.

[17] We believe that these are 3SUM-hard problems, although we have not yet established this property.

We take a random sample R_A of r points from A, and a random sample R_B of r points from B, construct their Voronoi diagrams $\mathrm{Vor}(R_A)$ and $\mathrm{Vor}(R_B)$, and triangulate each cell of the diagrams by triangles emanating from the site of the cell. We denote the resulting triangulated diagrams as Ξ_A and Ξ_B, respectively. Each triangulated diagram has complexity $O(r)$. Each cell τ of Ξ_A (resp., of Ξ_B) has an associated conflict list A_τ (resp., B_τ), of those points of A (resp., of B) that can be closer to a point in τ than the site of τ. With high probability, the size of each conflict list is at most $O\left(\frac{n}{r}\log r\right)$. We overlay Ξ_A and Ξ_B, and obtain a subdivision Ξ of the plane, with $O(r^2)$ constant-complexity cells. (Here there is no need for a hierarchical decomposition, like the ones used in Sections 3 and 4.1.)

Let σ be a cell of Ξ, formed by the intersection of a cell τ of Ξ_A and a cell τ' of Ξ_B, and let A_σ (resp., B_σ) be the points of A_τ (resp., of $B_{\tau'}$) that can be closer to points in σ than the corresponding sites.

A line $c \in C$ crosses only $O(r)$ cells of Ξ. Within each such cell σ, each point $x \in c \cap \sigma$ needs to find its nearest neighbor a_x in A, among the points of A_σ, and its nearest neighbor b_x in B, among the points of B_σ, and then test whether there exists $x \in c \cap \sigma$ such that $F(\mathsf{dist}(x, a_x), \mathsf{dist}(x, b_x)) \le t$. To do so, within each of these $O(r)$ cells, we need to locate the dual point c^* of c in the arrangement $\mathcal{A}(\mathcal{G}_\sigma)$, where[18] $\mathcal{G}_\sigma := \{G(a, b; t) \mid a \in A_\sigma,\ b \in B_\sigma\}$. More precisely, we need to determine whether c^* lies in any of these regions.

We now use the machinery developed in Section 2. Here we need to perform orientation tests for triples of boundary curves of the sets $G(a, b; t)$, for $(a, b) \in \Gamma := \bigcup_\sigma (A_\sigma \times B_\sigma)$. The curves bounding the regions $G(a, b; t)$ are not necessarily x-monotone and may be bounded. This requires some modification of the technique of Section 2.2, which, albeit technically somewhat involved, are nonetheless rather straightforward conceptually, and we omit their details, in the interest of brevity.

The construction of the order type of the curves bounding the regions $G(a, b; t)$ amounts to performing various tests, each of which involves three pairs in Γ, plus some additional parameters that specify which curves we test and what the two intersection points that we compare are. We employ Fredman's trick, which transforms each such test, involving three pairs (a_1, b_1), (a_2, b_2), (a_3, b_3), to testing whether the point $(b_1, b_2, b_3) \in \mathbb{R}^6$ belongs to a certain semi-algebraic region Q_{a_1,a_2,a_3}, which consists of all points (u_1, u_2, u_3) such that (a_1, u_1), (a_2, u_2), (a_3, u_3) satisfy the conditions in the test. Glossing over some technical issues, this amounts to batched point location of $O(ng^2)$ points in an arrangement of $O(ng^2)$ surfaces in $\mathbb{R}^6$. Applying the machinery in Section 3.1, this can be done in time $O\left(n^{12/7+\varepsilon} g^{24/7+\varepsilon}\right)$, for any $\varepsilon > 0$. This allows us to construct the arrangements $\mathcal{A}(\mathcal{G}_\sigma)$, over the cells σ, preprocess each of these arrangements for fast point location, as in the preceding applications, at no extra cost in the decision-tree model.

As before, searching with the lines of C takes $O\left(\frac{n^{2+\varepsilon}}{g}\log^2 g\right)$ time, and balancing the two costs yields the earlier bound $O\left(n^{60/31+\varepsilon}\right)$, for any $\varepsilon > 0$. That is, we have

Theorem 4.2. *Let A and B be two sets, each of n points in the plane, let C be a set of n lines in the plane, let F be a constant-degree bivariate piecewise algebraic function that is monotone increasing in both variables, and let t be a real parameter. We can determine, for each line $c \in C$ whether it contains a point x that satisfies $F(\mathsf{dist}(x, a), \mathsf{dist}(x, b)) \le t$, where dist is the Euclidean distance. The algorithm works in the algebraic decision-tree model, and takes $O\left(n^{60/31+\varepsilon}\right)$ time, for any $\varepsilon > 0$.*

Remark. As promised, we note that the above algorithm can be adapted, in a much simplified form, to obtain a quadratic algorithm for the problem in the uniform model. To do so, we construct the full Voronoi diagrams $\mathrm{Vor}(A)$ and $\mathrm{Vor}(B)$, and form their triangulated overlay Ξ. This step takes $O(n^2)$ time. For each cell σ of Ξ, all its points have the same nearest neighbor a_σ in A and the same nearest neighbor b_σ in B. Then, for each $c \in C$, we find the $O(n)$ cells of Ξ that c crosses, and, for each such cell σ, we need to test whether c^* lies in $G(a_\sigma, b_\sigma; t)$, an operation that takes $O(1)$ time. The overall cost of this step is also $O(n^2)$.

5. Discussion

As promised in the introduction, we make some comments on the differences between this work and the work of Aronov et al. [6], which tackle problems that have some features in common. Both works use Fredman's trick, implemented by a batched range-search mechanism, in which objects in one input set form points and objects in another set form surfaces in some suitable parametric space. However, the analysis in [6] works in the dual plane and uses hierarchical polynomial partitioning for points (dual to the lines in the input). This mechanism works efficiently only in the special case where one of the input sets consists of arbitrary points in the plane, and the other two sets are contained in *one-dimensional* curves. In this work, we apply a decomposition in the primal plane (the plane of the segments), and use hierarchical cuttings, where the crucial property in the analysis is that each set A, B consists of pairwise disjoint segments. This results in a special, low-complexity structure, which our analysis exploits. In addition, we present a new primal-dual range searching mechanism, exploiting and expanding the recent multi-level polynomial partitioning technique of [3]. This mechanism is fairly general and we feel that it could be used in other range-searching applications, and is therefore of independent interest.

Another major difference is the use of order types to construct the various arrangements $\mathcal{A}(\Gamma_\sigma)$. The fact that each comparison that we make involves only three objects of A and three of B, allows us to transform it into a test that involves

[18] Actually, there is no need to filter away points from A_τ, $B_{\tau'}$, to get A_σ, B_σ. Keeping all the points from each set does not affect the solution.

a point and a surface in six dimensions. In contrast, the standard technique, based on persistent data structures, calls for sorting the vertices of $\mathcal{A}(\Gamma_\sigma)$ in the x-direction, and then each test involves a point and a surface in eight dimensions. This makes the resulting range-searching machinery considerably less efficient. It is an interesting topic for further research to find additional applications of this paradigm. Puzzlingly, the use of order types seems inapplicable to the most efficient method presented in [6].

Finally, it would be interesting to modify our techniques so as to obtain (slightly) subquadratic algorithms for these problems in the uniform model; see Chan [15].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

We thank Zuzana Patáková for helpful discussions on multilevel polynomial partitioning.

References

[1] Pankaj K. Agarwal, Partitoning arrangements of lines II: applications, Discrete Comput. Geom. 5 (1990) 533–573, https://doi.org/10.1007/BF02187809.
[2] Pankaj K. Agarwal, Simplex range searching, in: Martin Loebl, Jaroslav Nešetřil, Robin Thomas (Eds.), A Journey Through Discrete Mathematics, Springer, 2017, pp. 1–30.
[3] Pankaj K. Agarwal, Boris Aronov, Esther Ezra, Joshua Zahl, Efficient algorithm for generalized polynomial partitioning and its applications, SIAM J. Comput. 50 (2) (2021) 760–787, https://doi.org/10.1137/19M1268550.
[4] Pankaj K. Agarwal, Jeff Erickson, Geometric range searching and its relatives, in: Bernard Chazelle, Jacob E. Goodman, Richard Pollack (Eds.), Advances in Discrete and Computational Geometry, in: Contemp. Math., vol. 223, AMS Press, 1999, pp. 1–56.
[5] Pankaj K. Agarwal, Marco Pellegrini, Micha Sharir, Counting circular arc intersections, SIAM J. Comput. 22 (4) (1993) 778–793, https://doi.org/10.1137/0222050.
[6] Boris Aronov, Esther Ezra, Micha Sharir, Testing polynomials for vanishing on Cartesian products of planar point sets: collinearity testing and related problems, Discrete Comput. Geom. (2022), https://doi.org/10.1007/s00454-022-00437-1, in press. Also in 36th International Symposium on Computational Geometry, SoCG 2020, pp. 8:1–8:14, https://doi.org/10.4230/LIPIcs.SoCG.2020.8.
[7] Luis Barba, Jean Cardinal, John Iacono, Stefan Langerman, Aurélien Ooms, Noam Solomon, Subquadratic algorithms for algebraic 3Sum, Discrete Comput. Geom. 61 (4) (2019) 698–734, https://doi.org/10.1007/s00454-018-0040-y.
[8] Saugata Basu, Richard Pollack, Marie-Françoise Roy, Algorithms in Real Algebraic Geometry, 2nd edition, Algorithms and Computation in Mathematics, vol. 10, Springer, 2006.
[9] Michael Ben-Or, Lower bounds for algebraic computation trees (preliminary report), in: Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 80–86, https://doi.org/10.1145/800061.808735.
[10] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, Mark H. Overmars, Computational Geometry: Algorithms and Applications, 3rd edition, Springer, 2008, https://www.worldcat.org/oclc/227584184.
[11] Mark de Berg, Otfried Schwarzkopf, Cuttings and applications, Int. J. Comput. Geom. Appl. 5 (4) (1995) 343–355, https://doi.org/10.1142/S0218195995000210.
[12] Jürgen Bokowski, Simon King, Susanne Mock, Ileana Streinu, The topological representation of oriented matroids, Discrete Comput. Geom. 33 (4) (2005) 645–668, https://doi.org/10.1007/s00454-005-1164-4.
[13] Jürgen Bokowski, Susanne Mock, Ileana Streinu, On the Folkman-Lawrence topological representation theorem for oriented matroids of rank 3, Eur. J. Comb. 22 (5) (2001) 601–615, https://doi.org/10.1006/eujc.2000.0482.
[14] Jean Cardinal, John Iacono, Aurélien Ooms, Solving k-SUM using few linear queries, in: 24th Annual European Symposium on Algorithms, ESA 2016, in: LIPIcs, vol. 57, 2016, pp. 25:1–25:17, https://doi.org/10.4230/LIPIcs.ESA.2016.25.
[15] Timothy M. Chan, More logarithmic-factor speedups for 3Sum, (median, +)-convolution, and some geometric 3Sum-hard problems, ACM Trans. Algorithms 16 (1) (2020) 7:1–7:23, https://doi.org/10.1145/3363541.
[16] Timothy M. Chan, Da Wei Zheng, Hopcroft's problem, log-star shaving, 2D fractional cascading, and decision trees, in: Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, (SODA'22), 2022, pp. 190–210, arXiv:2111.03744, https://doi.org/10.1137/1.9781611977073.10.
[17] Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, Micha Sharir, Algorithms for bichromatic line-segment problems and polyhedral terrains, Algorithmica 11 (2) (1994) 116–132, https://doi.org/10.1007/BF01182771.
[18] Bernard Chazelle, Leonidas J. Guibas, Fractional cascading: I. A data structuring technique, Algorithmica 1 (2) (1986) 133–162, https://doi.org/10.1007/BF01840440.
[19] Herbert Edelsbrunner, Leonidas J. Guibas, Jorge Stolfi, Optimal point location in a monotone subdivision, SIAM J. Comput. 15 (2) (1986) 317–340, https://doi.org/10.1137/0215023.
[20] Esther Ezra, Sariel Har-Peled, Haim Kaplan, Micha Sharir, Decomposing arrangements of hyperplanes: VC-dimension, combinatorial dimension, and point location, Discrete Comput. Geom. 64 (1) (2020) 109–173, https://doi.org/10.1007/s00454-019-00141-7.
[21] Ezra Esther, Micha Sharir, A nearly quadratic bound for point-location in hyperplane arrangements, in the linear decision tree model, Discrete Comput. Geom. 61 (4) (2019) 735–755, https://doi.org/10.1007/s00454-018-0043-8.
[22] Jon Folkman, Jim Lawrence, Oriented matroids, J. Comb. Theory, Ser. B 25 (2) (1978) 199–236, https://doi.org/10.1016/0095-8956(78)90039-4.
[23] Anka Gajentaan, Mark H. Overmars, On a class of $O(n^2)$ problems in computational geometry, Comput. Geom. 5 (1995) 165–185, https://doi.org/10.1016/0925-7721(95)00022-2.
[24] Omer Gold, Micha Sharir, Improved bounds for 3SUM, k-SUM, and linear degeneracy, in: 25th Annual European Symposium on Algorithms, ESA 2017, in: LIPIcs, vol. 87, 2017, pp. 42:1–42:13, https://doi.org/10.4230/LIPIcs.ESA.2017.42.

[25] Jacob E. Goodman, Richard Pollack, Multidimensional sorting, SIAM J. Comput. 12 (3) (1983) 484–507, https://doi.org/10.1137/0212032.
[26] Jacob E. Goodman, Richard Pollack, Semispaces of configurations, cell complexes of arrangements, J. Comb. Theory, Ser. A 37 (3) (1984) 257–293, https://doi.org/10.1016/0097-3165(84)90050-5.
[27] Allan Grønlund, Seth Pettie, Threesomes, degenerates, and love triangles, J. ACM 65 (4) (2018) 22:1–22:25, https://doi.org/10.1145/3185378.
[28] David Haussler, Emo Welzl, ε-nets and simplex range queries, Discrete Comput. Geom. 2 (1987) 127–151, https://doi.org/10.1007/BF02187876.
[29] Daniel M. Kane, Shachar Lovett, Shay Moran, Near-optimal linear decision trees for k-SUM and related problems, J. ACM 66 (3) (2019) 16:1–16:18, https://doi.org/10.1145/3285953.
[30] D.T. Lee, Franco P. Preparata, Location of a point in a planar subdivision and its applications, SIAM J. Comput. 6 (3) (1977) 594–606, https://doi.org/10.1137/0206043.
[31] Jiří Matoušek, Zuzana Patáková, Multilevel polynomial partitions and simplified range searching, Discrete Comput. Geom. 54 (1) (2015) 22–41, https://doi.org/10.1007/s00454-015-9701-2.
[32] Stefan Meiser, Point location in arrangements of hyperplanes, Inf. Comput. 106 (2) (1993) 286–303, https://doi.org/10.1006/inco.1993.1057.
[33] Neil Sarnak, Robert Endre Tarjan, Planar point location using persistent search trees, Commun. ACM 29 (7) (1986) 669–679, https://doi.org/10.1145/6138.6151.
[34] Jack Snoeyink, Point location, in: J.E. Goodman, J. O'Rourke, C.D. Tóth (Eds.), Handbook of Discrete and Computational Geometry, 3rd edition, CRC Press, 2008, pp. 1005–1028.
[35] V.V. Williams, On some fine-grained questions in algorithms and complexity, in: Procs. International Congress of Mathematicians, 2018, pp. 3447–3487, https://doi.org/10.1142/9789813272880_0188.