Computational Geometry: Theory and Applications 110 (2023) 101963

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

journal homepage: www.elsevier.com/locate/comgeo

Time and space efficient collinearity indexing ☆

Boris Aronov [a], Esther Ezra [b,*], Micha Sharir [c], Guy Zigdon [b]

[a] Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201, USA
[b] School of Computer Science, Bar Ilan University, Ramat Gan, Israel
[c] School of Computer Science, Tel Aviv University, Tel Aviv, Israel

ARTICLE INFO

Article history:
Received 12 June 2022
Received in revised form 27 October 2022
Accepted 3 November 2022
Available online 8 November 2022

Keywords:
Computational geometry
3SUM-indexing
Collinearity testing
Function inversion

ABSTRACT

The *collinearity testing* problem is a basic problem in computational geometry, in which, given three sets A, B, C in the plane, of n points each, the task is to detect a collinear triple of points in $A \times B \times C$ or report there is no such triple. In this paper we consider a preprocessing variant of this question, namely, the *collinearity indexing* problem, in which we are given two sets A and B, each of n points in the plane, and our goal is to preprocess A and B into a data structure, so that, for any query point $q \in \mathbb{R}^2$, we can determine whether q is collinear with a pair of points $(a, b) \in A \times B$. We provide a solution to the problem for the case where the points of A, B lie on an integer grid, and the query points lie on a vertical line, with a data structure of subquadratic storage and sublinear query time. We then extend our result to the case where the query points lie on the graph of a polynomial of constant degree. Our solution is based on the function-inversion technique of Fiat and Naor [11].

1. Introduction

Let A, B, C be three sets of points in the plane. The *collinearity testing* problem is to determine whether there exists a collinear triple (a, b, c) with $a \in A$, $b \in B$, $c \in C$. This classical problem is at least as hard as the 3SUM *problem* [13], in which we are given three sets X, Y, and Z, each consisting of n real numbers, and we would like to determine whether there exists a triple of numbers $(x, y, z) \in X \times Y \times Z$ that add up to zero.[1]

The 3SUM problem itself, conjectured for a long time to require $\Omega(n^2)$ time, was shown by Grønlund and Pettie [16] (with further improvements by Chan [7]) to be solvable in very slightly subquadratic time. Moreover, in the *linear decision-tree model*, in which we only count linear sign tests involving the input point coordinates (and do not allow any other operation to access the input explicitly), Grønlund and Pettie improved the running time to nearly $O(n^{3/2})$ (see also [12,15] for subsequent slight speedups), which was drastically further improved (still in the linear decision-tree model) to $O(n \log^2 n)$ time by Kane et al. [17]. In contrast, no subquadratic algorithm is known for the collinearity testing problem, either in the standard real RAM model (also known as the *uniform* model) or in the decision-tree model; see [4,5] for a discussion. Very

☆ Work by Boris Aronov 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 Esther Ezra was partially supported by NSF CAREER under grant CCF:AF-1553354 and by Grant 824/17 from the Israel Science Foundation. Work by Micha Sharir was partially supported by Grant 260/18 from the Israel Science Foundation.

* Corresponding author.
E-mail addresses: boris.aronov@nyu.edu (B. Aronov), ezraest@cs.biu.ac.il (E. Ezra), michas@tauex.tau.ac.il (M. Sharir), guy.zigdon@live.biu.ac.il (G. Zigdon).

[1] In fact, Gajentaan and Overmars [13], who introduced this concept, initially called this problem (as well as other related geometric problems) "n^2-hard," as it was strongly believed that it cannot be solved in subquadratic time (whereas it has a simple $O(n^2)$ time solution).

https://doi.org/10.1016/j.comgeo.2022.101963
0925-7721/

recently several input-restricted variants of collinearity testing have been studied in the decision tree model and also in the RAM model [3–5,7], however, the unrestricted case has still remained elusive (that is, the case where the input consists of arbitrary points in the plane with no further assumptions).

In this paper we study a preprocessing variant of collinearity testing, known as *collinearity indexing*, which we review next.

1.1. Collinearity indexing

Let A and B be two sets, each of n points in the plane. The *collinearity indexing* problem is to preprocess A and B into a data structure, so that, given any query point $q \in \mathbb{R}^2$, we can determine using the data structure whether there exists a pair $(a, b) \in A \times B$ such that a, b, and q are collinear. The goal is to design such a data structure that uses subquadratic storage and answers queries in sublinear time; see a more detailed discussion of this issue below. Notice that in this formulation we ignore the preprocessing time, which can even be $\omega(n^2)$, but only care about storage and query time.[2]

We first note that we can assume that A and B are disjoint (a situation that is easy to detect), for otherwise every query has a positive outcome, and the problem becomes trivial.

Near-linear query time is easy to obtain, with no additional storage. To do so, one simply sorts the points of $A \cup B$ in angular order around the query point q, and checks every consecutive pair, in this order, of an A-point and a B-point, for collinearity with q. The cost of the query is $O(n \log n)$.

We can improve the query time to $O(n)$ at the cost of quadratic storage. To do so, we pass to the dual plane, to obtain a set A^* of n lines dual to the points of A, and a set B^* of n lines dual to the points of B. We compute the arrangement $\mathcal{A}$ of $A^* \cup B^*$, and store its doubly-connected edge list (DCEL) representation; see, e.g., [9, Chapter 2]. Then, given a query point q, we take its dual line q^* and trace the faces of $\mathcal{A}$ that q^* crosses, checking whether q^* passes through a "bichromatic" vertex of $\mathcal{A}$, incident to a line of A^* and a line of B^*, which is the dual interpretation of the property that q is collinear with a point of A and a point of B. Using the DCEL representation and the *zone theorem* (which states that the overall zone complexity of a line in a planar arrangement of lines, namely the overall complexity of all the faces that the line crosses, is linear in the number of lines), this takes $O(n)$ time, see, e.g., [9].

We do not know whether the query cost can be made sublinear, still using quadratic storage, nor whether the storage can be reduced to subquadratic, still allowing linear query. These two problems are discussed below, and are left as challenging, seemingly hard, open problems, although they appear to be simpler than the problem that we address in this work.

We can obtain sublinear query time with superquadratic storage. There are several equivalent ways to describe such a procedure, but one of the simpler ways is to take the set V of the $O(n^2)$ bichromatic vertices of the arrangement $\mathcal{A}$, and preprocess it for halfplane range searching (see, e.g., [1,2]). It is easy to adapt the resulting procedure so that it can detect whether the query line q^* passes through a vertex in V. With s storage, a query takes $O^*(n^2/\sqrt{s})$ time, which is sublinear when s is superquadratic.[3] In the extreme case, when $s = \Theta(n^4)$, the query time becomes $O(\log n)$. Indeed, in this case, the above approach essentially constructs the primal arrangement of the $O(n^2)$ bichromatic lines, each connecting an A-point with a B-point, and preprocesses the arrangement for fast point location, using $O(n^4)$ storage and $O(\log n)$ query time.

In view of this discussion, the following problems arise:

(i) Preprocess A and B into a data structure that requires $O(n^2)$ storage and can answer a collinearity query in $o(n)$ time.
(ii) Preprocess A and B into a data structure that requires $o(n^2)$ storage and can answer a collinearity query in $O(n)$ time.
(iii) Preprocess A and B into a data structure that requires $o(n^2)$ storage and can answer a collinearity query in $o(n)$ time.

Of course, a solution to Problem (iii) will automatically solve the other two problems, but it is conceivable, and very likely, that the first two problems are easier to solve, although at the moment the solution to any of Problems (i)–(iii) remains elusive.

Our result In general, A and B may contain arbitrary points in the plane, and the queries are also arbitrary points. We consider special instances of these problems in which the locations of the points of $A \cup B$ and/or of the query points are restricted. We first consider the case where the points of A and B lie on an integer grid, and the query points lie *anywhere* on a vertical line, say the y-axis. For this case we show, in Section 2:

Theorem 1.1. *Let A, B be two sets in the plane, of n points each, both lying on an integer grid of polynomial size.[4] For any $0 < \delta < 1$ there exists a deterministic data structure of overall storage $O^*(n^{2-\delta/3})$, which answers collinearity-indexing queries on the y-axis in $O^*(n^\delta)$ time.*

By choosing $\delta = 3/4$, we conclude that there is a solution to Problem (iii) under the above assumptions, that is:

[2] We note that a related problem, referred to as "3POL-indexing," was defined in [14] in the context of lower bounds.
[3] Here and in what follows, $O^*(\cdot)$ hides a factor of n^ε, for any $\varepsilon > 0$.
[4] See Section 2 for the precise definition.

Corollary 1.2. *Given the setting of Theorem 1.1, there exists a deterministic data structure of overall storage $O^*(n^{7/4})$, which answers collinearity-indexing queries on the y-axis in $O^*(n^{3/4})$ time.*

In Section 3 we extend Theorem 1.1 to the case where the query points lie on the graph of a polynomial of constant degree, with coefficients of bounded bit complexity.

We note that our solution is an adaptation of the technique of Golovnev et al. [14] (see also Kopelowitz and Porat [18]), which solves the simpler 3SUM-*indexing* problem: That is, the goal is to preprocess two sets X, Y of n real numbers each, such that given a real number z as a query, one can quickly determine whether there is a pair $x \in X$, $y \in Y$ such that $x + y + z = 0$ (see also [10] for a related problem). As in [14,18], our solution is based on the function-inversion technique of Fiat and Naor [11]—see below.

A naïve solution for the case where A and B are unrestricted As just discussed, our main result holds for the case where A and B lie on a common integer grid, but we first discuss a more general setup.

Let A and B be two sets, each of n (arbitrary) points in the plane, and let γ be a vertical line containing the query points. Without loss of generality, we assume that γ is the y-axis. The collinearity indexing problem, restricted to γ, is to preprocess A and B into a data structure, so that, given any query point $q \in \gamma$, we can determine whether there exists a pair $(a, b) \in A \times B$ such that a, b, and q are collinear.

The problem is easy to solve using $O(n^2 \log n)$ preprocessing time, $O(n^2)$ storage, and $O(\log n)$ query time, as follows. For each pair of points $a \in A$ and $b \in B$, we construct the line $\lambda_{a,b}$ that connects a and b, and compute its intersection point with the y-axis. Then, we sort these points according to their y-value in $O(n^2 \log n)$ time into an array. Given a query point q (that lies on the y-axis), we need to determine whether it lies on any of the lines $\lambda_{a,b}$, which can simply be performed with a binary search in $O(\log n)$ time in the sorted array obtained above. We also recall from the above discussion that we can solve the problem with no preprocessing, using $O(n)$ storage, and $O(n \log n)$ query time (where the query point does not necessarily lie on the y-axis). Our goal, however, is to obtain a data structure that uses subquadratic storage and sublinear query time.

2. The case where *A* and *B* lie on an integer grid

We next present a more efficient solution for this problem, under the following further restrictions. We assume that the points of A and of B lie at (some of the) vertices of a $k \times k$ integer grid G, with integer coordinates (so $n \le k^2$), where k is bounded by a polynomial function of n. For concreteness, and without loss of generality, assume that $G = [t, k + t - 1] \times [0, k - 1]$ (t is an integer parameter that indicates where the y-axis is located with respect to G; t is also assumed to be bounded by a polynomial function of n). We continue to denote by γ the y-axis, and assume that the query points lie on γ.[5]

We will hereafter assume that no point of $A \cup B$ lies on γ. If γ contains a point of A and a point of B then every query has a positive outcome and the problem becomes trivial. If γ contains points of only A, say, then these points can form collinear triples only when q coincides with one of them, a situation that is easy to detect.

Our strategy is to consider all bichromatic lines that connect a point of A with a point of B. Let $a = (x_a, y_a) \in G$ and $b = (x_b, y_b) \in G$ be two grid points. The line $\lambda_{a,b}$ that passes through a and b has the equation

$$\frac{y - y_a}{x - x_a} = \frac{y_b - y_a}{x_b - x_a}, \qquad \text{or}$$

$$(y_b - y_a)(x - x_a) - (x_b - x_a)(y - y_a) = 0, \qquad \text{or}$$

$$(y_b - y_a)x - (x_b - x_a)y = x_a y_b - y_a x_b.$$

Since a and b belong to G, we have

$$y_b - y_a,\ x_b - x_a \in [-(k-1),\ (k-1)], \qquad \text{and}$$
$$x_a y_b - y_a x_b \in [-t(k-1) - (k-1)^2,\ t(k-1) + (k-1)^2].$$

Let Λ denote the set of all lines $\lambda_{a,b}$, over all $a, b \in G$. Note that $|\Lambda| = O(k^4)$. A non-vertical line $\lambda \in \Lambda$ intersects the y-axis at the point with y-coordinate

$$y = \frac{x_a y_b - y_a x_b}{x_a - x_b},$$

which is a rational with numerator in $[-t(k-1) - (k-1)^2,\ t(k-1) + (k-1)^2]$ and denominator in $[-(k-1),\ (k-1)]$.

[5] As in [14], the model of computation in this setting is the word RAM model where we assume that the word length is $O(\log k)$, in which case the bit complexity of any input point is $O(\log k)$ (specifically, each point is represented by a pair of integer numbers whose bit complexity is $O(\log k)$). The size of a data structure in this model is the number of words it contains, and query time (as well as preprocessing time) is the number of elementary word operations on the input.

Set

$$U = \left\{ \left(0, \frac{i}{k^2}\right) \mid |i| \le tk^3 + k^4 \right\},$$

that is, U is a set of evenly spaced points on the y-axis. It is easily checked that (i) each intercept of a line of Λ with the y-axis lies in an interval delimited by two consecutive points of U, and (ii) each of these intervals contains at most one such intercept. Indeed, (i) follows from the definition of U and the respective values of the y-coordinates, and (ii) holds since the smallest difference between two such intercepts is at least $1/(k-2) - 1/(k-1)$, which is larger than $1/k^2$. We also comment that we regard these intervals as half-open in order to uniquely associate intercepts with these intervals, in case an intercept hits a point of U. (An intercept can be shared by many lines of Λ, but distinct intercepts lie in distinct intervals.) Let V denote the (multi-)set of these intercepts.

Preparing for the Fiat-Naor setup Write $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$. Following the considerations in [14,18], we define a function $g\colon [n] \times [n] \mapsto \gamma$ by setting $g(i, j)$ equal to the (unique) intersection point of γ with the (non-vertical) line $\ell_{i,j} = \lambda_{a_i b_j}$. If $\ell_{i,j}$ is vertical, $g(i, j)$ is undefined; in such a case, we do not map the pair (i, j), since the corresponding points a_i, b_j do not form a collinear triple with any of the points on γ. The image of g is the multi-set V.

Let $h\colon U \mapsto [n] \times [n]$ be a *universal hash function*, randomly picked from a corresponding set of functions. That is, for each pair $x \ne y \in U$, we have $\Pr[h(x) = h(y)] \le \frac{1}{n^2}$, where probability is with respect to the random choice of h.[6] One can construct such a function by encoding U as a range of integers (using only the numerators of its elements), and encoding $[1, n] \times [1, n]$ as $[1, n^2]$, using, e.g., the lexicographical order, and use the class of functions of the form $h(i) = (ci + d \pmod p) \pmod{n^2}$, where $p > n^2$ is a prime number and c, d are random integers modulo p, with $c \ne 0$ (see, e.g., [8, Chapter 11]). Finally, define a function $f\colon [n] \times [n] \mapsto [n] \times [n]$ by

$$f(i, j) = h(\mathrm{flr}_\delta(g(i, j))),$$

for $(i, j) \in [n] \times [n]$, where $\delta = \frac{1}{k^2}$ is the separation parameter between any pair of consecutive elements of U, and flr_δ maps each point $q \in \gamma$ to the largest element of U that is smaller than or equal to q.

Building the data structure We now use the trade-off result of Fiat and Naor [11] for inverting a function:

Proposition 2.1 ([11]). *Let $L = [n] \times [n]$, and put $N := |L|$. For any function $f : L \to L$, and for any choice of values (S, T) such that $T \cdot S^3 = N^3$, there exists an algorithm for inverting f that uses $O^*(S)$ words of space and inverts f in $O^*(T)$ time.*[7]

Proposition 2.1 implies that, for any choice of values (S, T) that obey $TS^3 = n^6$, one can preprocess f (in overall time as specified above), into a data structure that uses $O^*(S)$ storage, so that, for any (i, j) in the range of f, one can compute, in $O^*(T)$ time, with probability $1 - \frac{1}{n^2}$, a pair in $f^{-1}(i, j)$. Since f may be many-to-one, $f^{-1}(i, j)$ may consist of many elements; in this case, the Fiat-Naor procedure returns just one of them. This is sufficient for our analysis as such a scenario implies that we may have several potential pairs $(a, b) \in A \times B$, which are collinear with the query point q, but reporting any single pair among them suffices—see below.

An element z of the range $R \subseteq U$ of $\mathrm{flr}_\delta \circ g$ is said to be an *h-singleton* if for any $z' \ne z$ in R we have $h(z) \ne h(z')$. Standard properties of universal hash functions (see once again [14,18]) ensure that the expected number of singleton elements in R is at least some constant fraction of $|R|$. Hence, if we repeat this scheme $\Theta(\log n)$ times, drawing h independently at each incarnation, the expected number of elements that are not singletons in all the schemes becomes smaller than 1, and we can therefore assume that our structure has the property that every element of R is a singleton in at least one instance, with high probability. Following the considerations in [14,18], we note that since the algorithm is allowed to have a large preprocessing time, we can select the $\Theta(\log n)$ functions h while deterministically guaranteeing that each z is an h-singleton for some function h that we selected.

We query with a point $q \in \gamma$ in each of these $O(\log n)$ structures, collect $O(\log n)$ potential inverses of f at $h(\mathrm{flr}_\delta(q))$, and claim, using the above reasoning, that if $q \in R$ then at least one of these candidates is a pair (i, j) such that a_i, b_j, and q are collinear. We go over the candidates and select the one that satisfies this property. If such a candidate exists then we found a pair in $(a, b) \in A \times B$ with which q is collinear. Otherwise, if none of the candidates satisfies the property, we conclude that q is not collinear with any pair in $A \times B$.

In summary, we obtain $O(\log n)$ structures, whose overall space complexity is $O^*(S)$, and the total query time is $O^*(T)$ (where T, S are any pair of parameters satisfying $TS^3 = n^6$), as is easily verified. This implies that by setting $T = n^\delta$, we obtain $S = n^{2-\delta/3}$, for any $0 < \delta < 1$. Any such choice makes S subquadratic and T sublinear. For example, we can choose $\delta = 3/4$, and then $T = n^{3/4}$ and $S = n^{7/4}$. This completes the proof of Theorem 1.1 and Corollary 1.2. □

[6] In the following discussion we comment that the choice of h can be derandomized—see below.

[7] The overall expected preprocessing time is $O^*(n^2)$, but, as observed in [14], the data structure of Fiat and Naor [11] can be derandomized using large preprocessing with similar asymptotic space and query bounds. We are using this latter variant in our analysis.

3. An extension: the query points lie on the graph of a polynomial

We now consider the case where the query points lie on the graph of a polynomial, which we continue to denote by γ. Technically, this excludes the case where γ is a vertical line, the case addressed in the previous section, as a vertical line is not the graph of a polynomial of the form that we consider. Nevertheless, by flipping the roles of the coordinate axes, we can include this case too. We continue to assume that the points in A, B lie on an integer grid, as above. Specifically, assume that γ is the graph of $y = Q(x)$, where $Q(\cdot)$ is a polynomial of constant degree $D \geq 1$, with integer coefficients bounded by a polynomial function of k. We believe that the technique can also be extended to the case where γ is the zero set of a bivariate polynomial of constant degree with integer coefficients bounded by a polynomial function of k, however, this extension raises many technical algebraic complications that we have decided not to address.

Let $a = (x_a, y_a) \in G$ and $b = (x_b, y_b) \in G$ be a pair of points on G. As before, the line $\lambda_{a,b}$ that passes through a and b has the equation

$$(y_b - y_a)x - (x_b - x_a)y = x_a y_b - y_a x_b, \tag{1}$$

and intersects γ in at most D points, whose x-coordinates are the roots of the polynomial

$$Q_{(a,b)}(x) := (y_b - y_a)x - (x_a y_b - y_a x_b) - (x_b - x_a)Q(x)$$

of degree D. Let p^* be one of the intersection points of $\lambda_{a,b}$ and γ; its x-coordinate p^*_x is a root of $Q_{(a,b)}(x)$. Let $\lambda_{a',b'}$ be a line passing through another pair of points $a', b' \in G$, and let p'^* be one of the intersection points of $\lambda_{a',b'}$ with γ. Provided that $p^* \neq p'^*$, our goal is to obtain a minimum separation bound for the x-coordinates of p^* and p'^*. In order to do so, we construct the *product polynomial* of the line equations for $\lambda_{a,b}$ and $\lambda_{a',b'}$, which is the polynomial[8]

$$F_{(a,b),(a',b')}(x, y) := \Big((y_b - y_a)x - (x_b - x_a)y - x_a y_b + y_a x_b\Big)\Big((y_{b'} - y_{a'})x - (x_{b'} - x_{a'})y - x_{a'} y_{b'} + y_{a'} x_{b'}\Big). \tag{2}$$

The zero set $Z(F)$ of $F_{(a,b),(a',b')}(x, y)$ is the union of the two lines $\lambda_{a,b}$ and $\lambda_{a',b'}$, and we are interested in the intersection points of this pair of lines with γ. These points are obtained by substituting $y = Q(x)$, from which we obtain a polynomial $P_{(a,b),(a',b')}(x) := F_{(a,b),(a',b')}(x, Q(x))$ of degree $2D$. We denote the maximum bit complexity of its coefficients by τ_P, and observe that it is bounded by $O(\tau + \log k) = O(\log k)$, where τ is the maximum bit complexity of the coefficients of Q (by our above assumption $\tau = O(\log k)$), and the $\log k$ term is contributed by the x- and the y-values of the points a, b, a', and b'.

Let us denote by $sep(P)$ the minimal distance between two roots of $P_{(a,b),(a',b')}$. Using a key property on separation bounds shown in [6, Corollary 10.22], we have:

$$sep(P) \geq (2D/\sqrt{3})^{-1}(2D)^{-D}(2D+1)^{(1-2D)/2}2^{\tau_P(1-2D)}.$$

It is easy to verify that

$$\rho := sep(P) \geq (2D)^{-2D-3/2}k^{-\alpha D},$$

for an absolute constant $\alpha > 0$, and therefore the separation is polynomial in k.

Next, using another key property, stated in [6, Corollary 10.4], it follows that the absolute values of the roots of $P_{(a,b),(a',b')}$ are bounded by $2^{O(\tau+\log k+\log D)} = 2^{O(\log k)} := k^c$, for an absolute constant $c > 0$. (This estimate holds since D is assumed to be a constant and $k = \Omega(\sqrt{n})$.)

We next mark along the curve γ points, such that the x-distances between pairs of consecutive points is ρ, in the x-range $[-k^c, k^c]$. This results in a partition of γ into "canonical intervals," whose endpoints are (in the notation below i is an integer):

$$U = \left\{(x_i,\ Q(x_i)) \mid\ x_i = i\rho \quad \text{for} \quad |i| \leq k^c/\rho\right\}.$$

Since the root separation bound described above applies to any pair of lines $\lambda_{a,b}$ and $\lambda_{a',b'}$ that intersect γ, it follows that each canonical interval with endpoints at $(x_i,\ Q(x_i))$ and $(x_{i+1},\ Q(x_{i+1}))$ contains at most one intercept of a line $\lambda_{a,b}$ with γ (which, as above, may be shared by many lines).

Next, define $g\colon [n] \times [n] \times [1, \ldots, D] \to \gamma$ by setting $g(i, j, l)$ to be the lth leftmost intersection point of $\lambda_{i,j}$ with γ (the function is undefined for indices l for which the lth intersection point does not exist). Then, let $h\colon U \to [n] \times [n] \times [1, \ldots, D]$ be a universal hash function, as described above. Finally, set $f\colon [n] \times [n] \times [1, \ldots, D] \to [n] \times [n] \times [1, \ldots, D]$ to be

$$f(i, j, l) = h(\mathrm{flr}_\rho(g(i, j, l))),$$

[8] We comment that this analysis also holds for the case where one of $\lambda_{a,b}$, $\lambda_{a',b'}$ is a vertical line.

for $(i, j, l) \in [n] \times [n] \times [1, \ldots, D]$, where flr_ρ maps each point $q \in \gamma$ to the left endpoint of the canonical interval in U containing it.

We can now apply almost verbatim the analysis of the preceding case where γ was the y-axis, using the trade-off bound of Fiat and Naor [11], and following similar considerations concerning h-singleton functions and the search of a query point $q \in \gamma$. Note that in this case the complexity of the domain and range of the function f is only $O(n^2)$ since D is constant, and therefore the Fiat-Naor tradeoff bounds remain asymptotically the same (although the constants in the bounds are larger and depend on D). We thus conclude:

Theorem 3.1. *Let A, B be two sets, each consisting of n points in the plane that lie on an integer grid of size polynomial in n. Let γ be the graph of a polynomial of constant degree with coefficients of bounded bit complexity, as above. For any $0 < \delta < 1$ there exists a deterministic data structure of overall storage complexity $O^*(n^{2-\delta/3})$, which answers collinearity-indexing queries, with respect to points on γ, in $O^*(n^\delta)$ time. In particular, when $\delta = 3/4$ the storage and query bounds are $O^*(n^{7/4})$ and $O^*(n^{3/4})$, respectively.*

Concluding remarks and open problems In this paper we made progress in the study of collinearity indexing for the case where the input sets A, B lie on an integer grid and the queries lie on a one-dimensional curve. A main open problem is to obtain a data structure with subquadratic storage and sublinear query time for the case where the points in $A \cup B$ lie on the integer grid and the queries are unrestricted. In this case we face the challenge that a line $\lambda_{a,b}$ passing through $a \in A$ and $b \in B$ may contain arbitrarily many potential query points, due to which the Fiat-Naor technique may result in an inefficient data structure.

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 Tsvi Kopelowitz and Ely Porat for helpful discussions.

References

[1] P.K. Agarwal, Simplex range searching and its variants: a review, in: M. Loebl, J. Nešetřil, R. Thomas (Eds.), Journey Through Discrete Mathematics: A Tribute to Jiří Matoušek, Springer Verlag, Berlin-Heidelberg, 2017, pp. 1–30.
[2] P.K. Agarwal, J. Erickson, Geometric range searching and its relatives, in: B. Chazelle, J.E. Goodman, R. Pollack (Eds.), Advances in Discrete and Computational Geometry, in: Contemp. Math., vol. 223, AMS Press, Providence, RI, 1999, pp. 1–56.
[3] B. Aronov, M. de Berg, J. Cardinal, E. Ezra, J. Iacono, M. Sharir, Subquadratic algorithms for some 3Sum-hard geometric problems in the algebraic decision tree model, Comput. Geom. Theory Appl. 109 (2023) 101945. Also in Proc. 32nd Int. Sympos. Alg. Comput. (ISAAC 2021), 3:1–3:15.
[4] B. Aronov, E. Ezra, M. 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. Also in Proc. 36th Sympos. on Computational Geometry (2020), 8:1–8:14, and in arXiv:2003.09533.
[5] L. Barba, J. Cardinal, J. Iacono, S. Langerman, A. Ooms, N. Solomon, Subquadratic algorithms for algebraic 3Sum, Discrete Comput. Geom. 61 (2019) 698–734. Also in Proc. 33rd Sympos. on Computational Geometry (2017), 13:1–13:15.
[6] S. Basu, R. Pollack, M.-F. Roy, Algorithms in Real Algebraic Geometry, Algorithms and Computation in Mathematics, Springer-Verlag, Berlin, Heidelberg, 2006.
[7] T.M. Chan, More logarithmic-factor speedups for 3Sum, (median,+)-convolution, and some geometric 3Sum-hard problems, ACM Trans. Algorithms 16 (2020) 7:1–7:23.
[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, MIT Press, 2009.
[9] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry, Algorithms and Applications, 3rd edition, Springer Verlag, 2008.
[10] A. Dumitrescu, W.L. Steiger, Space-time trade-offs for some ranking and searching queries, Inf. Process. Lett. 79 (5) (2001) 237–241.
[11] A. Fiat, M. Naor, Rigorous time/space tradeoffs for inverting functions, SIAM J. Comput. 29 (3) (1999) 790–803.
[12] A. Freund, Improved subquadratic 3sum, Algorithmica 77 (2) (2017) 440–458.
[13] A. Gajentaan, M.H. Overmars, On a class of $O(n^2)$ problems in computational geometry, Comput. Geom. Theory Appl. 5 (1995) 165–185.
[14] A. Golovnev, S. Guo, T. Horel, S. Park, V. Vaikuntanathan, Data structures meet cryptography: 3sum with preprocessing, in: Proc. 52nd Annu. ACM SIGACT Sympos. Theory Comput. (STOC 2020), 2020, pp. 294–307. Also in arXiv:1907.08355.
[15] O. Gold, M. Sharir, Improved bounds for 3sum, k-SUM, and linear degeneracy, in: Proc. European Sympos. Algorithms, 2017, 42:1–42:13. Also in arXiv:1512.05279.
[16] A. Grønlund, S. Pettie, Threesomes, degenerates, and love triangles, J. ACM 65 (2018) 22:1–22:25.
[17] D.M. Kane, S. Lovett, S. Moran, Near-optimal linear decision trees for k-SUM and related problems, J. ACM 66 (3) (2019) 16:1–16:18. Also in arXiv: 1705.01720.
[18] T. Kopelowitz, E. Porat, The strong 3sum-INDEXING conjecture is false, arXiv:1907.11206.