Space-Efficient Collinearity Testing

Boris Aronov* Esther Ezra† Micha Sharir‡ Guy Zigdon§

Abstract

The *collinearity testing problem* is a basic question in computational geometry: given a set P of n points in the plane, we are asked to determine whether P contains a collinear triple. We present a self-reduction for this problem, in the real-RAM model, that can be used to decrease the space complexity of collinearity testing algorithms. As a result, collinearity testing can be performed in $O(n^2)$ expected time using only $O(n^{1/2})$ working storage.

Introduction

Let P be a set of n points in the plane. The *collinearity testing* problem is to determine whether there exists a collinear triple (a, b, c) of (distinct) points in P. This classical problem in computational geometry is 3SUM-hard, i.e., at least as hard as the 3SUM *problem* [10], in which N is a set of n real numbers, and we want to determine whether there exist a triple (x, y, z) of numbers in N that add up to zero.

The 3SUM problem itself, conjectured for a long time to require $\Omega(n^2)$ time, was shown by Grønlund and Pettie [12] (with further improvements by Chan [4]) to be solvable in very slightly subquadratic time in the real-RAM model (also known as the *uniform* model). 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 [9, 11] for subsequent slight speedups). This was drastically further improved (still in the linear decision-tree model) to $O(n \log^2 n)$ time by Kane *et al.* [13]. In contrast, no subquadratic algorithm is known for the collinearity testing problem, neither in the standard real-RAM model nor in the decision-tree model; see [2, 3] for a discussion. Very recently, several input-restricted variants of collinearity testing were solved in subquadratic time, both in the decision-tree model and the RAM model [1–4]. However, a subquadratic solution to the unrestricted case still remains elusive.

In this paper we study collinearity testing with small working storage. This has been motivated by the work of Lincoln *et al.* [14], who have applied a simpler technique to reduce the working storage for the k-SUM problem (in which we want to determine whether a set N of real numbers contains a k-tuple of numbers that sum to zero), using a *self-reduction* mechanism. That is, given an algorithm for k-SUM, Lincoln *et al.* showed how to replace it by a different algorithm that uses, under suitable assumptions, smaller working storage and the same asymptotic running time. We obtain a similar self-reduction for the harder collinearity testing problem. Concretely, given a *base* algorithm for collinearity testing with running time $T(n)$ and working storage $S(n)$, we can construct another algorithm for the same problem with $O(r^2(n + T(n/r)))$ expected running time

*Tandon School of Engineering, New York University, Brooklyn NY, USA; boris.aronov@nyu.edu
†School of Computer Science, Bar Ilan University, Ramat Gan, Israel; ezraest@cs.biu.ac.il
‡School of Computer Science, Tel Aviv University, Tel Aviv, Israel; michas@tau.ac.il
§School of Computer Science, Bar Ilan University, Ramat Gan, Israel; guy.zigdon@live.biu.ac.il

and $O(r + n/r) + S(n/r)$ working storage, for any $1 \leq r \leq n$. The reduction is obtained by a cutting-based decomposition of the problem into subproblems of smaller size and applying the base algorithm to each of them. Since the subproblems are solved one by one, we only need working storage for just one subproblem, plus the bookkeeping cost of constructing and maintaining the decomposition. Controlling the bookkeeping costs is a significant feature of our self-reduction.

As a consequence, collinearity testing can be solved in $O(n^2)$ expected time and $O(n^{1/2})$ working storage. These bounds resemble the bounds obtained in [14] for the considerably simpler 3SUM and k-SUM problems.

Collinearity testing with small working storage

Our self-reduction mechanism uses several classical tools from computational geometry, such as cuttings, geometric duality, arrangements of lines, and the zone theorem; see, e.g., [8].

The input consists of a set P on n points in the plane. Denote by $T_0(n)$ and $S_0(n)$ the respective running time and working storage of the base algorithm A. Let L be the set of the lines dual to the points of P, obtained by a standard point-line duality transformation [8, Chapter 8]. From properties of point-line duality, a collinear triple of points of P in the primal plane corresponds to a *concurrent triple* of lines of L in the dual plane, and we consider in what follows the latter problem. Our algorithm is based on $(1/r)$-*cuttings* [5], and is conceptually simple.

Constructing the cutting with small working storage. Let $1 \leq r \leq n$ be an integer parameter to be fixed shortly. We construct a $(1/r)$-cutting $\Xi(L)$ for the lines in L. This is a decomposition of the plane into $O(r^2)$ pairwise openely disjoint triangles, so that each triangle is crossed by at most n/r lines of L. We present a concrete way of implementing (the randomized version of) Chazelle and Friedman's algorithm [5] for constructing such a cutting, which uses small working storage.

We take a random sample R of cr lines of L, for a sufficiently large constant $c > 0$, construct the arrangement $\mathcal{A}(R)$ of R (see below for our space-efficient way of constructing $\mathcal{A}(R)$), and triangulate its cells by connecting the leftmost vertex of each cell to all the other non-adjacent vertices (handling unbounded cells requires some easy and standard modifications). The overall number of these triangles is $O(r^2)$, and by the random sampling theory of Clarkson and Shor [6, 7], with constant probability, the interior of each such triangle Δ intersects at most $O(\frac{n}{r} \log r)$ lines of L. The set L_Δ of lines intersecting the interior of Δ is the *conflict list* of Δ. (Handling lines that meet Δ only at its boundary is simple and standard.) This completes the first stage of the construction of [5].

In the second stage we iterate over the triangular cells. For each cell Δ, we count, by brute force and with no extra working storage, how many lines of L it intersects. If this number is at most n/r, Δ is a cell of the final cutting. Otherwise, this number is tn/r, for some real number $t > 1$. We then sample $ct \log t$ lines[1] from the conflict list L_Δ of Δ, where c is the same constant defined above. We have already computed $|L_\Delta|$. If $|L_\Delta| > n/r$, we put $t = |L_\Delta|/(n/r)$. Next, for each line ℓ of L, we test whether it crosses Δ and, if so, add ℓ to a secondary sample T, with probability $(ct \log t)/|L_\Delta|$. We then construct $\mathcal{A}(T)$, clip it to within Δ, and triangulate each resulting cell, using the leftmost-vertex triangulation scheme as above. Again, the theory of Clarkson and Shor [6, 7] implies that, with constant probability, each resulting triangle is crossed by at most n/r lines of L. According to the analysis of [5], with constant probability, the overall number of the resulting triangles is still $O(r^2)$, and each is crossed by at most n/r lines of L. The resulting collection of these triangles is the desired $(1/r)$-cutting. (If the above properties do not hold, we abort the current construction

[1] In fact, this is the expected number of lines in the sample—see below for our sampling strategy.

and start a new one; in an expected constant number of trials we will obtain a cutting with the desired properties.)

This decomposition yields a divide-and-conquer mechanism for collinearity testing, where at each cell Δ we solve a subproblem of size n/r. That is, we apply the (dual) base algorithm A to the lines of L_Δ. If the algorithm A succeeds for any cell Δ, we report success. Otherwise, we report that P has no collinear triple. (Handling concurrent triples of lines meeting on $\partial\Delta$ is fairly standard.)

Ensuring small working storage. The standard implementation of this algorithm constructs the entire cutting, using two rounds of sampling as just described, and stores the conflict lists of the cells. We cannot afford to do any of these: The overall size of the conflict lists is $O(nr)$, which is way too large for our goal, but even the size of the cutting, which is $O(r^2)$, turns out to be too large. To overcome these issues, we do not construct and store explicitly the full cutting, but compute it in an incremental manner, so as to use less storage. To do so, consider the primary sampling stage, and let R be the random sample that the algorithm chooses; recall that $|R| = cr$. We construct $\mathcal{A}(R)$ using a line-sweep procedure. The working storage used by line sweeping consists of the storage used by the y-structure (the search tree along the sweepline λ) and the storage used by the event queue. The former storage, for cr lines, is clearly $O(r)$, and the latter storage is also $O(r)$ if we use the variant where the event queue only stores future intersections of pairs of lines that are currently consecutive along λ.

During the sweep we maintain all the faces of $\mathcal{A}(R)$ that λ crosses. Assuming general position of the lines of R, at each vertex v that the sweep encounters, one face of $\mathcal{A}(R)$ ends, one face begins, one face starts a new edge on its top boundary, and one face starts a new edge on its bottom boundary. (If we detect a vertex of degree ≥ 3, we terminate the algorithm right away, as such a concurrency implies collinearity among the points dual to the lines of R.) By the zone theorem [8], the total complexity of all the faces that λ crosses is $O(r)$. When a face φ of $\mathcal{A}(R)$ ends, we triangulate it by chords drawn from its leftmost vertex, and then process these triangles one at a time. (Special, albeit simple and standard, handling of the unbounded faces is needed.)

Let Δ be one of these triangles. We compute the size $|L_\Delta|$ of the conflict list L_Δ of Δ by brute force. If $|L_\Delta| \leq n/r$, we construct the list itself, and run the base algorithm A on (the set of points dual to) L_Δ. If $|L_\Delta| = tn/r$, for some $t > 1$, we construct a secondary sample R_Δ of $ct \log t$ lines of L_Δ, in expectation, using the mechanism described above. (A technical issue that arises here is that t could be large, that is, $t \gg r/\log r$, which would then make the working storage too large. However, as shown in [5], the probability of obtaining a cell with a large value of t is exponentially small in t, which allows us to assume that this does not happen, and that we always have $t = O(r/\log r)$, for otherwise we simply scrap the sample and try a new one, as above.) We then construct the arrangement $\mathcal{A}(R_\Delta)$ using the same line-sweeping machinery, but clip each triangle that we obtain to within Δ, and run the base algorithm on each of these triangles. This implies that the working storage is $O(r)$ (for maintaining the various samples, and for carrying out the sweep) plus $O(n/r)$ (for maintaining a single conflict list), plus $S_0(n/r)$, the working storage used by the base algorithm.

We repeat this procedure at each vertex that the primary or secondary sweeps encounter, and terminate as soon as the base algorithm detects a collinear triple. Otherwise, if we fail in all cells of the cutting, we report that P has no collinear triple. In the full version we provide the remaining details of the analysis, which show that the resulting self-reduction does have the performance bounds stated in the introduction. That is:

Theorem 2.1 *(a) Suppose that collinearity testing on a set of n points in the plane can be solved in time $T(n)$ and working storage $S(n)$. Then it can also be solved in expected time $O(r^2(n+T(n/r)))$ and working storage $O(r+n/r)+S(n/r)$, for any integer r with $1 \le r \le n$. (b) Consequently, collinearity testing on a set of n points in the plane can be solved in randomized expected time $O(n^2)$ and working storage $O(n^{1/2})$.*

Part (b) is obtained by two applications of part (a), both with $r = n^{1/2}$, where the base algorithm in the first round is the classical collinearity-testing algorithm, and in the second round it is the algorithm obtained in the first round. See the full version for details, as well as an extension of our algorithm to higher dimensions.

References

[1] B. Aronov, M. de Berg, J. Cardinal, E. Ezra, J. Iacono and M. Sharir, Subquadratic algorithms for some 3Sum-hard geometric problems in the algebraic decision tree model, *Comput. Geom.: Theory Appl.*, 109 (2023), appeared online 15.08.2022. Also in *Proc. 32nd Inter. Symp. Alg. Comput. (ISAAC)* (2021), 3:1–3:15.

[2] B. Aronov, E. Ezra and M. Sharir, Testing polynomials for vanishing on Cartesian products of planar point sets: Collinearity testing and related problems. *Discrete Comput. Geom.*, to appear. Also in *Proc. 36th Sympos. Computational Geometry* (2020), 8:1–8:14, and in arXiv:2003.09533.

[3] L. Barba, J. Cardinal, J. Iacono, S. Langerman, A. Ooms and N. Solomon, Subquadratic algorithms for algebraic generalizations of 3SUM, *Discrete Comput. Geom.* 61 (2019), 698–734.

[4] 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.

[5] B. Chazelle and J. Friedman, A deterministic view of random sampling and its use in geometry, *Combinatorica* 10 (1990), 229–249.

[6] K. L. Clarkson, New applications of random sampling in computational geometry, *Discrete Comput. Geom.* 2(2) (1987), 195–222.

[7] K. L. Clarkson and P. W. Shor, Applications of random sampling in computational geometry, II, *Discrete Comput. Geom.* 4 (1989), 387–421.

[8] M. de Berg, O. Cheong, M. van Kreveld and M. H. Overmars, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Santa Clara, CA, USA, 3rd edition, 2008.

[9] A. Freund, Improved subquadratic 3SUM, *Algorithmica* 77(2) (2017), 440–458.

[10] A. Gajentaan and M. H. Overmars, On a class of $O(n^2)$ problems in computational geometry, *Comput. Geom.* 5(3) (1995), 165–185.

[11] O. Gold and M. Sharir, Improved bounds for 3SUM, k-SUM, and linear degeneracy, *Proc. European Sympos. Algorithms* (2017), 42:1–42:13. Also in arXiv:1512.05279.

[12] A. Grønlund and S. Pettie, Threesomes, degenerates, and love triangles, *J. ACM* 65 (2018), 22:1–22:25.

[13] D. M. Kane, S. Lovett and S. Moran, Near-optimal linear decision trees for k-SUM and related problems, *J. ACM* 66(3) (2019), 16:1-16:18.

[14] A. Lincoln, V. Vassilevska Williams, J. R. Wang and R. R. Williams, Deterministic time-space trade-offs for k-SUM, *Proc. 43rd Int. Colloq. Automata, Languages and Programming (ICALP)* (2016), 58:1–58:14.

Acknowledgments. We would like to thank Tsvi Kopelowitz for helpful discussions. Work by Boris Aronov was partially supported by NSF grants CCF-15-40656 and CCF-20-08551 and by grant 2014/170 from the US-Israel Binational Science Foundation. Work by Esther Ezra was partially supported by Grants 824/17 and 800/22 from the Israel Science Foundation. Work by Micha Sharir was partially supported by ISF Grant 260/18.