UNIVERSITÉ LIBRE DE BRUXELLES

Faculté des Sciences
Département d'Informatique
Année Académique 2019/2020

Thèse présentée en vue de l'obtention du grade de Docteur en Sciences

Algorithms and Data Structures for 3SUM and Friends

by

Aurélien Ooms

Jury de thèse:

Prof. Stefan Langerman (Université libre de Bruxelles, Président)
Prof. Samuel Fiorini (Université libre de Bruxelles, Secrétaire)
Prof. Jean Cardinal (Université libre de Bruxelles, Promoteur)
Prof. John Iacono (Université libre de Bruxelles)
Prof. Moshe Lewenstein (Bar Ilan University)
Prof. Nabil Mustafa (ESIEE Paris)

To the Profane

Solving a problem consists in mapping a given data to an output in some automated way. In Computer Science, we solve problems with computers: we organize the data with data structures, and process those structures with algorithms. Solving problems consumes resources, for example, time. Problems are considered harder if they take more resources to solve.

We, computer scientists, are lazy. We do not want to solve the same problem over and over with ad-hoc solutions depending on the data. By automated we really mean that the solutions we design work for any data, and in particular, they work for any data size "n". Ideally, the resource consumption of our solutions should scale well with this parameter. This is why we study the behaviour of such solutions when applied to large[1] inputs.

Geometry is about space, points, curves, surfaces ... We study geometric problems: The given data is geometric, and the question about the data is geometric. The proposed algorithms and data structures therefore exploit geometry.

This thesis studies questions about geometric problems that are simple to formulate. However simple they are, nobody has managed to answer them. One of the problems asks to decide, given n points in the plane, whether three of them lie on a common line. Of course, we can solve this problem. The question of interest here is how fast it can be solved.

It is easy to solve this problem by testing all possible candidates but that is quite inefficient. There is a more involved but standard construction that takes significantly less time to execute but that is still considered inefficient. As of today, we do not know of any reason why this problem would be much harder than simply reading the data from begin to end.

Does this matter at all? It turns out simple problems appear to be bottlenecks of more complex ones: unless we manage to improve our under-

[1] *Gigantic multiplied by colossal multiplied by staggeringly huge is the sort of concept we're trying to get across here.* – Douglas Adams, The Restaurant at the End of the Universe.

standing of the simple ones, we are stuck with inefficient solutions for all of them.

In this thesis, we expose novel algorithms and data structures related to the problem stated above. Hopefully, this contribution will one day, directly or indirectly, help us solve some of the interesting open questions about this problem.

Token of Appreciation

TBD

Contents

To the Profane i

Token of Appreciation iii

Table of Contents vii

On Notation viii

0 In a Dozen Pages 1

I Without Proof 14

1 Models of Computation 15
1.1 Algorithms . . . 15
1.1.1 Random Access Machines . . . 16
1.1.2 Computation and Decision Trees . . . 16
1.2 Data Structures . . . 19
1.2.1 Encodings . . . 19

2 History 21
2.1 3SUM & k-SUM . . . 21
2.1.1 Variants . . . 22
2.1.2 Point Location . . . 23
2.1.3 Information Theoretic Lower Bound . . . 23
2.1.4 Higher Lower Bounds . . . 24
2.1.5 Uniform Algorithms . . . 25
2.1.6 Nonuniform Algorithms . . . 27
2.2 GPT & 3POL . . . 28
2.2.1 Variants . . . 29
2.2.2 Reductions from k-SUM . . . 29

2.2.3 Lower Bounds and Order Types . . . 31
2.2.4 Algorithms . . . 33
2.2.5 More on Order Types . . . 33
2.2.6 Encodings . . . 34
2.2.7 The Intermediate Problem . . . 36
2.2.8 Combinatorics . . . 37

3 Contributions **39**
3.1 Meiser Applied to k-SUM . . . 39
3.2 Grønlund and Pettie Applied to 3POL . . . 42
3.3 Slightly Subquadratic Encodings for GPT . . . 44
3.4 Better Encodings for 3SUM . . . 47

4 Developments **49**
4.1 Better Nonuniform Algorithms for k-SUM . . . 49
4.1.1 Using Vertical Decomposition . . . 49
4.1.2 Using Inference Dimension . . . 50
4.2 Timothy Chan Strikes Again . . . 51

5 Open Questions **52**
5.1 About Algorithms . . . 52
5.2 About Encodings . . . 54

II The Computational Geometer's Toolbox **55**

6 Arrangements **56**
6.1 Counting Cells . . . 56
6.2 Pseudolines . . . 57
6.3 Zone Theorem . . . 58
6.4 Cell Decomposition . . . 59
6.4.1 Bottom Vertex Triangulation . . . 59
6.4.2 Vertical Decomposition . . . 60

7 Chirotopes **62**
7.1 Duality . . . 62
7.2 Canonical Labelings . . . 64

8 Divide and Conquer **66**
8.1 Epsilon Nets and Cuttings . . . 66
8.1.1 Range Spaces . . . 66
8.1.2 VC-dimension . . . 67
8.1.3 Epsilon Nets . . . 68
8.1.4 Hyperplanes in Linear Dimension . . . 68
8.1.5 Cuttings . . . 69
8.1.6 Algebraic Range Spaces . . . 70
8.1.7 Derandomization . . . 70
8.2 Hierarchical Cuttings . . . 70

9 Existential Theory of the Reals **73**
9.1 Cylindrical Algebraic Decomposition . . . 73

III Algorithms **75**

A Solving k-SUM using Few Linear Queries **76**
A.1 Meiser Solves k-SUM . . . 77
A.1.1 Query Complexity . . . 77
A.1.2 Time Complexity . . . 81
A.1.3 Query Size . . . 85
A.2 Missing Details . . . 86
A.2.1 Keeping Queries Linear in Algorithm 1 . . . 86
A.2.2 Algebraic Computation Trees . . . 88
A.2.3 Uniform Random Sampling . . . 89
A.2.4 Proof of Lemma A.5 . . . 90

B Subquadratic Algorithms for Algebraic 3SUM **93**
B.1 First Subquadratic Algorithms for 3POL . . . 93
B.1.1 Nonuniform Algorithm for Explicit 3POL . . . 94
B.1.2 Uniform Algorithm for Explicit 3POL . . . 101
B.1.3 Nonuniform Algorithm for 3POL . . . 104
B.1.4 Uniform Algorithm for 3POL . . . 110
B.2 Subproblems . . . 115
B.2.1 Polynomial Batch Range Searching . . . 115
B.2.2 Polynomial Dominance Reporting . . . 118

B.3 Applications . . . 123
B.3.1 GPT for Points on Curves . . . 124
B.3.2 Incidences on Unit Circles . . . 126
B.3.3 Points Spanning Unit Triangles . . . 128

IV Data Structures **129**

C Subquadratic Encodings for Point Configurations **130**
C.1 Encoding Order Types via Hierarchical Cuttings . . . 131
C.2 Sublogarithmic Query Complexity . . . 142
C.3 Higher-Dimensional Encodings . . . 149

D Encoding 3SUM **155**
D.1 Representation by Numbers . . . 155
D.2 Space-Optimal Representation . . . 158
D.3 Subquadratic Space and Constant Query Time . . . 158

Bibliography **161**

List of Contributions and Open Questions **176**

On Notation

This short chapter hopes to lift any ambiguity in the notation used.

Big-Oh

To express the asymptotic behaviour of functions representing resource complexities (time and space) we use the standard *big-oh* notation (see for instance [55, Chapter 3]). For brevity, we add a few tweaks.

- The notation $\tilde{O}(\cdot)$ ignores polylogarithmic factors. For instance, we have $n^3 \log^2 n = \tilde{O}(n^3)$.
- The symbol ϵ (not to be mistaken for ε) denotes a positive real number that can be made as small as desired. Writing $T(n) = O(n^{12/7+\epsilon})$ means that for any fixed $\delta > 0$, $T(n) = O(n^{12/7+\delta})$, where the constant of proportionality may depend on δ. In particular, $n^2 \log^2 n = O(n^{2+\epsilon})$.
- The notation $O_{p_1,p_2,\ldots}(f(n))$ means that the constant of proportionality depends on the parameters p_i.
- When we write $O(f(n_1, n_2, \ldots))$ we assume that one of the variables n_i is the one going towards infinity in the big-oh definition, while the others are monotone functions of n_i (increasing or decreasing depending on the context).

Because this asymptotic notation takes little care of constant factors, logarithms are in base two unless otherwise indicated.

Sets

We denote by $\mathbb{R}^d$ the d-dimensional Euclidean space and try to be consistent with the set notation used to represent the subsets of this space.

- A point is indicated by a lowercase letter, for instance a vertex p.

- A set of points is indicated by an uppercase letter, for instance an hyperplane H, a cell C, or a simplex S.
- A set of sets of points is indicated by a calligraphic uppercase letter, for instance a set of hyperplanes $\mathcal{H}$, a net $\mathcal{N}$, or an arrangement $\mathcal{A}(\mathcal{H})$.

For finite sets, we sometimes use the short notation $[n] = \{ 1, 2, \ldots, n \}$ and describe a set of cardinality n as a n-set.

0

In a Dozen Pages

This thesis is a compilation of the contributions from four papers:

A "*Solving k-SUM Using Few Linear Queries*" with Jean Cardinal and John Iacono [41].

B "*Subquadratic Algorithms for Algebraic 3SUM*" with Luis Barba, Jean Cardinal, John Iacono, Stefan Langerman, and Noam Solomon [20].

C "*Subquadratic Encodings for Point Configurations*" with Jean Cardinal, Timothy Chan, John Iacono, and Stefan Langerman [38].

D "*Encoding 3SUM*" with Sergio Cabello, Jean Cardinal, John Iacono, Stefan Langerman, and Pat Morin [37].

Paper A was presented at CG:YRF 16 and ESA 16. Paper B was presented at EuroCG 17 and SoCG 17. It is published in DCG. Paper C was presented at FWCG 17, EuroCG 18, and SoCG 18. It is published in JoCG. Paper D was presented at EuroCG 19.

We begin this thesis with an overview of the studied topic. We explain the context in which those papers were written and expose the contributions contained in each of them.

Degeneracy Testing Problems

In this thesis, we study *Degeneracy Testing Problems*: an instance of size n of such a problem is a single point in high-dimensional euclidean space $q \in \mathbb{R}^{O(n)}$. Such an instance is called *general* if and only if it passes a series of algebraic tests (usually $n^{O(1)}$ of them). If it fails one of the tests, it is

called *degenerate*. Our goal is to determine how fast an instance can be classified as general or degenerate.

The terminology is justified because most instances are general: the set of degenerate instances is a zero-measure subset of the input space. It also makes sense to visualize the input space as the euclidean space: the algebraic tests naturally induce a partition of the input space into semialgebraic sets. Solving the problem therefore amounts to locate the input point q in this partition of space. Our goal is thus to determine how fast this input point can be located.

Degeneracy testing problems are easy decision problems because there are only a finite number of candidate tests to try. The ones we study can all be solved by brute-force in polynomial time because the number of tests is polynomial. We show how, in some cases, this naive approach is definitely subsumed by divide and conquer techniques exploiting the geometry of the setting.

GPT

Let us illustrate by giving a first example of a degeneracy testing problem. We begin with a definition:

Definition 1. A set of n points in $\mathbb{R}^d$ is in general position if and only if every $(d+1)$-subset spans the entire space. A point set that is not in general position is called *degenerate*.

The General Position Testing problem (GPT) asks to decide if a given set of n point is in general position. We can solve this problem by brute-force in $O(n^{d+1})$ time. We can do it an order of magnitude faster by constructing the dual arrangement of hyperplanes in $O(n^d)$ time. On the one hand, improving on this slightly better solution appears to be non-trivial: there exists a class of algorithms that cannot do better even though they exploit one of the core structures of the problem, the chirotope axioms. On the other hand, information theory only gives a decision tree lower bound of $\Omega(n \log n)$. A popular conjecture is that no $o(n^d)$ time real-RAM algorithm exists for this problem.

Nonuniform Algorithms

Another model of computation in which no $o(n^d)$ time algorithm for GPT is known is the algebraic computation tree model. In this model, an algorithm is a rooted tree whose internal nodes are either arithmetic operations or sign tests on real variables, and whose leaves are the result of the computation.

This model is more generous than the real-RAM model in the sense that all computations that can be carried out by only knowing the input size incur no cost. Because a computation tree has a fixed size, we need a different tree for each input size. Therefore, we say that this model is *nonuniform* since it allows to have a distinct algorithm for each input size.

This thesis considers both uniform algorithms in the real-RAM and word-RAM models of computation and nonuniform algorithms in the algebraic computation tree, algebraic decision tree, and linear decision tree models of computation. For a given task, the complexity of the nonuniform algorithm is less than the complexity of the uniform algorithm. While a nonuniform algorithm is rarely practical, designing those at least means making progress on the question of whether a sensible computation tree lower bound can be derived.

3SUM

The 3SUM problem also falls in the category of degeneracy testing problems. This problem asks to decide whether a given set of n numbers contains a triple whose sum is zero. We can solve this problem by brute-force in $O(n^3)$ time, and in $O(n^2)$ time with a slightly more clever algorithm.

However toyish 3SUM may look like, it is considered one of several key problems in P: many geometric problems reduce from it in subquadratic time. Hence, a conjectured quadratic lower bound on 3SUM implies a conditional lower bound on all those more practical problems.

Like for GPT, there exist lower bounds for 3SUM in restricted models of computation: 3SUM cannot be decided in $o(n^2)$ time if the only way we inspect the input is by testing for the sign of weighted sums of three input numbers.

Even before this lower bound was known, it was conjectured that a quadratic lower bound would hold in other models of computation like the

real-RAM model.

A first stab at the conjecture was made when it was proven that for integer input numbers, it is possible to beat the conjectured lower bound by a few logarithmic factors [19]. However, it remained open whether such improvements were possible for real inputs.

Eventually, in a breakthrough paper, Grønlund and Pettie gave a subquadratic uniform algorithm that shaves a root of a logarithmic factor from quadratic time [91]. Since then more roots of logarithmic factors have been shaved [79, 81]. To this day, it is still conjectured that, for all $\delta > 0$, 3SUM requires $\Omega(n^{2-\delta})$ time to solve in the real-RAM model.

k-SUM

The paper of Grønlund and Pettie also discusses the following generalization of the 3SUM problem: "For a fixed k, given a set of n real numbers, decide whether there exists a k-subset whose elements sum to zero." This problem is called the k-SUM problem.

Obviously, the 3SUM problem is the k-SUM problem where $k = 3$. Moreover, there is a simple reduction from k-SUM to 2SUM when k is even and to 3SUM when k is odd. Those reductions yield a $O(n^{\frac{k}{2}} \log n)$ time real-RAM algorithm for k even and a $O(n^{\frac{k+1}{2}})$ time real-RAM algorithm for k odd.

In their paper, in addition to the slightly subquadratic uniform algorithm for 3SUM, Grønlund and Pettie give a strongly subquadratic nonuniform algorithm for 3SUM. The algorithms runs in time $\tilde{O}(n^{3/2})$, and, because of the aforementioned reduction, immediately yields an improved $\tilde{O}(n^{\frac{k}{2}})$ nonuniform time complexity for k-SUM when k is odd.

As for uniform time complexity we do not know whether this nonuniform improvement can be transferred to the real-RAM model: we do not know of any real-RAM $o(n^{\frac{k+1}{2}})$ time algorithm for k-SUM when k is odd.

The k-SUM problem reduces to the following point location problem: "Given a input point $q \in \mathbb{R}^n$, locate q in the arrangement of $\binom{n}{k}$ hyperplanes of equation $x_{i_1} + x_{i_2} + \cdots + x_{i_k} = 0$." Applying the best nonuniform algorithms for point location in arrangements of hyperplanes by Meyer auf der Heide [114] and Meiser [113] yields linear decision trees of depth $n^{O(1)}$ for k-SUM, where the constant of proportionality in the big-oh does not depend on k.

Paper A Our first contribution is a finer analysis of Meiser's algorithm that shows that the depth of his decision tree when applied to k-SUM is actually $O(n^3 \log^2 n)$. On top of that, we show how to implement a variant of this decision tree in the real-RAM model so that its uniform running time is $n^{\frac{k}{2}+O(1)}$ while keeping the nonuniform running time unchanged. Note that a naive implementation of Meiser's algorithm has a uniform running time of $n^{k+O(1)}$.

The approach taken by Meiser's algorithm follows the prune and search method: Take a sample of the hyperplanes for which we do not yet know the relative location of the input point. Locate the input point with respect to this sample by brute-force. This amounts to identifying the cell of the sample's arrangement which contains the input point. Refine the location of the input point in the sample by partitioning the containing cell into low complexity subcells. Whenever this low complexity subcell is located completely on one side of an hyperplane, the input point is located on the same side, and so we can discard this hyperplane without a single query to the input point. Since some hyperplanes may intersect the low complexity subcell, rince and repeat.

The location refinement is necessary because this is the only way we can guarantee a bound on the number of hyperplanes we have to recurse on. Using the theory of ε-nets, we can show that, for a sample of reasonable size, any simplex that is not intersected by a sample hyperplane is not intersected by more than a constant fraction of the set of hyperplanes from which the sample is drawn (with high probability). We define the refined subcell of the algorithm presented above to be the simplex of the bottom-vertex triangulation of the sample's arrangement that contains the input point.

Sorting

Before continuing, let us go back to the origins of those different problems. The link between them will be made even clearer.

Sorting is one of the oldest and most relevant data management problems. It is the archetypal computation tree problem. Usually presented, sorting is about permutations in *arrays*, but we do not like that. We use a different abstraction:

Problem 1 (Sorting)**.** Given n numbers $q_1, q_2, \ldots, q_n \in \mathbb{R}$, for each pair $1 \leq$

$i < j \leq n$, decide whether $q_i < q_j$.

In other words, we see the sorting problem as an information retrieval problem: how many comparisons do we need to make in order to *know* the answer to all comparisons. The usual sorting algorithms rearrange an array of input numbers into sorted order. Another way to think about it is that they compute the permutation that sends the input array to a sorted version of itself. This permutation is a data structure such that given any index in the input array, we can query for the corresponding index in the sorted array called the *rank* of the element). To retrieve the result of a comparison of two elements of the input array we can compare their ranks in this permutation. The usual way of defining the sorting problem restricts the data structure that should be used to encode the $\binom{n}{2}$ comparisons. The way we model the sorting problem lifts this restriction because it does not ask to structure this information in a nice way.

The sorting problem also has its decision problem variant: Element Uniqueness.

Problem 2 (Element Uniqueness)**.** Given n numbers $q_1, q_2, \ldots, q_n \in \mathbb{R}$, decide whether there exists $1 \leq i < j \leq n$, such that $q_i = q_j$.

It is easy to see that Sorting amounts to locating the point q in the arrangement of hyperplanes of equations $x_i - x_j = 0$, and that Element Uniqueness reduces both to and from the 2SUM problem in linear time. Actually, under reasonable assumptions on the computation model, sorting and element uniqueness are the same problems: if all questions we ask about the input are linear in the input numbers, then proving that the input does not contain any duplicate entries requires to sort the input. The same statement carries over to the k-SUM with respect to its "sorting version": computing the sign of all $\binom{n}{k}$ sums of k input numbers. Therefore, in those models of computation, the sorting problem is equivalent to 2SUM. Because of this relationship between Sorting and the k-SUM problem, we see why the better understanding of k-SUM is a natural next move.

Among all the problems this thesis touches on, Sorting and Element Uniqueness are the simplest and best understood. We know $\Omega(n \log n)$ lower bounds in many nonuniform models of computation and we also know a long list of simple real-RAM algorithms for those problems whose running times

match those lower bounds. For all that matters here, those problems are solved.

A Zoo of Generalizations

Obviously, the k-SUM problem is not the only possible way to generalize Sorting.

Hopcroft's problem asks whether given n points and n hyperplanes in $\mathbb{R}^d$, one of the points lies on one of the hyperplanes. When $d = 1$, this problem is Element Uniqueness. Finding the location ("above"/"below") of each point with respect to each hyperplane generalizes Sorting.

Orthogonal Vectors (OV) is a problem that has gained popularity in the emerging field of fine grained complexity theory [154]. The problem is to decide whether a set of n d-dimensional vectors contains an orthogonal pair. It is easy to see that this problem is equivalent to Hopcroft's problem in $\mathbb{R}^{d-1}$.

The dominance reporting problem asks, given n points in $\mathbb{R}^d$, to report all pairs of points such that the first dominates the other in all dimensions. Once again, for $d = 1$, this problem is Sorting because it asks for the answer to all comparisons of the type $p_i \leq q_i$.

Sorting $X + Y$ is also a canonical problem in P: given two sets X and Y of n numbers each, sort the set $\{\, x + y \colon\ x \in X, y \in Y \,\}$. Sorting $X + Y$ reduces linearly to the sorting version of 4SUM because it asks for the sign of all comparisons of the type $x + y \leq x' + y'$.

We already saw that GPT and Sorting belong to the same family of high-dimensional point location problems. There is a good reason for that: when $d = 1$, GPT is Element Uniqueness. In one-dimensional space, GPT asks whether any two points are the same. Picturing this space as the (horizontal) real line, we see that the "sorting version" of GPT asks to compute for each pair of points which one is on the "left" of the other which simply amounts to Sorting the one-dimensional input points.

Intermediate Problems

A perspicacious reader may have frowned at the previous paragraphs wondering whether this embryo of classification brings more insight than confusion. "Sure!", one may say, "Many problems involve sorting, and

therefore, according to this dubious definition, are generalizations of it."

However, the author begs to differ.

As a first example, take one of the best known algorithms for 3SUM. This algorithm reduces an instance of 3SUM to a sorting phase and a searching phase [91]. The sorting phase consists in answering all questions of the type $x_i + x_{i'} \leq x_j + x_{j'}$ with some restriction on the indices. Well, it turns out that this phase is exactly an instance of the sorting version of the 2SUM problem where the input numbers are the differences $x_i - x_j$.[1] Moreover, to make the uniform algorithm practical, they implement the resolution of the 2SUM instance via dominance reporting. Note that this sorting problem is similar to the Sorting $X + Y$ problem where the answers to most questions are not cared for.

As a second example, we have the GPT problem. For this problem, the fact that the one-dimensional version is Sorting does not seem to give any insight on how to apprehend the more general problem, even in two dimensions. In order to make some progress in that direction, we need to capture more precisely to which family of problems GPT belongs. For that, we look at the algebra behind the problem.

GPT asks whether for any choice of $\binom{n}{d+1}$ input points p_i with coordinates $(p_{i,1}, p_{i,2}, \ldots, p_{i,d})$, the determinant

$$\det \begin{pmatrix} 1 & p_{1,1} & p_{1,2} & \cdots & p_{1,d} \\ 1 & p_{2,1} & p_{2,2} & \cdots & p_{2,d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & p_{d+1,1} & p_{d+1,2} & \cdots & p_{d+1,d} \end{pmatrix}$$

is zero. This determinant is a degree-d $(d^2 + d)$-variate polynomial. In particular, when $d = 2$, the determinant is a degree-2 6-variate polynomial. The GPT testing problem then amounts to testing whether the coordinates of any combination of the input points yields a root of that polynomial. We therefore consider the more general d-dimensional-k-POL (dD-k-POL) problem: Given a dk-variate constant degree polynomial F and a set S of n points in $\mathbb{R}^d$, decide whether $F(S^k)$ contains any zeroes. For instance, 2D-3POL generalizes GPT with $d = 2$ where the constant degree polynomial

[1] This observation generalizes to the k-SUM problem: any k-SUM instance can be reduced to a larger $(k-1)$-SUM instance followed by a searching phase.

is the 3×3 determinant mentioned above. Moreover, 1D-3POL generalizes 3SUM where the polynomial is simply the sum function. Equally interesting is the fact that 2D-2POL generalizes Hopcroft's problem with $d = 2$ where the polynomial is the dot product.

Paper B In paper B, we generalize Grønlund and Pettie's approach to solve 1D-3POL (or more simply, 3POL) in subquadratic time. Our approach is essentially the same in that it reduces 3POL to a sorting phase and a searching phase, the sorting phase being an instance of 2D-2POL.[2] Again, the implementation of the uniform algorithm solves this instance of 2D-2POL using dominance reporting (a generalization of it).

This result illustrates why a better understanding of the landscape of problems surrounding GPT helps to identify intermediate problems whose resolution marks progress towards the question of whether GPT admits subquadratic algorithms. Figure 1 depicts this landscape.

Encodings

Naive algorithms for Element Uniqueness, 3SUM, and k-SUM would search all possible combinations of 2, 3, or k input numbers for a match and reach horrible running times: respectively $O(n^2)$, $O(n^3)$, and $O(n^k)$.

The way better uniform algorithms for those problems work is by a combination of sorting and searching. The sorting part constructs a data structure and the searching part uses this data structure to answer the question at hand. The achieved running time is a balance between the cost of sorting and the cost of searching. This results in better running times than the naive "search only" solutions.

For instance, an efficient algorithm for the Element Uniqueness problem would first sort the input, then scan it for duplicates among adjacent numbers in this sorted order. The data structure that is constructed is the permutation we talked about earlier. This structure achieves a good compressing ratio by encoding the answer to all $O(n^2)$ pairwise comparisons of two input numbers in $O(n \log n)$ bits.

[2] Note that according to the implied definition, dD-k-SUM is equivalent to k-SUM. Therefore, the 2SUM instance in the sorting phase of their 3SUM algorithm is an instance of 2D-2SUM in disguise.

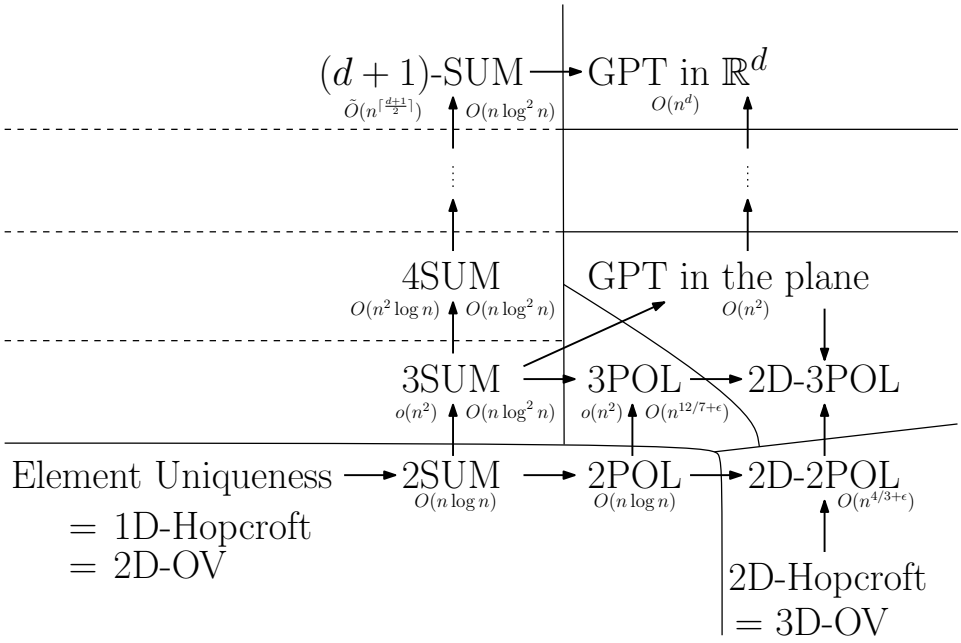


Figure 1. The decision problems surrounding GPT. Arrows indicate linear time reductions. $A = B$ indicates that A and B are the same problems. Known upper bounds are indicated next to the problem's name. When the uniform and nonuniform upper bounds differ, we place the best uniform upper bound on the right. Solid lines indicate apparent time-complexity differences between the best known nonuniform algorithms. Dotted lines indicate apparent time-complexity differences between the best known uniform algorithms.

For this reason Element Uniqueness is comparatively simpler than the 3SUM and k-SUM problems. For a nonuniform algorithm, constructing this structure is sufficient to solve the problem: Once the construction is over, we can discard the input because all the information we need is encoded in the data structure. Since the nonuniform models of computation we consider do not care about computations not involving the input, the searching part does not cost anything.

The case of 3SUM and k-SUM is more complex: The sorting phase only encodes a fraction of all possible queries. This leaves a significant amount of work for the searching phase. Therefore, both the sorting phase and the searching phase contribute to the nonuniform cost of those algorithms.

The case of GPT is also interesting. The naive algorithm queries $O(n^{d+1})$ input tuples while the better algorithm constructs the dual arrangement in $O(n^d)$ time. As in the case of Element Uniqueness, this dual arrangement is the data structure that encodes the answer to all $O(n^{d+1})$ queries while achieving a significant space gain.

While the goal sought in the design of algorithms is to find the best possible balance between those sorting and searching phases, a different question becomes evident: "What is the most resource efficient data structure encoding all the combinatorial information carried by the input?". For this question, resource efficiency can be measured in three ways: space requirements, construction time, and query time. If one only cares about space, a trivial answer points its head out: if there are only X combinatorial types then each type can be encoded with $\lceil \log X \rceil$ bits. However, this solution is unlikely to yield good construction time or good query time.

In the case of Element Uniqueness, this question has been answered long ago. Sorting the input in $O(n \log n)$ time will construct a permutation using $O(n \log n)$ bits that can be queried for any pairwise comparison in $O(1)$ time. However, the question is still widely open for 3SUM, k-SUM, and GPT.

Papers C & D In Paper C we design the first subquadratic space data structure for encoding the combinatorial type of a two-dimensional GPT instance. This data structure can be constructed in quadratic time and queries are answered in sublogarithmic time. Those results can be adapted to work for higher-dimensional GPT to yield sub-$O(n^d)$ space data structures with good construction and query times.

Since 3SUM reduces to GPT, the results of Paper C can be applied to encode the combinatorial type of 3SUM instances. However, since 3SUM is much better understood than GPT we should aim for better encodings. This is exactly what we do in Paper D. By filling the gaps in the partial data structure used in Grønlund and Pettie's algorithm [91], we design an encoding that uses $O(n^{3/2} \log n)$ bits, can be constructed in $O(n^2)$ time and answers queries in constant time.

The Secret Ingredient

What makes the success of our methods are now-standard geometric divide-and-conquer tools: ε-nets and cuttings.

The idea of an ε-net is simple, yet powerful. Given a sample of m hyperplanes in $\mathbb{R}^d$, construct a sample of those hyperplanes of size $f(d, \varepsilon)$ uniformly at random. Then, with high probability, any simplex that is not intersected by an hyperplane of the sample is intersected by at most εm hyperplanes of the original set. The sample is called a *net* because it does not let the fat simplices through: a simplex intersected by more than εm hyperplanes must be intersected by one of the sample. This yields an efficient point location tool: construct a sample, find the cell of its arrangement that contains the query point, find the simplex of its triangulation that contains the query point, and recurse on the fraction of hyperplanes that intersect it.

Triangulate the whole arrangement of the sample and you get a ε-cutting: a partition of space into simplices such that each simplex is intersected by at most εm hyperplanes. We use cuttings in applications where multiple queries are made on the same set of hyperplanes.

Wrapping it up

Since publication of those papers, a few developments have surfaced.

Ezra and Sharir [72] show how trading simplices of the bottom-vertex triangulation for prisms of the vertical decomposition in Meiser's algorithm yields a shallower decision tree of depth $O(n^2 \log n)$. Essentially, the improvement over our result in Paper A lies in the fact that, for vertical decomposition, the sample size can be taken to be an order of magnitude smaller.

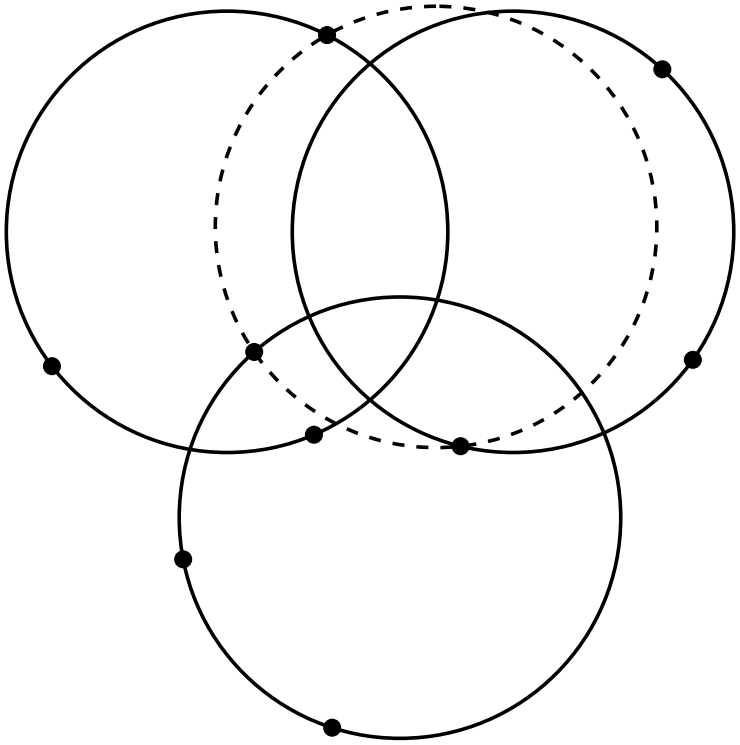
In a breakthrough paper, Kane, Lovett, and Moran [103] give a linear

decision tree of depth $O(n \log^2 n)$ for k-SUM, almost matching the $\Omega(n \log n)$ lower bound. This improves both on Paper A and Ezra and Sharir [72].

In [46], Chan shaves more logarithmic factors from the time complexity of uniform algorithms for 3SUM and 3POL. While we focused on applications that solve 3SUM-hard geometric problems with one-dimensional input in Paper B, he shows how the ideas that work for 3POL also work for some 3SUM-hard geometric problems with two-dimensional data.

I

Without Proof



1

Models of Computation

In Theoretical Computer Science, we simplify reality in order to make our job easier: we create models of what we think computers are and use mathematics to reason about them. This allows us, not only to show, but to *prove* that our algorithms are correct and efficient, according to those models. Hopefully we manage to capture the essence of what practical computation is about, making our lemmas and theorems relevant in practice.

We have to start this thesis somewhere, and, for our results to make sense, it is only natural that we begin by defining the models we use.

1.1 Algorithms

We study two classes of algorithms: uniform and nonuniform.

Uniform algorithms are considered practical. They can be implemented on a real computer because they take care of data management issues in an automated way.

Nonuniform algorithms ignore the data management aspect of practical computation. They do not give details on how to structure the intermediate results of their computation. Those results are assumed to be instantly accessible without any organization. Since we consider problem instances of arbitrarily large size, this is not possible in practice: processor units can only hold a fixed amount of data at any given time. The data has to be stored somewhere in some structured way.

The naming uniform simply means that the algorithm (its description) is the same for all input sizes. This is what is generally expected from algorithm design: to output a finite size description of an automated problem solving method that works for any instance size. Nonuniform algorithms on the

other hand are allowed to have distinct descriptions for each input size. Each nonuniform algorithm can be seen as an infinite sequence of uniform algorithms $A = A_1, A_2, \ldots$, each A_n hardcoding the data management part in its description. This description therefore is allowed to grow with n. Designing a nonuniform algorithm A amounts to describing a method that given n outputs A_n.

In this thesis, we study the uniform *random access machine* (RAM) models real-RAM and word-RAM, and the nonuniform decision tree models algebraic computation tree, algebraic decision tree, and linear decision tree. Those are described in the following paragraphs.

1.1.1 Random Access Machines

The real-RAM and word-RAM models have the RAM in common. They both assume a memory whose access cost is constant and a constant number of standard operations on the numbers they manipulate.

The real-RAM model is a classic of Computational Geometry. The input is assumed to consist of n real numbers and those numbers can be manipulated exactly at constant cost using arithmetic and comparison operators. This makes sense from the geometer's point of view since geometric data is best abstracted as real numbers. For data management purposes, the model also allows to manipulate $\log n$-bits integers with arithmetic, comparison, and bitwise operators but does not allow the conversion from a real number to an integer.

The word-RAM model models computers as we know them more closely. It considers an input consisting of n integers called words and allows a constant number of standard unary or binary operators to be executed in constant time. Those words have bitsize $w \geq \log n$.

1.1.2 Computation and Decision Trees

A decision tree is a constant-degree rooted tree whose internal nodes encode binary queries to an omniscient oracle and whose leaves encode the result of the computation. The complexity of the tree is defined to be the length of its longest root to leaf path called the *depth* of the tree.

A decision tree cannot inspect the input directly. To make progress, it inquires about the input through "yes/no" questions asked to the oracle.

The oracle answers those questions by "yes" or "no" honestly and accuratly.

An identification problem consists in, given an input of size n, distinguishing among $f(n)$ input classes, that is, to put the instance in the right box. A decision problem is an identification where $f(n) = 2$.

Of course, for every identification problem with $f(n)$ input classes, there is a decision tree that solves the problem by asking $O(\log f(n))$ "yes/no" questions of the form: "Is the input in the following range of input classes?" This is best possible: every query carries at most one additional bit of information, and there are at least $\lceil \log f(n) \rceil$ bits necessary to label each input class. This lower bound is called the Information Theoretic Lower Bound, or ITLB, and holds for any decision tree model.

The queries the above decision tree makes are too powerful to make the study of this kind of model interesting in general. Moreover, practically, those queries have no structure a priori and it is not at all obvious how they can be encoded. For a decision tree model to make sense, we have to restrict the kind of queries that can be made.

Linear Decision Trees

A first natural way to restrict the complexity of the queries of our decision tree model is to only allow linear queries on the input. This will be sufficient to solve problems such as Sorting, Element Uniqueness, 3SUM, k-SUM, and subset-sum.

In the *s-linear decision tree model*, queries consists in testing the sign of a linear function on at most s numbers $q_{i_1}, \ldots, q_{i_s}$ of the input $q_1, \ldots, q_n$. Such a query is called a s-linear query and can be written as

$$\alpha_1 q_{i_1} + \cdots + \alpha_s q_{i_s} \overset{?}{\leq} \alpha_0$$

Each question is defined to cost a single unit. All other operations can be carried out for free but may not examine the input vector q. We refer to n-linear decision trees simply as linear decision trees.

Note that this model generalizes the Comparison Tree Model commonly used for Sorting and Element Uniqueness, where queries have the form $q_i \overset{?}{\leq} q_j$.

Algebraic Decision Trees

The linear decision tree model is not powerful enough for more complicated problems. General Position Testing for instance involves discovering the sign of quadratic polynomials of the input, which is impossible in general with linear queries only.

The (bounded-degree) *algebraic decision tree (ADT)* [128, 146, 155] is an algebraic generalization of linear decision trees. An algebraic decision tree performs constant-cost branching operations that amount to test the sign of a constant-degree polynomial of the input numbers.

In this model, for an input $q_1, \ldots, q_n$, a query can be written as

$$p(q_1, \ldots, q_n) \overset{?}{\leq} 0,$$

where $p \in \mathbb{R}[x_1, \ldots, x_n]$ is a polynomial of constant degree. Again, operations not involving the input are free.

Algebraic Computation Trees

Computation trees differ from decision trees in that instead of exclusively asking "yes/no" questions, they perform arithmetic operations on variables whose results can be remembered and reused as operands, but only inspected using comparison queries. This allows, for instance, to make decision based on arbitrary polynomial expressions of the input with a sane way of counting the complexity of constructing such an expression. This would not be allowed in the algebraic decision tree model.

The internal nodes of an *algebraic computation trees* are labeled with arithmetic ($r \leftarrow o_1 \operatorname{op} o_2, \operatorname{op} \in \{+, -, \times, \div\}$) and branching ($z : 0$) operations. Any internal node labeled $r \leftarrow o_1 \operatorname{op} o_2$ has outdegree one and is such that either $o_k = q_i$ for some i or there exists an ancestor $o_k \leftarrow x \operatorname{op} y$ of this node, and any internal node labeled $z : 0$ has outdegree three and is such that either $z = q_i$ for some i or there exists an ancestor $z \leftarrow x \operatorname{op} y$ of this node. Similarly to the previously described decision tree models, leafs of the tree correspond to input classes.

Using this model can be interpreted as only counting the operations that involve the input, that is, members of the input or results of previous operations involving the input. All other arithmetic operations are for free.

1.2 Data Structures

Data Structures are ubiquitous in algorithmics. Given some data, we want to store it in computer memory in a way that 1) allows us to efficiently access the information we want to extract from this data and 2) does not require too much storage space. The term *structure* is by opposition to a randomly ordered stream of data.

Usually, a third important feature of those structures is that they can be also updated efficiently if the underlying data changes. We do not study or use that aspect in this thesis, instead we only require that those data structures can be constructed from scratch in an efficient way. Data structures allowing efficient updates are usually named *dynamic* while the ones we study are *static* data structures.

Those intuitive concepts are summarized as follows:

Preprocessing Time Given some data, the time it takes to construct the corresponding data structure in the given computation model.

Space The amount of space the data structure consumes in the given computation model. Can be expressed in bits, words, memory cells, ...

Query Time Given a data structure, the time consumed by a single query in the given computation model.

In §1.2.1 we give a precise definition of a particular type of data structure we study: encodings.

1.2.1 Encodings

A data structure is called *succinct* if its space usage is close to the ITLB. This concept was first introduced by Jacobson in his PhD thesis [101]. Since then, it has been extensively studied. Raman et al. studied the dynamic implementation of such data structures and gave the first rank-select succinct data structures [130, 131]. Those data structures are even efficient in practice as show by Vigna with their implementations [153].

In this thesis, we also try to get good bounds on the space used by the data structures we design. However, in most cases we are still very far from

the ITLB. We make all our data structure design problems fit in a single framework dubbed *instance encoding* so that their space and query time requirements can be easily compared.

Definition 2. For fixed k and given a function $f : [n]^k \to [O(1)]$, we define a ($S(n)$,$Q(n)$)-encoding of f to be a string of $S(n)$ bits such that, given this string and any $i \in [n]^k$, we can compute $f(i)$ in $Q(n)$ time in the word-RAM model with word size $w \geq \log n$.

2

History

In this chapter we define and recount the history of the different problems on which we made progress: 3SUM and k-SUM (§2.1), and GPT and 3POL (§2.2).

Note that we expose the state of knowledge as it was when we started this thesis back in 2015. For developments during the years 2016 to 2019 see §4.

2.1 3SUM & k-SUM

The 3SUM problem is defined as follows: given n distinct real numbers, decide whether any three of them sum to zero.

Problem 3 (3SUM)**.** Given n numbers $q_1 < q_2 < \ldots < q_n \in \mathbb{R}$, decide whether there exist $i < j < k \in [n]$ such that $q_i + q_j + q_k = 0$.

The seminal paper by Gajentaan and Overmars showed the crucial role of 3SUM in understanding the complexity of several problems in computational geometry [80].

A popular conjecture is that no $n^{2-\Omega(1)}$-time real-RAM algorithm solves 3SUM.

Conjecture 2.1. *There is no $n^{2-\Omega(1)}$-time real-RAM algorithm for 3SUM.*

This conjecture has been used to show conditional lower bounds for problems in P, notably in computational geometry with problems such as GeomBase, general position testing (GPT) [80] and Polygonal Containment [21], and more recently for string problems such as Local Alignment [3]

and Jumbled Indexing [18], as well as dynamic versions of graph problems [2, 105, 125], triangle enumeration and Set Disjointness [105].

The k-SUM problem is a straightforward generalization of the 3SUM problem: given n distinct real numbers, decide whether k of them sum to zero.

Problem 4 (k-SUM)**.** Given n numbers $q_1 < q_2 < \ldots < q_n \in \mathbb{R}$, decide whether there exist $i_1 < i_2 < \cdots < i_k \in [n]$ such that $\sum_{j=1}^{k} q_{i_j} = 0$.

It has been known for long that k-SUM is W[1]-hard and proved recently to be W[1]-complete [1]. The k-SUM problem is also a fixed-parameter version of the subset-sum problem, a standard *NP*-complete problem.

Similarly to 3SUM, the k-SUM problem has proved to be a computational bottleneck in high dimensional convex hull construction or general position testing [70].

For all those reasons, 3SUM and k-SUM are considered as key subjects of an emerging theory of complexity-within-P (or "fine-grained" complexity), on par with other problems such as all-pairs shortest paths, orthogonal vectors, boolean matrix multiplication, and conjectures such as the Strong Exponential Time Hypothesis [4, 42, 98, 117, 126].

2.1.1 Variants

For the sake of simplicity, we will sometimes consider the following definition of 3SUM:

Problem 5 (3SUM variant)**.** Given three sets A, B, and C of n real numbers, decide whether any triple of numbers in $A \times B \times C$ sums to zero.

A similar variant can be defined for k-SUM:

Problem 6 (k-SUM variant)**.** Given k sets S_i of n real numbers, decide whether there exists $(s_1, s_2, \ldots, s_k) \in S_1 \times S_2 \times \cdots \times S_k$ such that $\sum_{i=1}^{k} s_i = 0$.

The k-SUM problem can be further generalized to the linear degeneracy testing problem (k-LDT) where we allow coefficients other than zero and one.

Problem 7 (k-LDT)**.** Given n input numbers $q_1 < \cdots < q_n \in \mathbb{R}$ and constants $p_0, \ldots, p_n \in \mathbb{R}$ decide whether there exists $i_1, i_2, \ldots, i_k \in [n]$ such that $\sum_{j=1}^{k} p_j q_{i_j} = p_0$.

Our algorithms apply to this more general problem with only minor changes.

2.1.2 Point Location

The Point Location problem is a classic problem in Computational Geometry.

Problem 8 (Point Location)**.** Given a set $\mathcal{H}$ of m hyperplanes and a query point $q \in \mathbb{R}^d$, find the cell $C \in \mathcal{A}(\mathcal{H})$ such that $q \in C$.

Because the bounds in Theorem 6.1 are attained when $\mathcal{H}$ is in general position, there is a lower bound of $\Omega(d \log m)$ on the depth of decision trees solving this problem.

For our purpose, it is useful to see the k-SUM problem as a point location problem in $\mathbb{R}^n$ where the coordinates of q are the input numbers and where $\mathcal{H}$ is the set of all $\binom{n}{k}$ hyperplanes of equation $\sum_{j=1}^{k} x_{i_j} = 0$. Locating q in $\mathcal{A}(\mathcal{H})$ amounts to deciding in n-dimensional space, for each hyperplane $H \in \mathcal{H}$, whether q lies on, above, or below H. Since q lies on some H if and only if the k-SUM instance is degenerate, this constitutes a valid reduction. We emphasize that in this reduction the set of hyperplanes depends only on k and n and not on the actual input vector q.

Sorting can also be seen as a point location problem in $\mathbb{R}^n$. In this case $\mathcal{H}$ is the set of all $\binom{n}{2}$ hyperplanes of equation $x_i = x_j$. The arrangement $\mathcal{A}(\mathcal{H})$ has exactly $n!$ n-dimensional cell that correspond to the $n!$ permutations the input might have. Identifying the cell containing the input point reveals its permutation, hence solving the sorting problem.

See [144, Section 34.6] for more on point location in high dimension.

2.1.3 Information Theoretic Lower Bound

The ITLB for sorting is the logarithm of the number of possible permutations of the input, that is $\lceil \log n! \rceil = \Omega(n \log n)$, for *any* decision tree. This does not hold for element uniqueness: in an arbitrarily powerful decision tree model, using a single query, we can find out whether the input contains duplicates. This matches the useless lower bound of $\log 2 = 1$.

This observation holds for any decision problem: if the problem amounts to retrieve a single bit of information, the "yes/no" answer, we cannot we derive any useful lower bound for arbitrary decision tree models.

Restricting our model to only allow linear queries, we can derive a more useful lower bound for element uniqueness.

Lemma 2.1. *In the linear decision tree model, any algorithm solving the Element Uniqueness problem must sort its input (if it has no duplicates).*

Proof. Assume that the input q has no duplicates. Then, looking at q as a point in $\mathbb{R}^n$, q is contained in a d-cell of the arrangement of hyperplanes of equation $x_i = x_j$. Sorting the input amounts to identifying this cell. Assume we have solved the element uniqueness problem for this instance and further assume, by contradiction, that we have not identified the cell containing q. In the model we consider, each query/answer pair corresponds to a halfspace containing q. Because we have not identified the d-cell containing q, it must be that points from at least two d-cells of the arrangement are contained in the intersection of the halfspaces. Because halfspaces are convex sets, their intersection is convex too, and thus connected. In this connected set, a path from q to any point from the other d-cell must at some point intersect one of the hyperplanes $x_i = x_j$ that separates the two cells. An adversary can therefore update q to be this intersection point without changing any of the previous answers, even though this new point contains duplicates, a contradiction. □

Because of the decision tree lower bound on Sorting, we have the following:

Corollary 2.2. *Any linear decision tree solving the Element Uniqueness problem has depth* $\Omega(n \log n)$.

The same conclusion can be drawn for the 3SUM and k-SUM problems since Element Uniqueness reduces to them.

Corollary 2.3. *Any linear decision tree solving the k-SUM problem has depth* $\Omega(n \log n)$.

2.1.4 Higher Lower Bounds

In 1999, Erickson showed that we cannot solve k-SUM or k-LDT in subquadratic time in the k-linear decision tree model [69]:

Theorem 2.4 (Erickson [69])**.** *The depth of a k-linear decision tree solving the k-LDT problem is* $\Omega(n^{\lceil \frac{k}{2} \rceil})$.

The proof uses an adversary argument which can be explained geometrically. As we already observed, we can solve k-LDT problems by modeling them as point location problems in an arrangement of hyperplanes. Solving one such problem amounts to determining which cell of the arrangement contains the input point. The adversary argument of Erickson [69] is that there exists a cell having $\Omega(n^{\lceil \frac{k}{2} \rceil})$ boundary facets and in this model point location in such a cell requires testing each facet.

In 2005, Ailon and Chazelle slightly extended the range of query sizes for which a nontrivial lower bound could be established, elaborating on Erickson's technique [15]. They study s-linear decision trees to solve the k-SUM problem when $s > k$. In particular, they give an additional proof for the $\Omega(n^{\lceil \frac{k}{2} \rceil})$ lower bound of Erickson and generalize the lower bound for the s-linear decision tree model when $s > k$. Note that the exact lower bound given by Erickson for $s = k$ is $\Omega((nk^{-k})^{\lceil \frac{k}{2} \rceil})$ while the one given by Ailon and Chazelle is $\Omega((nk^{-3})^{\lceil \frac{k}{2} \rceil})$. Their result improves therefore the lower bound for $s = k$ when k is large. The lower bound they prove for $s > k$ is the following

Theorem 2.5 (Ailon and Chazelle [15])**.** *The depth of a s-linear decision tree solving the k-LDT problem is*

$$\Omega\big(nk^{-3}\big)^{(1-\epsilon_k)\left(\frac{2k-s}{2}\right)/\left\lceil \frac{s-k+1}{2} \right\rceil},$$

where $\epsilon_k > 0$ tends to 0 as $k \to \infty$.

This lower bound breaks down when $k = \Omega(n^{\frac{1}{3}})$ or $s \geq 2k$ and the cases where $k < 6$ give trivial lower bounds. For example, in the case of 3SUM with $s = 4$ we only get an $\Omega(n)$ lower bound.

2.1.5 Uniform Algorithms

On the real-RAM and word-RAM, it is easy to solve any k-SUM instance in time $O(n^{k/2} \log n)$ for k even and $O(n^{\frac{k+1}{2}})$ for k odd, and hence 3SUM in time $O(n^2)$. Those algorithms inspect the input using k-linear queries exclusively and the lower bounds of Erickson and Ailon and Chazelle in this model of computation essentially match their complexity.

Because fixing two of the numbers a and b in a triple only allows for one solution to the equation $a + b + x = 0$, an instance of 3SUM has at most

n^2 degenerate triples. An instance giving a matching lower bound is for example the set of n integers $\{ \frac{1-n}{2}, \frac{3-n}{2}, \ldots, \frac{n-1}{2} \}$ (for odd n) with $\frac{3}{4}n^2 + \frac{1}{4}$ degenerate triples. One might be tempted to think that the number of "solutions" to the problem would lower bound the complexity of algorithms for the decision version of the problem, as it is the case for this problem, and other problems, in restricted models of computation [68, 69]. This intuition is incorrect.

In 2008, Baran, Demaine, and Pătraşcu gave the first subquadratic algorithms for 3SUM [19]. They design subquadratic Las Vegas algorithms for 3SUM on integer and rational numbers in the circuit RAM, word RAM, external memory, and cache-oblivious models of computation. The idea of their approach is to exploit the parallelism of the models, using linear and universal hashing. However, since their algorithms do not handle real inputs, this did not settle the question of subquadratic algorithms in full generality.

In 2014, Grønlund and Pettie gave the first subquadratic algorithms for real-input 3SUM [91]. Using an old trick due to Fredman [77], they prove the existence of a linear decision tree solving the 3SUM problem using a strongly subquadratic number of linear queries. The classical quadratic algorithm for 3SUM uses 3-linear queries while the decision tree of Grønlund and Pettie uses 4-linear queries and requires $O(n^{\frac{3}{2}}\sqrt{\log n})$ of them.

Theorem 2.6 (Grønlund and Pettie [91])**.** *There is a 4-linear decision tree of depth $O(n^{3/2}\sqrt{\log n})$ for 3SUM.*

They show how to adapt this decision tree to run in subquadratic time in the real-RAM model. They design two subquadratic 3SUM real-RAM algorithms. A deterministic one running in $O(n^2/(\log n/\log\log n)^{\frac{2}{3}})$ time and a randomized one running in $O(n^2(\log\log n)^2/\log n)$ time with high probability.

In 2015, Chan and Lewenstein designed strongly subquadratic word-RAM algorithms for a high-dimensional variant of integer 3SUM with applications to jumbled indexing [47]. This result generalizes the folk wisdom that 3SUM on small integers can be solved in nearlinear time using FFT's.

The same year, Freund [79] and Gold and Sharir [81] improved on the results of Grønlund and Pettie [91]. Freund [79] gave a deterministic algorithm for 3SUM running in $O(n^2 \log\log n/\log n)$ time. Gold and Sharir [81] gave another deterministic algorithm for 3SUM with the same running time

and shaved off the $\sqrt{\log n}$ factor in the decision tree complexities of 3SUM and k-SUM given by Grønlund and Pettie.

Theorem 2.7 (Freund [79], Gold and Sharir [81])**.** *There is a* $O(\frac{n^2 \log\log n}{\log n})$*-time real-RAM algorithm for 3SUM.*

2.1.6 Nonuniform Algorithms

The nonuniform algorithm of Grønlund and Pettie for 3SUM in the 4-linear decision tree model generalizes to k-SUM [91]. In the $(2k-2)$-linear decision tree model, only $O(n^{\frac{k}{2}}\sqrt{\log n})$ queries are required for odd values of k. Putting this in perspective with the lower bounds of Erickson and Ailon and Chazelle, this indicates that increasing the size of the queries, thus making the model more powerful, does yield improvements on the depth of the minimal-height decision tree.

It has been well established that there exist nonuniform polynomial-time algorithms for the subset-sum and knapsack problems, even though those problems are NP-complete. In 1984, Meyer auf der Heide was the first to use a point location algorithm to solve the knapsack problem in the linear decision tree model in polynomial time [114]. He thereby answered a question raised by Dobkin and Lipton [58, 59], Yao [156] and others. However, if one uses this algorithm to locate a point in an arbitrary arrangement of hyperplanes the running time is increased by a factor linear in the greatest coefficient in the equations of all hyperplanes. In 1993, a second algorithm was described by Meiser [113], and is derived from a point location data structure for arrangements of hyperplanes using *bottom vertex triangulation* and *ε-nets* (see §6.4.1 and §8.1.3). The complexity of Meiser's point location algorithm is polynomial in the dimension, logarithmic in the number of hyperplanes and does not depend on the coefficients in the equations of the hyperplanes. A useful complete description of this algorithm is given by Bürgisser et al. [36, Section 3.4].

When applied to k-SUM, those algorithms can be cast as the construction of a n-linear decision tree, even though k is constant. They both yield $n^{O(1)}$-time nonuniform algorithms for k-SUM where the constant of proportionality does not depend on k, thus exhibiting the potential superiority of n-linear queries.

2.2 GPT & 3POL

In the plane, the General Position Testing problem (GPT) asks whether, given n points, three of them are collinear [**?**]. Because of some elementary algebra, the following form is equivalent:

Problem 9 (GPT in the plane)**.** Given n points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$, decide whether there exist $i < j < k \in [n]$ such that

$$\det \begin{pmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{pmatrix} = 0.$$

Note that this determinant is a degree-two polynomial in six variables

$$x_i y_j - y_i x_j - x_i y_k + y_i x_k + x_j y_k - y_j x_k.$$

In Computational Geometry this determinant is better known as the (counterclockwise) orientation $\nabla(i, j, k) \in \{-, 0, +\}$ of the points i, j, and k with coordinates (x_i, y_i), (x_j, y_j), and (x_k, y_k) and is defined as the sign of this determinant. It is called *orientation* or *sidedness* because for three given points this sign gives the orientation of the triangle those points define. It is often written as

$$\nabla(i, j, k) = \text{sign}((x_j - x_i)(y_k - y_i) - (y_j - y_i)(x_k - x_i)),$$

with only five subtractions and two multiplications.

The problem generalizes to higher dimension: given n points in $\mathbb{R}^d$, decide whether any $d+1$ of them lie on a common hyperplane. Again, using algebra:

Problem 10. Given n points $p_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,d}) \in \mathbb{R}^d$, decide whether there exist $i_1 < i_2 < \cdots < i_{d+1} \in [n]$ such that

$$\det \begin{pmatrix} 1 & p_{i_1,1} & p_{i_1,2} & \cdots & p_{i_1,d} \\ 1 & p_{i_2,1} & p_{i_2,2} & \cdots & p_{i_2,d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & p_{i_{d+1},1} & p_{i_{d+1},2} & \cdots & p_{i_{d+1},d} \end{pmatrix} = 0.$$

Again, this determinant is a degree d polynomial in $d^2 + d$ variables.

Later in this document we mention GPT without specifying the dimension. The problem is then assumed to be in the plane. When we consider a different parameterization we explicitly mention it.

2.2.1 Variants

Similarly to the 3SUM problem, we can study a variant of GPT where the tested triples come from different sets.

Problem 11 (GPT variant in the plane)**.** Given three sets A, B, and C of n points in $\mathbb{R}^2$, decide whether any triple of points in $A \times B \times C$ is collinear.

This variant is useful for a particular reduction from 3SUM. A similar variant can be defined for the d-dimensional problem.

2.2.2 Reductions from k-SUM

This section is divided into two parts. The first part exposes a folk reduction from k-SUM to GPT showing GPT in $\mathbb{R}^d$ is as hard as $(d+1)$-SUM. The second part exposes other interesting connections between k-SUM and GPT that can be drawn through Vandermonde determinants, yielding equivalent hardness proofs for GPT [70].

Folklore

Given an instance of the three-set variant of 3SUM, we can reduce it to an instance of GPT where the input sets are on three horizontal lines.

Observation 2.8. *Given three sets A, B, and C of n real numbers, let*

$$A' = \{\, a' = (a, 0) \colon\ a \in A \,\},$$
$$B' = \{\, b' = (b, 2) \colon\ b \in B \,\},$$
$$C' = \{\, c' = (-c/2, 1) \colon\ c \in C \,\}.$$

Then for any $a \in A$, $b \in B$, and $c \in C$, we have that $a + b + c = 0$ if and only if a', b', and c' are collinear.

Proof. The equation of a line defined by points $a' \in A'$ and $b' \in B'$ is

$$y = \frac{2}{b-a}x - \frac{2}{b-a}a$$
$$\iff\ x = a + \frac{y(b-a)}{2},$$

which simplifies to $x = \frac{a+b}{2}$ when $y = 1$. □

Therefore, solving GPT is at least as hard as solving 3SUM.

In the dual (see §7.1), this reduction is even easier to apprehend and trivially generalizes to d-dimensional GPT.

Observation 2.9. *Given k sets S_i of n real numbers, in $\mathbb{R}^{k-1}$ let S'_i, $1 \leq i \leq k-1$, be the set of n hyperplanes $x_i = s$ for $s \in S_i$ and let S'_k be the set of n hyperplanes $\sum_{i=1}^{k-1} x_i + s = 0$ for $s \in S_k$. Then for any $s_i \in S_i$, $1 \leq i \leq k$, we have that $\sum_{i=1}^{k} s_i = 0$ if and only if the corresponding hyperplanes intersect.*

Proof. Trivially. □

This proves that GPT in $\mathbb{R}^d$ is as hard as $(d+1)$-SUM.

Points on the Weird Moment Curve

When $d = 1$, GPT is exactly the element uniqueness problem.

Problem 12 (GPT on the real line)**.** Given n points $q_1, \ldots, q_n \in \mathbb{R}$, decide whether there exist $i < j \in [n]$ such that

$$\det \begin{pmatrix} 1 & q_i \\ 1 & q_j \end{pmatrix} = q_j - q_i = 0.$$

One seemingly stupid way to solve this problem is to consider the Vandermonde matrix

$$V(x) = \begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{n-1} \end{pmatrix},$$

and to compute its determinant

$$|V(q)| = \det \begin{pmatrix} 1 & q_1 & q_1^2 & \ldots & q_1^{n-1} \\ 1 & q_2 & q_2^2 & \ldots & q_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & q_n & q_n^2 & \ldots & q_n^{n-1} \end{pmatrix} = \prod_{1 \leq i < j \leq n} (q_j - q_i).$$

This can be done in $O(n \log n)$ ring multiplications by defining $p(x) = \prod_{i=0}^{n-1}(x - q_i)$ and computing

$$\begin{aligned}\det(V(q))^2 &= \operatorname{disc}_x(p(x)) \\ &= (-1)^{\binom{n}{2}} \operatorname{res}_x(p(x), \operatorname{diff}_x(p(x))) \\ &= (-1)^{\binom{n}{2}} \prod_{i=0}^{n-1} \operatorname{diff}_x(p(x))(q_i),\end{aligned}$$

using fast polynomial interpolation to first find the coefficients of $p(x)$ and then using fast polynomial evaluation to evaluate $\operatorname{diff}_x(p(x))$ at all q_i [6, 106, 107, 148].

Consider points on the moment curve $m_d(t) = (t, t^2, \ldots, t^d)$. Given a GPT instance consisting of points $m_d(q_i)$ on this curve the determinant in Problem 10 becomes

$$\det \begin{pmatrix} 1 & q_1^1 & q_1^2 & \cdots & q_1^d \\ 1 & q_2^1 & q_2^2 & \cdots & q_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & q_{d+1}^1 & q_{d+1}^2 & \cdots & q_{d+1}^d \end{pmatrix} = |V(q)|,$$

and solving the problem amounts to deciding Element Uniqueness.

Consider points on the *weird* moment curve $\omega_d(t) = (t, t^2, \ldots, t^{d-1}, t^{d+1})$. If our input only consists of points $\omega_d(q_i)$ on this curve then the determinant becomes

$$\det \begin{pmatrix} 1 & q_0^1 & q_0^2 & \cdots & q_0^{d-1} & q_0^{d+1} \\ 1 & q_1^1 & q_1^2 & \cdots & q_1^{d-1} & q_1^{d+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & q_d^1 & q_d^2 & \cdots & q_d^{d-1} & q_d^{d+1} \end{pmatrix} = |V(q)| \cdot \sum_{i=1}^{d+1} q_i,$$

which is the predicate we want to test for $(d+1)$-SUM [70] (modulo the $|V(q)|$ factor which is nonzero for distinct input numbers).

In the plane, this reduction from instances of 3SUM to points on the cubic curve $\omega_2(t) = (t, t^3)$ is also part of folklore.

2.2.3 Lower Bounds and Order Types

Even though 3SUM and k-SUM reduce to GPT, it does not necessarily mean that the lower bounds for those problems carry over. The lower bounds

in §2.1.3 are in the linear decision tree model and we cannot hope to solve GPT in those models because they do not allow us to answer quadratic polynomial queries on the input. That being said, solving an instance of GPT as in Observation 2.8 with orientation predicates is the same as trying to solve the original 3SUM instance with 3-linear queries. Hence, the quadratic lower bound of Erickson [69] carries over and GPT requires $\Omega(n^2)$ time in a model where one is only allowed orientation queries. More generally, GPT in $\mathbb{R}^d$ requires $\Omega(n^{\lceil \frac{d+1}{2} \rceil})$ time in a model where one is only allowed to query the orientation of d-simplexes of the input points.

However, neither those lower bounds, nor the weaker ones in the n-linear decision tree model, carry over in the more generous algebraic decision tree model and algebraic computation tree model. To get a superlinear lower bound for GPT in those models we need work a little bit harder.

Given an instance of GPT, its *order type* is defined to be the map that sends each triple of input points to its orientation.

Definition 3 (Order Type of a Point Set)**.** Given a set of n labeled points $P = \{p_1, p_2, \ldots, p_n\}$, we define the *order type* of P to be the function $\chi\colon [n]^3 \to \{-, 0, +\}\colon (a, b, c) \mapsto \nabla(p_a, p_b, p_c)$ that maps each triple of point labels to the orientation of the corresponding points, up to isomorphism.

Definition 4 (Realizable Order Type)**.** When considering an arbitrary function $f\colon [n]^3 \to \{-, 0, +\}$ we say that f is a realizable order type if there is a n-set $P \subset \mathbb{R}^2$ such that its order type is f.

If we consider the problem of identifying the order type of a set of points, we can achieve a lower bound that holds in any arbitrary decision tree model. It suffices for that to bound the number of possible order types.

Theorem 2.10 (Goodman and Pollack [87], Alon [16])**.** *There are* $2^{\Theta(n \log n)}$ *realizable order types.*

With the following corollary

Corollary 2.11. *A decision tree identifying the order type of a point sets has depth* $\Omega(n \log n)$.

The question now is whether we can do the same as in the Element Uniqueness versus Sorting case: show that the decision problem is as hard as the identification problem. Ben-Or [25] shows exactly that.

Theorem 2.12 (Ben-Or [25])**.** *Given a set $W \subset \mathbb{R}^d$ having N connected components, the depth of any algebraic computation tree or bounded-degree algebraic decision tree deciding whether an input point $q \in \mathbb{R}^d$ is in W is $\Omega(\log N - d)$.*

Because this Theorem applies to both W and its complement, we can replace N by $\max\{\#W, \#(\mathbb{R}^d - W)\}$ in the statement, where $\#X$ denote the number of connected components of the set X.

Corollary 2.13. *Any algebraic computation tree or bounded-degree algebraic decision tree that solves GPT has depth $\Omega(n \log n)$.*

Proof. Let $W \subset \mathbb{R}^{2n}$ be the set defined by the union of the $\binom{n}{3}$ algebraic varieties of equation

$$(x_j - x_i)(y_k - y_i) - (y_j - y_i)(x_k - x_i) = 0.$$

Solving GPT on input $q \in \mathbb{R}^{2n}$ amounts to deciding whether $q \in W$. The set $\mathbb{R}^{2n} - W$ has $2^{\Omega(n \log n)}$ connected components by Theorem 2.10. □

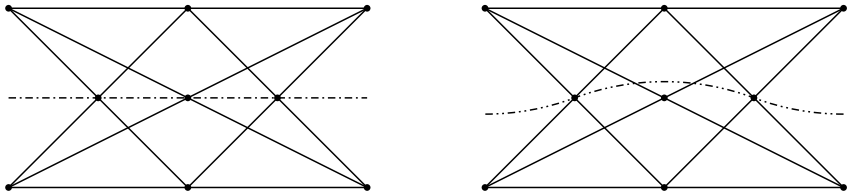
Note that both Theorem 2.10 and Theorem 2.12 rely on the Petrovskiĭ-Oleĭnik-Thom-Milnor (POTM) Theorem that bounds the number of connected components of algebraic varieties [23, 115, 151].

2.2.4 Algorithms

By brute-force, GPT can be solved in $O(n^3)$ time. Quadratic time can be achieved by constructing the dual arrangement [93, Theorem 24.4.1]. No subquadratic algorithm known, uniform or nonuniform. Essentially, nothing known.

2.2.5 More on Order Types

A great deal of the literature in computational geometry deals with the notion of order type [7–14, 16, 17, 28, 30, 31, 67, 71, 73–75, 82, 85–90, 94, 100, 104, 108, 112, 121, 138–140, 149]. The order type of a point set has been further abstracted into combinatorial objects known as (rank-three) *oriented matroids* [75]. The *chirotope axioms* define consistent systems of signs of triples [28]. From the topological representation theorem [30], all such *abstract* order types correspond to pseudoline arrangements, while,



a. Realizable order type.

b. Abstract order type which is not realizable.

Figure 2.1. Pappus's configuration.

from the standard projective duality, order types of point sets correspond to straight line arrangements. See Chapter 6 of The Handbook for more details [139].

When the order type of a pseudoline arrangement can be realized by an arrangement of straight lines, we call the pseudoline arrangement *stretchable*. As an example of a nonstretchable arrangement, Levi gives Pappus's configuration where eight triples of concurrent straight lines force a ninth, whereas the ninth triple cannot be enforced by pseudolines [108] (see Figure 2.1). Ringel shows how to convert the so-called "non-Pappus" arrangement of Figure 2.1 (b) to a simple arrangement while preserving nonstretchability [140]. All arrangements of eight or fewer pseudolines are stretchable [84], and the only nonstretchable simple arrangement of nine pseudolines is the one given by Ringel [138]. More information on pseudoline arrangements is available in Chapter 5 of The Handbook [83].

Figure 2.1 shows that not all pseudoline arrangements are stretchable. Indeed, most are not: there are $2^{\Theta(n^2)}$ abstract order types [73] and only $2^{\Theta(n \log n)}$ realizable order types (Theorem 2.10).

Theorem 2.14 (Felsner [73])**.** *There are* $2^{\Theta(n^2)}$ *abstract order types.*

This discrepancy stems from the algebraic nature of realizable order types, as illustrated by the main tool used in the upper bound proofs (the POTM Theorem [23, 115, 151]).

2.2.6 Encodings

At SoCG'86, Bernard Chazelle asked [89]:

> "How many bits does it take to know an order type?"

This question is of importance in Computational Geometry for the following two reasons: First, in many algorithms dealing with sets of points in the plane, the only relevant information carried by the input is the combinatorial configuration of the points given by the orientation of each triple of points in the set (clockwise, counterclockwise, or collinear) [61, 67, 104]. Second, computers as we know them can only handle numbers with finite description and we cannot assume that they can handle arbitrary real numbers without some sort of encoding. The study of *robust* algorithms is focused on ensuring the correct solution of problems on finite precision machines. Chapter 41 of The Handbook of Discrete and Computational Geometry is dedicated to this issue [158].

Regarding encoding the function χ of Definition 3 (as in Definition 2), Theorem 2.14 together with information theory imply that $\Theta(n^2)$ bits are necessary and sufficient for abstract order types whereas $\Theta(n \log n)$ bits are necessary and sufficient for realizable order types. Optimal encodings matching those bounds can be produced by a simple enumeration algorithm. However, it is unclear how the original information can be efficiently reconstructed from those encodings. On the other hand, storing all $\binom{n}{3}$ orientations in a lookup table to render this information accessible seems wasteful.

Another obvious idea for encoding the order type of a point set is to store the coordinates of the points, and answer orientation queries by computing the corresponding determinant. While this should work in many practical settings, it cannot work for all point sets. Perles's configuration shows that some configuration of points, containing collinear triples, forces at least one coordinate to be irrational [92] (see Figure 2.2). It is easy to see that order types of points in general position can always be represented by rational (or integer) coordinates. However, it is well known that some configurations require doubly exponential coordinates, hence coordinates with exponential bitsizes if represented in the binary numeral system [90].

Goodman and Pollack defined λ-matrices which can encode abstract order types using $O(n^2 \log n)$ bits [85] and can be constructed in $O(n^2)$ time [63]. They asked if the space requirements could be moved closer to the information-theoretic lower bounds. Everett, Hurtado, and Noy complained that this encoding does not allow a fast decoding for individual triples [71]. Knuth and Streinu independently gave new encodings of size $O(n^2 \log n)$

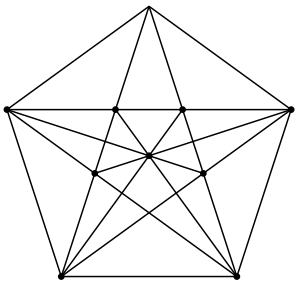


Figure 2.2. Perles's configuration.

that allow orientation queries in constant time [104, 149].[1] Felsner and Valtr showed how to encode abstract order types optimally in $O(n^2)$ bits via the wiring diagram of their corresponding allowable sequence [73, 74] (as defined in [82]). Aloupis et al. gave an encoding of size $O(n^2)$ that can be computed in $O(n^2)$ time and that can be used to test for the isomorphism of two distinct point sets in the same amount of time [17].

2.2.7 The Intermediate Problem

On par with GPT we consider an algebraic generalization of the 3SUM problem: we replace the sum function by a constant-degree polynomial in three variables $F \in \mathbb{R}[x, y, z]$ and ask to determine whether there exists a *degenerate* triple (a, b, c) of input numbers such that $F(a, b, c) = 0$. We call this problem the *3POL problem*.

Problem 13 (3POL)**.** Let $F \in \mathbb{R}[x, y, z]$ be a trivariate polynomial of constant degree, given three sets A, B, and C, each containing n real numbers, decide whether there exist $a \in A$, $b \in B$, and $c \in C$ such that $F(a, b, c) = 0$.

In addition to generalizing 3SUM, this problem can model particular instances of GPT: consider points in the plane that are constrained to lie on three parameterized polynomial curves of constant degree, then an algorithm for 3POL can determine whether any three of those points are collinear. Note that this generalizes the reductions from 3SUM found in §2.2.2, where the input points either lie on three horizontal lines or on a single cubic curve.

[1] We attract the attention of the reader on the fact that we express size in bits. Other authors, [71] and [149] in particular, express size in number of words, which is off by at least a logarithmic factor.

In fact, this problem is not new: it has already been studied before from a combinatorics point of view as exposed in the next section.

2.2.8 Combinatorics

In a series of results spanning fifteen years, Elekes and Rónyai [65], Elekes and Szabó [66], Raz, Sharir and Solymosi [136], and Raz, Sharir and de Zeeuw [133] give upper bounds on the number of degenerate triples for the 3POL problem. For the particular case $F(x, y, z) = f(x, y) - z$ where $f \in \mathbb{R}[x, y]$ is a constant-degree bivariate polynomial, Elekes and Rónyai [65] show that the number of degenerate triples is $o(n^2)$ unless f is *special*. Special for f means that f has one of the two special forms

$$f(u, v) = h(\varphi(u) + \psi(v)) \qquad \text{or} \qquad f(u, v) = h(\varphi(u) \cdot \psi(v)),$$

where h, φ, ψ are univariate polynomials of constant degree. It must be noted that the 3SUM problem falls in the special category since, in that case, f is the sum function. Elekes and Szabó [66] later generalized this result to a broader range of functions F using a wider definition of specialness. Raz, Sharir and Solymosi [136] and Raz, Sharir and de Zeeuw [133] improved both bounds to $O(n^{11/6})$. They translated the problem into an incidence problem between points and constant-degree algebraic curves. Then, they showed that unless f (or F) is special, these curves have low multiplicities. Finally, they applied a theorem due to Pach and Sharir [122] bounding the number of incidences between the points and the curves.

Theorem 2.15 (Raz, Sharir and de Zeeuw [133])**.** *Let A, B, C be n-sets of real numbers and $F \in \mathbb{R}[x, y, z]$ be a polynomial of constant degree, then*

$$|Z(F) \cap (A \times B \times C)| = O(n^{11/6}),$$

unless F has some group related form.[2]

Raz, Sharir and de Zeeuw [133] also look at the number of degenerate triples for the General Position Testing problem when the input is restricted to points lying on a constant number of constant-degree algebraic curves.

[2]Because our results do not depend on the meaning of *group related form*, we do not bother defining it here. We refer the reader to Raz, Sharir and de Zeeuw [133] for the exact definition.

Theorem 2.16 (Raz, Sharir and de Zeeuw [133])**.** *Let C_1, C_2, C_3 be three (not necessarily distinct) irreducible algebraic curves of degree at most d in $\mathbb{C}^2$, and let $S_1 \subset C_1, S_2 \subset C_2, S_3 \subset C_3$ be finite subsets. Then the number of proper collinear triples in $S_1 \times S_2 \times S_3$ is*

$$O_d(|S_1|^{1/2}|S_2|^{2/3}|S_3|^{2/3} + |S_1|^{1/2}(|S_1|^{1/2} + |S_2| + |S_3|)),$$

unless $C_1 \cup C_2 \cup C_3$ is a line or a cubic curve.

Nassajian Mojarrad, Pham, Valculescu and de Zeeuw [120] and Raz, Sharir and de Zeeuw [134] proved bounds for versions of the problem where F is a 4-variate polynomial.

3

Contributions

In this chapter we expose our contributions. **All sections have tables that list past results and our contributions together with a reference. When the result is a contribution from this thesis, this reference has the format *Section(Contribution)*. The section reference points to the details and proofs to attain the result and the contribution reference points to the theorem statement.**

3.1 Meiser Applied to k-SUM

In Paper A we focus on the computational complexity of k-SUM. We recall its definition.

Problem 4 (k-SUM)**.** Given n numbers $q_1 < q_2 < \ldots < q_n \in \mathbb{R}$, decide whether there exist $i_1 < i_2 < \cdots < i_k \in [n]$ such that $\sum_{j=1}^{k} q_{i_j} = 0$.

In §A.1.1, we show the existence of an n-linear decision tree of depth $\tilde{O}(n^3)$ for k-SUM using a careful implementation of Meiser's algorithm [113]. Although the high-level algorithm itself is not new, we refine the implementation and analysis for the k-SUM problem.[1] Meiser presented his algorithm as a general method of d-dimensional point location in the arrangement of m hyperplanes that yielded a $\tilde{O}(d^4 \log m)$-depth algebraic computation tree; when viewing the k-SUM problem as a point location problem, with $d = n$ and $m = O(n^k)$, Meiser's algorithm can be applied to this problem, yielding a $\tilde{O}(n^4)$-depth algebraic computation tree. We show that while the original

[1] After submitting our results, we learned from a personal communication with Hervé Fournier that a similar analysis for arbitrary hyperplanes appears in his PhD thesis [76] (in French).

algorithm was cast as a nonuniform polynomial-time algorithm, it can be implemented in the linear decision tree model with an $\tilde{O}(n^3)$ upper bound. Moreover, this result implies the same improved upper bound on the depth of algebraic computation trees for the k-SUM problem, as shown in §A.2.2.

There are two subtleties to this result. The first is inherent to the chosen complexity model: even if the number of queries to the input is small (in particular, the degree of the polynomial complexity is invariant on k), the time required to *determine which queries should be performed* may be arbitrary. In a naïve analysis, we show it can be trivially bounded by $\tilde{O}(n^{k+2})$. In §A.1.2 we improve on this and present an algorithm to choose which decisions to perform whereby the running time can be reduced to $\tilde{O}(n^{\frac{k}{2}+8})$. Hence, we obtain an $\tilde{O}(n^{\frac{k}{2}+8})$ time randomized algorithm in the RAM model expected to perform $\tilde{O}(n^3)$ linear queries on the input.

Contribution 1. *There exist linear decision trees of depth $O(n^3 \log^2 n)$ solving the k-SUM and the k-LDT problems. Furthermore, for the k-SUM problem there exists an $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$ Las Vegas algorithm in the word-RAM model expected to perform $O(n^3 \log^2 n)$ linear queries on the input. This algorithm also solves the k-LDT problem within the same time bounds provided the coefficients of the underlying linear equation are constant rational numbers.*

For those analyses, we consider algorithms in the standard word-RAM model with $\Theta(\log n)$-size words, but in which the input $q \in \mathbb{R}^n$ is accessible *only* via a linear query oracle. Hence we are not allowed to manipulate the input numbers directly. The complexity is measured in two ways: by counting the total number of queries, just as in the linear decision tree model, and by measuring the overall running time, taking into account the time required to determine the sequence of linear queries. This two-track computation model, in which the running time is distinguished from the query complexity, is commonly used in results on comparison-based sorting problems where analyses of both runtime and comparisons are of interest [39, 40, 147].

The second issue we address is that the linear queries in the above algorithm may have size n, that is, they may use all the components of the input. The lower bound of Erickson shows that if the queries are of minimal size, the number of queries cannot be a polynomial independent of k such as what we obtain, so non-minimal query size is clearly essential to a drastic

Table 3.1. Complexities of past and new algorithms for the k-SUM problem. For our new algorithms, the number of blocks is a parameter that allows us to change the query size (see §A.1.3). The origin of the constant in the exponent of the time complexity is due to Lemma A.5. We conjecture it can be reduced, though substantial changes in the analysis will likely be needed to do so.

Reference	Blocks	Query Size	Queries	Time
Folklore	–	k	$\tilde{O}(n^{\lceil \frac{k}{2} \rceil})$	$\tilde{O}(n^{\lceil \frac{k}{2} \rceil})$
Meiser [113]	–	n	$\tilde{O}(n^3)$ §A.1.1	$\tilde{O}(n^{k+2})$ §A.1.1
[81, 91]	–	$2k-2$	$\tilde{O}(n^{\frac{k}{2}})$	$\tilde{O}(n^{\frac{k}{2}})$
· (1)	1	n	$\tilde{O}(n^3)$ §A.1.1	$\tilde{O}(n^{\lceil \frac{k}{2}+8 \rceil})$ §A.1.2
§A.1.3 (2)	b	$k\lceil \frac{n}{b} \rceil$	$\tilde{O}(b^{k-4}n^3)$	$\tilde{O}(b^{\lfloor \frac{k}{2}-9 \rfloor}n^{\lceil \frac{k}{2}+8 \rceil})$
§A.1.3 (A.7)	$b = n^{\Theta(1)}$	$o(n)$	$\tilde{O}(n^3)$	$\tilde{O}(n^{\lceil \frac{k}{2}+8 \rceil})$
§A.1.3 (A.8)	$b = \Theta(n^\alpha)$	$O(n^{1-\alpha})$	$\tilde{O}(n^{3+(k-4)\alpha})$	$\tilde{O}(n^{(1+\alpha)\frac{k}{2}+8.5})$

reduction in the number of queries needed. This gives rise to the natural question as to what is the relation between query size and number of queries. In particular, one natural question is whether queries of size less than n would still allow the problem to be solved using a number of queries that is a polynomial independent of k. We show that this is possible; in §A.1.3, using a blocking scheme, we introduce a family of algorithms exhibiting an explicit tradeoff between the number of queries and their size.

Contribution 2. *For any integer $b > 0$, there exists a $k\lceil \frac{n}{b} \rceil$-linear decision tree of depth $\tilde{O}(b^{k-4}n^3)$ solving the k-SUM problem. Moreover, this decision tree can be implemented as an $\tilde{O}(b^{\lfloor \frac{k}{2} \rfloor - 9}n^{\lceil \frac{k}{2} \rceil + 8})$ Las Vegas word-RAM algorithm.*

We expose two corollaries of this contribution: We show that we can restrict our algorithms to use $o(n)$-linear queries while keeping the same complexity bounds, up to polylogarithmic factors. We also give a range of tradeoffs for $O(n^{1-\alpha})$-linear decision trees. Although the proposed algorithms still involve nonconstant-size queries, this is the first time such tradeoffs are explicitly tackled. Table 3.1 summarizes our results.

3.2 Grønlund and Pettie Applied to 3POL

In Paper B we focus on the computational complexity of 3POL. Since 3POL contains 3SUM, an interesting question is whether a generalization of Grønlund and Pettie's 3SUM algorithm exists for 3POL. If this is true, then we might wonder whether we can "beat" the $O(n^{11/6}) = O(n^{1.833\ldots})$ combinatorial bound of Raz, Sharir and de Zeeuw [133] with nonuniform algorithms (see §2.2.8). We give a positive answer to both questions: we design a uniform $O(n^2(\log\log n)^{3/2}/(\log n)^{1/2})$-time algorithm and a nonuniform $O(n^{12/7+\varepsilon}) = O(n^{1.7143})$-time algorithm for 3POL. To prove our uniform result, we present a fast algorithm for the Polynomial Dominance Reporting (PDR) problem, a far reaching generalization of the Dominance Reporting problem. As the algorithm for Dominance Reporting and its analysis by Chan [44] is used in fast algorithms for all-pairs shortest paths, (min,+)-convolutions, and 3SUM, we expect this new algorithm will have more applications.

To make the exposition of our results simpler, we study two different variants of the 3POL problem. The first variant is the 3POL problem as defined in the previous chapter (§2.2.7). We recall its definition here.

Problem 13 (3POL)**.** Let $F \in \mathbb{R}[x, y, z]$ be a trivariate polynomial of constant degree, given three sets A, B, and C, each containing n real numbers, decide whether there exist $a \in A$, $b \in B$, and $c \in C$ such that $F(a, b, c) = 0$.

The second variant is a special case of the 3POL problem where we restrict the trivariate polynomial F to have the form $F(a, b, c) = f(a, b) - c$. We call it the explicit 3POL problem because the dependency on the third variable is explicitly given:

Problem 14 (explicit 3POL)**.** Let $f \in \mathbb{R}[x, y]$ be a bivariate polynomial of constant degree, given three sets A, B, and C, each containing n real numbers, decide whether there exist $a \in A$, $b \in B$, and $c \in C$ such that $c = f(a, b)$.

Similarly to Grønlund and Pettie [91], we consider both nonuniform and uniform models of computation. For the nonuniform model, Grønlund and Pettie consider linear decision trees. Because linear decision trees are not powerful enough to even solve the problems we are looking at, in this contribution, we consider *bounded-degree algebraic decision trees*

Table 3.2. Algorithmic results for 3POL and its explicit variant.

Reference	Model	Complexity
§B.1.1 (3), §B.1.3 (5)	ADT	$O(n^{12/7+\epsilon})$
§B.1.2(4), §B.1.4(6)	real-RAM	$O(n^2(\log\log n)^{3/2}/(\log n)^{1/2})$

(ADT), an algebraic generalization of linear decision trees (see §1.1.2). For the uniform model we consider the real-RAM model with only the four arithmetic operators (see §1.1.1), the same model as in [91].

In §B.1.1 and §B.1.2, we first design uniform and nonuniform algorithms for explicit 3POL.

Contribution 3. *Explicit 3POL can be solved in $O(n^{12/7+\varepsilon})$ time in the bounded-degree algebraic decision tree model.*

Contribution 4. *Explicit 3POL can be solved in $O(\frac{n^2(\log\log n)^{3/2}}{(\log n)^{1/2}})$ time in the real-RAM model.*

In §B.1.3 and §B.1.4, we show that those algorithms can be adapted to solve the more general 3POL problem.

Contribution 5. *3POL can be solved in $O(n^{12/7+\varepsilon})$ time in the bounded-degree algebraic decision tree model.*

Contribution 6. *3POL can be solved in $O(\frac{n^2(\log\log n)^{3/2}}{(\log n)^{1/2}})$ time in the real-RAM model.*

Table 3.2 gives a summary of our results.

Applications Our results can be applied to many algebraic degeneracy testing problems, such as GPT, a well known 3SUM-hard problem for which no subquadratic algorithm is known (see §2.2). Raz, Sharir and de Zeeuw results on the 3POL problem [133] can be applied to obtain a combinatorial bound of $O(n^{11/6})$ on the number of collinear triples when the input points are known to be lying on a constant number of polynomial curves, provided those curves are neither lines nor cubic curves. A corollary of our first result is that GPT where the input points are constrained to lie on $o((\log n)^{1/6}/(\log\log n)^{1/2})$ constant-degree polynomial curves (including

lines and cubic curves) admits a subquadratic real-RAM algorithm and a strongly subquadratic bounded-degree algebraic decision tree. Interestingly, both reductions from 3SUM to GPT on 3 lines (map a to $(a, 0)$, b to $(b, 2)$, and c to $(-\frac{c}{2}, 1)$) and from 3SUM to GPT on a cubic curve (map a to (a^3, a), b to (b^3, b), and c to (c^3, c)) construct such special instances of GPT (see §2.2.2 for details on those reductions). This constitutes the first step towards closing the major open question of whether GPT can be solved in subquadratic time. To further convince the reader of the expressive power of the 3POL problem, we also give reductions from the problem of counting triples of points spanning unit circles (§B.3.2), from the problem of counting triples of points spanning unit area triangles (§B.3.3), and from the problem of counting collinear triples in any dimension (§B.3.1).

A Remark The algorithms we present manipulate polynomial expressions. In computational geometry, it is customary to assume the real-RAM model can be extended to allow the computation of roots of constant degree polynomials. We distance ourselves from this practice and take particular care of using the real-RAM model and the bounded-degree algebraic decision tree model with only the four arithmetic operators (see Chapter 9 on the Existential Theory of the Reals).

3.3 Slightly Subquadratic Encodings for GPT

In Paper C, we are interested in *compact* encodings for order types: we wish to design data structures using as few bits as possible that can be used to quickly answer orientation queries of a given abstract or realizable order type. There exist various ways to encode abstract order types optimally (see §2.2.6). However, it is not known how to decode the orientation of one triple from any of those optimal encodings in, say, sublinear time. Moreover, since the information-theoretic lower bound for realizable order types is only $\Omega(n \log n)$, we must ask if this space bound is approachable for those order types while keeping orientation queries reasonably efficient. Our contributions rely heavily on the hierarchical cuttings of Chazelle [48] described in detail in §8.2.

For ease of presentation, we state our results in terms of the encoding definition of §1.2.1 that we restate here.

Definition 2. For fixed k and given a function $f : [n]^k \to [O(1)]$, we define a ($S(n)$,$Q(n)$)-encoding of f to be a string of $S(n)$ bits such that, given this string and any $i \in [n]^k$, we can compute $f(i)$ in $Q(n)$ time in the word-RAM model with word size $w \geq \log n$.

In §C.1, we give the first optimal encoding for abstract order types that allows efficient query of the orientation of any triple: the encoding is a data structure that uses $O(n^2)$ bits of space with queries taking $O(\log n)$ time in the word-RAM model with word size $w \geq \log n$.

Contribution 7. *All abstract order types have an encoding using $O(n^2)$ bits of space and allowing for queries in $O(\log n)$ time.*

Our encoding is far from being space-optimal for realizable order types. We show that its construction can be easily tuned to only require slightly subquadratic space in this case.

Contribution 8. *All realizable order types have a $O(\frac{n^2(\log\log n)^2}{\log n})$-bits encoding allowing for queries in $O(\log n)$ time.*

In §C.2, we further refine our encoding to reduce the query time to $O(\log n/\log\log n)$.

Contribution 9. *All abstract order types have an encoding using $O(n^2)$ bits of space and allowing for queries in $O(\frac{\log n}{\log\log n})$ time.*

Contribution 10. *All realizable order types have a $O(\frac{n^2\log^\epsilon n}{\log n})$-bits encoding allowing for queries in $O(\frac{\log n}{\log\log n})$ time.*

In the realizable case, we give quadratic upper bounds on the preprocessing time required to compute an encoding in the real-RAM model.

Contribution 11 (Contributions 17 and 18)**.** *In the real-RAM model and the constant-degree algebraic decision tree model, given n real-coordinate input points in $\mathbb{R}^2$ we can compute the encoding of their order type as in Contributions 7, 8, 9, and 10 in $O(n^2)$ time.*

In §C.3 we generalize our encodings to chirotopes of point sets in higher dimension.

Table 3.3. Past results and new contributions on order type encoding.

Reference	Query Time	Space (bits)	Preprocessing
Trivial	$O(1)$	$O(n^3)$	$O(n^3)$
Enumeration	$2^{\Omega(n \log n)}$	$O(n \log n)$	$2^{\Omega(n \log n)}$
λ-matrices [85]	?	$O(n^2 \log n)$	$O(n^2)$
Numerical [90]	$O(n)$	$2^{\Theta(n)}$	?
Permutations [104, 149]	$O(1)$	$O(n^2 \log n)$	$O(n^2)$
Wiring diagrams [73, 74]	$O(n^2)$	$O(n^2)$	$O(n^2)$
Canonical Labeling [17]	?	$O(n^2)$	$O(n^2)$
Abstract §C.1,§C.2 (9)	$O(\frac{\log n}{\log \log n})$	$O(n^2)$	$O(n^2)$
Realizable §C.1,§C.2 (8)	$O(\log n)$	$O(\frac{n^2 \log^2 \log n}{\log n})$	$O(n^2)$
Realizable §C.1,§C.2 (10)	$O(\frac{\log n}{\log \log n})$	$O(\frac{n^2}{\log^{1-\epsilon} n})$	$O(n^2)$
Realizable in $\mathbb{R}^d$ §C.3 (12)	$O(\frac{\log n}{\log \log n})$	$O(\frac{n^d \log^2 \log n}{\log n})$	$O(n^d)$

Contribution 12. *All realizable chirotopes of rank $k \geq 4$ have an encoding using $O(\frac{n^{k-1}(\log \log n)^2}{\log n})$ bits of space and allowing for queries in $O(\frac{\log n}{\log \log n})$ time.*

Contribution 13. *In the real-RAM model and the constant-degree algebraic decision tree model, given n real-coordinate input points in $\mathbb{R}^d$ we can compute the encoding of their chirotope as in Theorem 12 in $O(n^d)$ time.*

Table 3.3 gives a summary of our results.

A Remark Our data structure is the first subquadratic encoding for realizable order types that allows efficient query of the orientation of any triple. It is not known whether a subquadratic algebraic computation tree exists for identifying the order type of a given point set. Any such computation tree would yield another subquadratic encoding for realizable order types, although the obtained encoding might not be query-efficient. We see the design of compact encodings for realizable order types as a subgoal towards subquadratic nonuniform algorithms for this related problem, a major open problem in Computational Geometry. Note that pushing the preprocessing time below quadratic would yield such an algorithm.

3.4 Better Encodings for 3SUM

In Paper D, we consider the following problem: given three sets of real numbers, output a word-RAM data structure from which we can efficiently recover the sign of the sum of any triple of numbers, one in each set.

This problem is similar to the problem of encoding the order type of a finite set of points studied in Paper C. While this contribution showed that it was possible to achieve slightly subquadratic space and logarithmic query time, we show here that we can encode the *3SUM type* of n numbers using $\tilde{O}(n^{3/2})$ bits to allow for constant time queries in the word-RAM. We also study lower and upper bounds of other natural 3SUM type encodings.

As there are only $O(n^3)$ queries, a table of size $n^3 \log_2 3 + O(1)$ bits suffices to give constant query time [60]. This can be improved to $O(n^2 \log n)$ bits of space by storing for each pair (i, j) the values $k_<(i, j) = \max\{0\} \cup \{k\colon\ a_i + b_j + c_k < 0\}$ and $k_>(i, j) = \min\{n+1\} \cup \{k\colon\ a_i + b_j + c_k > 0\}$. For a query (i, j, k), we compare k against the values $k_<(i, j)$ and $k_>(i, j)$ to recover $\chi(i, j, k)$ in $O(1)$ time. All $k_<(i, j)$ and $k_>(i, j)$ can be computed in $O(n^2)$ time via the classic quadratic time algorithm for 3SUM.

One seemingly simple representation is to store the numbers in A, B and C; however these are reals and thus we need to make them representable using a finite number of bits. In §D.1 we show that a minimal integer representation of a 3SUM instance may require $\Theta(n)$ bits per value, which would give rise to a $O(n)$ query time and $O(n^2)$ space, which is far from impressive.

Contribution 14. *Every 3SUM instance has an equivalent integer instance where all values have absolute value at most $2^{O(n)}$. Furthermore, there exists an instance of 3SUM where all equivalent integer instances require numbers at least as large as the nth Fibonacci number and where the standard binary representation of the instance requires $\Omega(n^2)$ bits.*

In [38] the problem of given a set of n lines, to create an encoding of them so that the orientation of any triple (the *order type*) can be determined was studied; our problem is a special case of this where the lines only have three slopes. Can we do better for the case of 3SUM? We answer this in the affirmative.

Again, for ease of presentation, we state our results in terms of the encoding definition of §1.2.1 that we restate here.

Table 3.4. Past results and new contributions on 3SUM type encoding.

Reference	Query time	Space (bits)	Preprocessing
Trivial	$O(1)$	$O(n^3)$	$O(n^3)$
Almost trivial	$O(1)$	$O(n^2 \log n)$	$O(n^2)$
Order type (Paper C)	$O(\log n)$	$O(\frac{n^2 \log^2 \log n}{\log n})$	$O(n^2)$
Order type (Paper C)	$O(\frac{\log n}{\log \log n})$	$O(\frac{n^2}{\log^{1-\epsilon} n})$	$O(n^2)$
Numerical §D.1(14)	$O(n)$	$O(n^2)$	$n^{O(1)}$
Space-optimal §D.2(15)	$n^{O(1)}$	$O(n \log n)$	$n^{O(1)}$
Query-optimal §D.3(16)	$O(1)$	$\tilde{O}(n^{3/2})$	$O(n^2)$

Definition 2. For fixed k and given a function $f : [n]^k \to [O(1)]$, we define a ($S(n)$,$Q(n)$)-encoding of f to be a string of $S(n)$ bits such that, given this string and any $i \in [n]^k$, we can compute $f(i)$ in $Q(n)$ time in the word-RAM model with word size $w \geq \log n$.

In §D.2 we show how to use an optimal $O(n \log n)$ bits of space with a polynomial query time.

Contribution 15. *All 3SUM types have a $O(n \log n)$-bits $n^{O(1)}$-querytime encoding.*

Finally, in §D.3 we show how to use $\tilde{O}(n^{3/2})$ space to achieve $O(1)$-time queries.

Contribution 16. *All 3SUM types have a $\tilde{O}(n^{3/2})$-bits $O(1)$-querytime encoding.*

Table 3.4 gives a summary of past and new results.

4

Developments

Since the publication of our results, new developments have surfaced.

4.1 Better Nonuniform Algorithms for k-SUM

Meiser's algorithm solves a broader class of problems than the k-SUM problems. In its general formulation, this algorithm solves the point location problem (Problem 8): to locate an input point in a fixed arrangement of m hyperplanes in R^d.

Our analysis of Meiser's algorithm yields a nonuniform upper bound of $O(d^3 \log d \log m)$ for this problem. However, this is far from optimal. The information theoretic lower bound obtained through Buck's theorem (Theorem 6.1) is only $\Omega(d \log m)$.

Two new developments have surfaced. The first reduces the nonuniform complexity of Meiser's algorithm for the point location problem from $O(d^3 \log d \log m)$ down to $O(d^2 \log m)$. The second invents a completely novel technique to reduce the nonuniform complexity of the same problem down to $O(d \log d \log m)$, provided the coefficients of the hyperplanes are constants.

We briefly sketch those two developments in the next subsections.

4.1.1 Using Vertical Decomposition

In 2016, Ezra and Sharir [72] improve the decision tree depth of Meiser's algorithm by using vertical decomposition (see §6.4.2) as the cell refinement subroutine.

A big player in the complexity of our algorithm is the size of the sample needed for the ε-net. It turns out that with vertical decomposition the sample size can be significantly reduced: $O(d)$ instead of $O(d^2 \log d)$ for simplices. A

sample size of $O(d \log d)$ can easily be attained using existing results on the VC-dimension of certain range spaces. One of the key results in [72] is to get the sample size down to $O(d)$.

For instance, when applying this improved algorithm to the k-SUM problem or the subset-sum problem, the change in the sample size causes Meiser's decision tree to be much shallower than our implementation: the depth of the tree is reduced by a $n \log n$ factor in both cases leading to overall complexities of $O(n^2 \log n)$ and $O(n^3)$ respectively.

4.1.2 Using Inference Dimension

In 2017, Kane, Lovett, and Moran [103] proved that the point location problem can be solved in $O(d \log d \log m)$ linear queries.

Their decision tree is also a prune and search algorithm following the generic approach of Meiser: sample, locate, prune, recurse. The key ingredient is a new tool dubbed *inference dimension* that takes the role of the VC-dimension in Meiser's algorithm.

The major catch with this new decision tree is that it will only work for sparse hyperplanes: hyperplanes with a small hamming weight. This is precisely the case for the application of point location to the k-SUM and subset-sum problems. For those applications, the decision trees in [103] are shallower than the decision trees in [72] by a factor of $n/\log n$: $O(n \log^2 n)$ for k-SUM and $O(n^2 \log n)$ for subset-sum.

Another feature of this algorithm is that it only uses *comparison queries*. When applied to k-SUM, our algorithm and the one in [72] are n-linear decision trees, while the one in [103] is a $2k$-linear decision tree.

In 2018, the authors of [103] published another paper generalizing their decision tree to work for arbitrary hyperplanes [102]. However, the depth they obtain is much worse in that case: $O(d^4 \log d \log m)$ for arbitrary hyperplanes. Moreover, it requires the use of *generalized comparison queries* which are more complex than the comparison queries in [103].

In 2019, Hopkins, Kane, and Lovett [99] provide another decision tree for point location of depth $O(d \log^2 m)$ when the set of hyperplanes is drawn from a distribution with weak restrictions. This new algorithm only uses comparison queries.

4.2 Timothy Chan Strikes Again

In 2017, Timothy Chan presented a $O((n^2/\log^2 n)(\log\log n)^{O(1)})$-time real-RAM algorithm for 3SUM [46], shaving a logarithmic factor from previous solutions [79, 81].

His technique relies on cuttings in near-logarithmic dimension on an augmented real-RAM model with constant time nonstandard operations on $\Theta(\log n)$ bits words. Cuttings essentially replace dominance reporting in Grønlund and Pettie's algorithm [91]. This is inspired by another one of his papers on APSP [45].

This new technique for shaving off logarithmic factors can be applied to other problems, such as $(\text{median}, +)$-convolution, $(\text{median}, +)$-matrix multiplication, and 3SUM-hard problems in computational geometry including explicit 3POL.

5

Open Questions

These are a few interesting questions that are left unanswered.

5.1 About Algorithms

The best known linear decision tree for k-SUM has depth $O_k(n \log^2 n)$ while the lower bound is $\Omega_k(n \log n)$ [103] (see §4.1.2). Can we match the information theoretic lower bound for k-SUM with linear decision trees?

Open Question 1. *Is there a $O_k(n \log n)$-depth LDT for k-SUM?*

The same question can be asked about the point location problem. The best linear decision tree for arbitrary hyperplanes has depth $O(d^2 \log m)$ and the lower bound is $\Omega(d \log m)$ [72] (see §4.1.1).

Open Question 2. *Is there a $O(d \log m)$-depth LDT for point location?*

In Paper A we show how to efficiently implement Meiser's algorithm in the word-RAM model when applied to k-SUM. Can the same be done for the algorithms of Ezra and Sharir [72] and Kane, Lovett, and Moran [103]?

Open Question 3. *Is there a word-RAM algorithm for k-SUM running in time $n^{\frac{k}{2}+O(1)}$ accessing the input through $n^{3-\Omega(1)}$ n-linear queries only?*

We also have not managed to match the running time of the best known uniform algorithm (see §2.1.5).

Open Question 4. *Is there a word-RAM algorithm for k-SUM running in time $\tilde{O}(n^{\lceil \frac{k}{2} \rceil})$ accessing the input through $n^{o(k)}$ n-linear queries only?*

Grønlund and Pettie [91] showed how to solve 3SUM in subquadratic time in the real-RAM model. However, so far, their technique and its subsequent improvements [46, 79, 81], have failed to generalize to uniform algorithms for k-SUM.

Open Question 5. *Is there a $o(n^{\lceil \frac{k}{2} \rceil})$-time real-RAM algorithm for k-SUM with $k \geq 4$?*

The best known algorithms for 3SUM are only *slightly* subquadratic, shaving a polylogarithmic factor from quadratic runtime. In Paper B, we show how to adapt one of those algorithms to solve the 3POL problem within the same time bound. The best lower bound we have for those problems is $\Omega(n \log n)$. The current conjecture is that no significant improvement can be made with respect to known algorithms (see §2.1). We have to ask if this is indeed true.

Open Question 6. *Is there a $n^{2-\Omega(1)}$-time real-RAM algorithm for 3SUM?*

The question can also be asked more generally for 3POL.

Open Question 7. *Is there a $n^{2-\Omega(1)}$-time real-RAM algorithm for 3POL?*

The best known nonuniform algorithm for 3SUM is a 6-linear decision tree of depth $O(n \log^2 n)$ (see §4.1.2), the nonuniform algorithms in [79, 81, 91] are 4-linear decision trees of depth $\tilde{O}(n^{3/2})$, and we know that there is no 3-linear decision tree of depth $o(n^2)$ for 3SUM [69] (see §2.1). We want to know whether better 4-linear decision trees exists for 3SUM.

Open Question 8. *Is there a $o(n^{\frac{3}{2}})$-depth 4-linear decision tree for 3SUM?*

If this is the case, then we would also like to know if this model of computation suffers from more severe limitations than the 6-linear decision tree model.

Open Question 9. *Is there a $n \log^{O(1)} n$-depth 4-linear decision tree for 3SUM?*

The inference dimension method introduced in [103] is essentially a pigeon hole argument applied to linear combinations of linear equations with small coefficients. At first sight, this kind of argument does not seem to apply to

polynomial equations, even when the polynomials are of degree two. One can of course apply a Veronese embedding [95, 96] to linearize the equations but this has the effect of blowing up the dimension which directly impacts the efficiency of the method. Could some generalization of inference dimension, or any other method, improve on the $O(n^{12/7+\varepsilon})$ nonuniform time bound we obtain in Paper B?

Open Question 10. *Is there a $o(n^{12/7})$-depth bounded-degree algebraic decision tree or algebraic computation tree for 3POL?*

In §B.3.1 we show how 3POL algorithms can be used to solve particular instances of GPT in subquadratic time. Despite all our efforts, we still lack a subquadratic algorithm for GPT in the general case, even a nonuniform one. Like 3SUM and 3POL, the best known lower bound is $\Omega(n \log n)$ (see §2.2.3).

Open Question 11. *Is there a $o(n^2)$-depth bounded-degree algebraic decision tree or algebraic computation tree for GPT?*

5.2 About Encodings

In Paper C we show how to get compact encodings for abstract order types with slightly sublogarithmic query time. Is it possible to bring this query time down to constant, or event just strongly sublogarithmic?

Open Question 12. *Is there a $O(n^2)$-bits $O(\log^{1-\epsilon} n)$-querytime encoding for abstract order types?*

In the same paper, we manage to get slightly subquadratic space encodings for realizable order types. Is it possible to get this down to strongly subquadratic while keeping the query time reasonable, say polynomial?

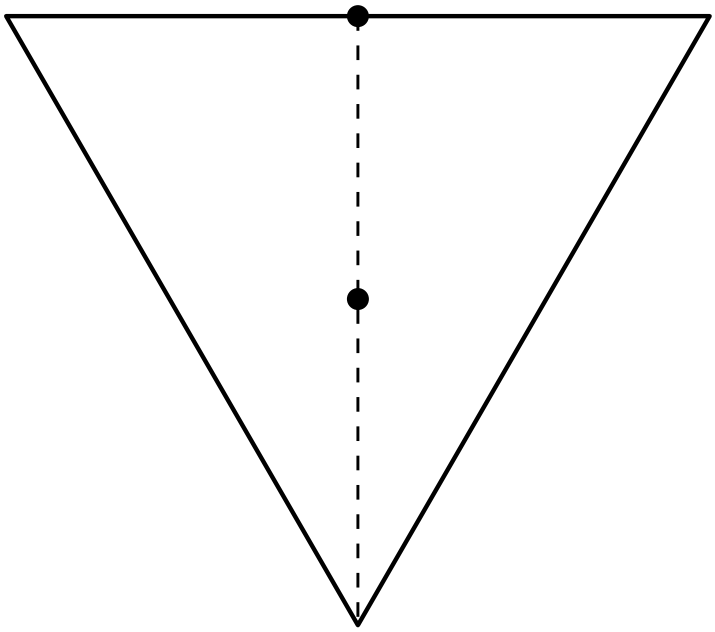
Open Question 13. *Is there a $n^{2-\Omega(1)}$-bits $n^{O(1)}$-querytime encoding for realizable order types?*

In Paper D we modify Grønlund and Pettie's nonunifom algorithm to obtain a 3SUM type encoding with a strongly subquadratic space requirement and efficient queries. Is there any way to exploit the nearlinear decision tree in [103] to obtain a better space bound while keeping query time efficient?

Open Question 14. *Is there a $o(n^{3/2})$-bits $\log^{O(1)} n$-querytime encoding for 3SUM types?*

II

The Computational Geometer's Toolbox



6

Arrangements

An *hyperplane* in $\mathbb{R}^d$ is the set of points $q \in \mathbb{R}^d$ satisfying a linear equation of the form $\sum_{i=1}^{d} p_i q_i = 0$ for some tuple of coefficients $p \in \mathbb{R}^d$. An *affine* hyperplane has an additional independent term p_0 to get the equation $\sum_{i=1}^{d} p_i q_i = p_0$. In general we will drop the adjective "affine" and talk about hyperplanes without specifying whether $p_0 = 0$. In the plane an hyperplane is a line. In $\mathbb{R}^3$ an hyperplane is a plane. On the real line an hyperplane is a point.

An hyperplane H partitions the space $\mathbb{R}^d$ into two halfspaces, often denoted by H^- and H^+. Naturally, H^- is defined to be the set of points such that $\sum_{i=1}^{d} p_i q_i < p_0$ and H^+ is defined to be the set of points such that $\sum_{i=1}^{d} p_i q_i > p_0$. When we want to be more precise as to whether the inequality is strict we can write $H^<$, $H^\leq$, $H^>$, or $H^\geq$. We also define H^0 and $H^=$ as synonyms of H.

A set of hyperplanes $\mathcal{H} = \{ H_1, H_2, \ldots, H_m \}$ partitions $\mathbb{R}^d$ into convex regions called *cells*. This partition is denoted by $\mathcal{A}(\mathcal{H})$ and is called the *arrangement* of $\mathcal{H}$. Each sign vector $\sigma \in \{-, 0, +\}^m$ corresponds to a (potentially empty) cell of this arrangement defined as the set of points q such that $q \in H_i^{\sigma_i}$ for all $H_i \in \mathcal{H}$.

6.1 Counting Cells

It is useful to understand the complexity of such an arrangement when the number m of hyperplanes grows and the dimension d is fixed. The goal is to bound the number of nonempty cells an arrangement can have. When $m \leq d$ each of the 3^m cells is nonempty, assuming general position. However, when we fix d and make m grow, we get a more reasonable behavior.

Theorem 6.1 (Buck [35], see also [93, Theorem 24.1.1 and Corollary 24.1.2])**.** *Consider the partition of space defined by an arrangement of m hyperplanes in $\mathbb{R}^d$. The number of regions of dimension $k \leq d$ is at most*

$$\binom{m}{d-k} \sum_{i=0}^{k} \binom{m-d+k}{i}.$$

This bound is attained when the hyperplanes are in general position. The number of regions of all dimensions is $\Theta(m^d)$ in that case.

This bound allows us to derive precise lower and upper bounds for algorithms manipulating those arrangements. See [93] for more details and generalizations.

6.2 Pseudolines

In the plane, line arrangements are generalized to pseudoline arrangements by dropping the "straight" nature of lines. A pseudoline in $\mathbb{R}^2$ is a simple curve connecting points at infinity. An arrangement of pseudolines is a collection of pseudolines that pairwise intersect exactly once. This definition can be made more formal by considering pairwise-intersecting simple closed curves in the projective plane instead. Some important differences between line and pseudoline arrangements have already been discussed in §2.2.5.

Putting those differences aside, pseudoline arrangements share sufficient similarity with line arrangements as to allow a generic algorithmic treatment of both kinds. For instance, their arrangement complexities (§6.1) are asymptotically the same, the Zone Theorem (§6.3) holds in general, canonical labelings (§7.2) can be constructed by looking at the order type only (see §2.2.3 for a definition of order type), and generic cell decompositions (§6.4) allow the use of the divide-and-conquer methods of Chapter 8 in both cases.

In Paper C we consider the problem of encoding order types of line or pseudoline arrangements. Strictly speaking, an order type consists of pure combinatorial information and does not require an embedding to be known. In particular, given an order type, it is $\exists\mathbb{R}$-complete to decide whether a straigth embedding exists [118, 119]. For simplicity, the algorithms constructing the encodings in Paper C assume an embedding is given.

For more on pseudoline arrangements see Chapter 5 of the Handbook on the subject [83]. For more on equivalent representations of pseudoline

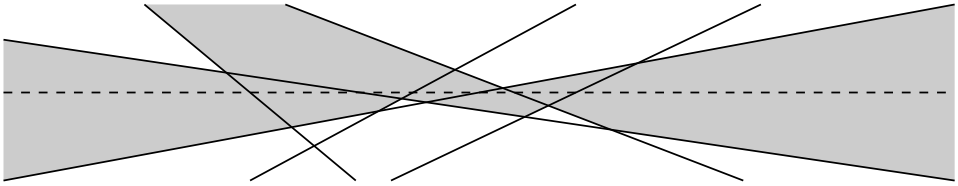


Figure 6.1. The zone defined by the dashed line in the two-dimensional arrangement of the plain lines is emphasized in light grey.

arrangements see Chapter 6 of the Handbook on oriented matroids [139].

6.3 Zone Theorem

The zone of a given pseudoline of an arrangement is the set of cells of the arrangement supported by that pseudoline. Figure 6.1 illustrates a zone in a two-dimensional arrangement of lines.

We define the complexity of each cell to be the number of its sides. We define the complexity of a zone to be the sum of the complexities of its cells. The Zone Theorem states that the complexity of any zone is linear.

Theorem 6.2 (Zone Theorem in the plane [27])**.** *Given an arrangement of $n+1$ pseudolines, the sum of the numbers of sides in all the cells supported by one of the pseudolines is at most $\lfloor 9.5n \rfloor - 1$.*[1]

This result is important because it allows optimal incremental construction of arrangements of line arrangements, a frequently used tool.

In [63] and in the first edition of [61], there were claims of generalization of this result to arrangements of hyperplanes in higher dimension. However, the proofs turned out to be flawed [64]. Edelsbrunner, Sturmfels, and Sharir were the first to provide a valid proof of the generalized result [64].

Theorem 6.3 (Zone Theorem [64])**.** *Given an arrangement of n hyperplanes in $\mathbb{R}^d$, the sum of the numbers of 0-faces, 1-faces, ..., and $(d-1)$-faces in all the cells supported by one of the hyperplanes is $O(n^{d-1})$.*

They also cite valid proofs by Houle and Matoušek for a weaker version of the theorem: The Zone Theorem bounds the sum of complexities of the

[1] Note that an earlier weaker (worse constant factor) linear bound is implied by a theorem in [50].

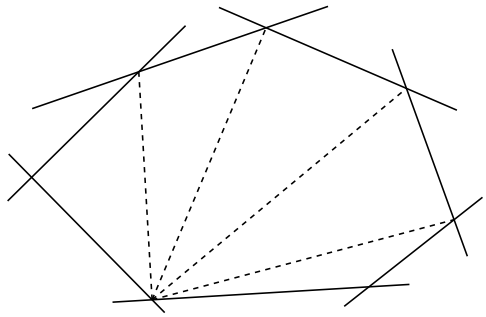


Figure 6.2. The bottom vertex triangulation of a cell in an arrangement of lines.

cells in a zone. The full generalization of the theorem defines the complexity of each cell to be the number of all 0-faces, 1-faces, ..., and $(d-1)$-faces supporting the cell. The weak generalization only counts $(d-1)$-faces. However, no publication of those proofs is known to the author.

Note that the Zone Theorem implies the asymptotic upper bound of $O(n^d)$ on the complexity of the arrangement (§6.1).

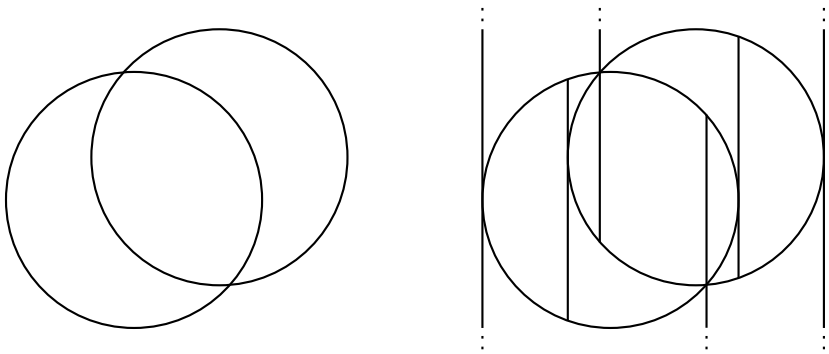
6.4 Cell Decomposition

Cells of arrangements may not behave nicely. In particular, they may have arbitray description complexity. In this section we give two decomposition schemes that allow to partition cells into low-complexity subcells. Access to such decompositions is an essential ingredient of divide-and-conquer methods described in Chapter 8.

Lookup the Handbook [93, §24.3.2], for a more general overview of cell decomposition techniques.

6.4.1 Bottom Vertex Triangulation

Given an arrangement of hyperplanes in $\mathbb{R}^d$, its bottom vertex triangulation is the partition of space obtained by triangulating each cell of the arrangement recursively as follows: taking $d = 2$ as the base case (because edges are already simplices), let p be the bottom-most vertex of the cell, for each facet of the cell not containing p, for each $(d-1)$-dimensional simplex S' of the bottom vertex triangulation of the facet in $\mathbb{R}^{d-1}$, construct a d-dimensional simplex S that is the convex hull of S' and p. The bottom



a. Some curves in $\mathbb{R}^2$.

b. Vertical decomposition of the curves in Figure 6.3a.

Figure 6.3. Vertical decomposition.

vertex triangulation of an arrangement of lines in the plane is illustrated in Figure 6.2.

In Paper A we construct a simplex of the bottom vertex triangulation of a ε-net as part of a point location procedure. In Paper B (§B.2.2) and Paper C (all sections) we use hierarchical cuttings based on this decomposition in order to construct an efficient point location data structure.

See also [53] for a more thorough description and an application to nearest neighbour data structures.

6.4.2 Vertical Decomposition

Given an arrangement of curves in $\mathbb{R}^2$, its vertical decomposition is the partition of space obtained by shooting a vertical segment from each vertex of the arrangement until it hits a curve of the arrangement. The vertical decomposition of an arrangement of two circles is illustrated in Figure 6.3.

We use this decomposition in Paper B (§B.2.1) in order to apply a divide-and-conquer algorithm of Matoušek [109] to an arrangement of curves. This algorithm was originally designed to work for an arrangement of hyperplanes using bottom-vertex triangulation decomposition. In Paper C (§C.1, §C.2), we use this decomposition in order to encode the order type of an arrangement

of pseudolines.

Note that in the first application we only use vertical decomposition on a constant number of bounded-degree polynomial curves. In the second application we only care about the existence of such a decomposition. For those reasons, we do not discuss efficient construction of this decomposition.

For the construction of the vertical decomposition of an arrangment of polynomial curves in $\mathbb{R}^2$, we refer the reader to Pach and Sharir [123], Chazelle et al. [49], and Edelsbrunner et al. [62].

A Remark This decomposition can be generalized to work for hypersurfaces in $\mathbb{R}^d$. Unfortunatly, the behaviour of such decompositions quickly degenerates with d. While bottom vertex triangulation gives a bound of $O(n^d)$ cells in $\mathbb{R}^d$ and vertical decomposition gives a bound of $O(n^2)$ cells in $\mathbb{R}^2$, we only know of a few upper bounds in fixed dimensions and some of them are worse than $O(n^d)$. This is why in our application of Meiser's algorithm we use the bottom vertex triangulation. However, as observed by Ezra and Sharir [72], the bad behaviour of vertical decompositions is not a bottleneck of Meiser's algorithm since we only need to look at a single cell of the decomposition. Moreover, the complexity of those cells is better than that of simplicial ones: vertical decomposition yields prisms supported by at most $2d$ hyperplanes of the arrangement whereas simplices of the bottom vertex triangulation are supported by $d^2 + d$ hyperplanes in the worst case. This has a direct impact on the VC-dimension of the range spaces defined by those objects and makes vertical decomposition the winning strategy for this particular algorithm.

7

Chirotopes

Chirotopes are the generalization of order types introduced in §2.2.3. We generalize Definitions 3 and 4 to point sets in $\mathbb{R}^d$.

Definition 5 (Chirotope of a Point Set in $\mathbb{R}^d$)**.** Given a set of n points $p_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,d}) \in \mathbb{R}^d$, its (rank-$(d+1)$) chirotope is the function $\chi\colon [n]^{d+1} \to \{-, 0, +\}$ such that

$$\chi(i_1, i_2, \ldots, i_{d+1}) = \det \begin{pmatrix} 1 & p_{i_1,1} & p_{i_1,2} & \cdots & p_{i_1,d} \\ 1 & p_{i_2,1} & p_{i_2,2} & \cdots & p_{i_2,d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & p_{i_{d+1},1} & p_{i_{d+1},2} & \cdots & p_{i_{d+1},d} \end{pmatrix},$$

up to isomorphism.

Definition 6 (Realizable Chirotope)**.** When considering an arbitrary function $f\colon [n]^{d+1} \to \{-, 0, +\}$ we say that f is a realizable chirotope if there is a n-set $P \subset \mathbb{R}^2$ such that its chirotope is f.

In this chapter, we give details on two essential ingredients of our data structures in Paper C: the classic point-hyperplane duality and the canonical labeling of order types and chirotopes. The first one is necessary to be able to apply the encodings we obtain for line arrangements and hyperplane arrangements to point configurations. The second is used to get the preprocessing time of those encodings down to $O(n^d)$.

7.1 Duality

Technically speaking, the encoding we describe for realizable chirotopes in Paper C encodes the chirotope of a given arrangement of lines or hyperplanes.

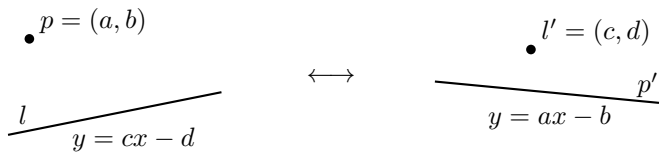


Figure 7.1. Order preserving duality: "p is above l" if and only if "l' is above p'".

Moreover, for ease of presentation, we make the assumption that the vertices of this arrangement have finite coordinates. In the two-dimensional case, this is equivalent to having no two lines parallel. In these paragraphs, we give the details necessary to rigorously handle all realizable chirotopes, including degenerate ones. This is especially important in higher dimension, where the situation is a bit more complicated than in two dimensions.

In two dimensions, we wish to encode order types of point configurations. Since our encoding construction algorithm works with an arrangement of lines as input, we need a mapping from those primal points to their dual lines. This mapping should preserve the order type of the point configuration, hence it needs to be *order-preserving*. One such order-preserving duality is the mapping $(a, b) \leftrightarrow y = ax - b$ (see Figure 7.1).

To avoid parallel lines in the dual, it suffices to avoid intersection points at infinity. In the primal, this translates to avoiding two points of the configuration defining a vertical line, that is, with the same x coordinate. This is easily done by performing a tiny rotation in the primal. This (proper) rotation does not change the order type of the point set.

In higher dimension, the order-preserving point-line duality generalizes to the following order-preserving point-hyperplane duality: We map each d-dimensional point $(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ to the hyperplane $y_d = \sum_{i=1}^{d-1} x_i y_i - x_d$ and the hyperplane $x_d = \sum_{i=1}^{d-1} y_i x_i - y_d$ to the d-dimensional point $(y_1, y_2, \ldots, y_d) \in \mathbb{R}^d$.

As before, we want hyperplanes to be non-parallel. In fact, we need an even stronger assumption: We want all linearly independent subsets of d hyperplanes to intersect in a point with finite coordinates. Having no intersection points at infinity in the dual means having no d points spanning a hyperplane parallel to the x_d axis in the primal. This is easy to avoid by applying tiny rotations in the primal. Again, those (proper) rotations do

not change the chirotope of the point set.

In dimension three and higher, one would think degenerate arrangements lead to annoying nongeneral situations. However, those situations are easy to handle with our technique: Degenerate subsets of hyperplanes are linearly dependent. The determinant corresponding to a query asking about a degenerate $d+1$ subset is therefore zero. Our technique will identify those degenerate queries and map them to the correct answer in a space-efficient way.

7.2 Canonical Labelings

Given a point set, the composition of its order type χ with a permutation ρ produces a new order type $\chi' = \chi \circ \rho$. This composition corresponds to a relabeling of the point set. Aloupis et al. [17] defined the canonical labeling $\rho^*(\chi)$ of an order type χ to be a permutation such that for all permutations π we have $\rho^*(\chi \circ \pi) = \pi^{-1} \circ \rho^*(\chi)$. In other words, given two isomorphic order types χ and χ', we have $\chi \circ \rho^*(\chi) = \chi' \circ \rho^*(\chi')$, and $\rho^*(\chi')^{-1} \circ \rho^*(\chi)$ is the isomorphism that sends χ to χ'.[1] They proved that the function ρ^* is computable in $O(n^2)$ time. This first tool is useful to identify isomorphic order types.

They also showed that given any order type χ, a string $E(\chi)$ of $O(n^2)$ bits, called the representation of χ, can be computed in $O(n^2)$ time, such that, if χ and χ' are two isomorphic order types, then $E(\chi) = E(\chi')$. This second tool is useful to quickly compare two order types (a naive solution would take $\Theta(n^3)$ time by first computing a canonical labeling, and then comparing all triples).

Lemma 7.1 (Aloupis et al. [17])**.** *Given an order type presented as an oracle, its canonical labeling of $O(n \log n)$ bits and its canonical representation of $O(n^2)$ bits can be computed in $O(n^2)$ time in the word-RAM model.*

Both tools generalize to chirotopes of point configurations in any dimension d and, more generally, to chirotopes of rank $d+1$.

Lemma 7.2 (Aloupis et al. [17])**.** *For all $d \geq 2$, given a rank-$(d+1)$ chirotope presented as an oracle, its canonical labeling of $O(n \log n)$ bits and its*

[1] Sometimes, two order types χ and $-\chi$ are also considered to be isomorphic. See [17] for more details.

canonical representation of $O(n^d)$ bits can be computed in $O(n^d)$ time in the word-RAM model.

8

Divide and Conquer

A general divide-and-conquer paradigm in algorithm design is, given an instance of size n, to 1) divide it into at most a smaller instances of size at most $\frac{n}{b}$ in $p(n)$ preprocessing time, 2) solve those instances recursively, solving constant size instances by brute force, and 3) postprocess the solutions of those smaller instances to obtain the solution to the original one in $p(n)$ time. An upper bound $T(n)$ on the time complexity of such an algorithm is the reccurence

$$T(n) = a\ T\left(\frac{n}{b}\right) + p(n).$$

The asymptotic behavior of such a recurrence can then be derived from the Master Theorem [26, 55].

In this chapter we expose a couple standard divide-an-conquer techniques in Computational Geometry that allow the implementation of this paradigm (or variants of it). Papers A, B, and C explicitly rely on those techniques. The subquadratic-space constant-querytime encoding in Paper D can also be interpreted as an ad-hoc implementation of those concepts (see §D.3).

8.1 Epsilon Nets and Cuttings

The efficient construction of ε-nets and cuttings is a necessary condition for many divide-and-conquer schemes in Computational Geometry. Those constructions are based on the concept of Vapnik-Chervonenkis dimension (VC-dimension) of a range space.

8.1.1 Range Spaces

A *range space* (or set system) consists of a *ground set* (or universe) and a family of subsets of this ground set called *ranges*. A simple example of a

range space is to take a finite set of points on the real line as the ground set and to take the subsets of points induced by intervals as the ranges. In general, we have the following definition.

Definition 7 (Range Space)**.** A pair $(X, \mathcal{R})$ such that X is a set and $\mathcal{R} \subseteq 2^X$ is a family of subsets of X.

Note that, for simplicity, the examples, definitions, and lemmas we state here consider the case of finite ground sets. Those can be generalized to infinite universes.

In the worst case, $\mathcal{R}$ could be equal to 2^X. If X is taken to be points in $\mathbb{R}^d$ and the ranges are taken to be subsets induced by simple geometric objects, as in most geometric applications, this is not possible.

For instance, consider points and intervals on the real line. For any set of points, only $O(|X|^2)$ subsets can be induced by intervals. Another example is that of points in the plane and subsets induced by halfplanes. Because any halfplane can be moved to have exactly two points on its boundary, there are $O(|X|^2)$ such subsets. Those observations can be generalized. For that we need a few more definitions.

8.1.2 VC-dimension

A set is *shattered* by a family of ranges if each of its subsets can be induced by a range of the family.

Definition 8 (Shattered Set)**.** Let $(X, \mathcal{R})$ be a range space. A subset of $T \subseteq X$ is said to be shattered by $\mathcal{R}$ if every subset $T' \subseteq T$ is such that $T' = T \cap R$ for some $R \in \mathcal{R}$.

In our points and intervals example it is easy to see that a set of two distinct points can be shattered but a set of three points cannot. For the points and halfplanes example sets of three noncollinear points are shattered but not sets of four points.

The VC-dimension of a range space is defined to be the size of the largest shattered subset.

Definition 9 (VC-dimension)**.** Let $(X, \mathcal{R})$ be a range space. Let v be the size of the largest $T \subseteq X$ such that T is shattered by $\mathcal{R}$. The VC-dimension of the range space is defined to be v.

The Perles-Sauer-Shelah-Vapnik-Chervonenkis lemma gives a direct connection between this parameter and the number of ranges.

Lemma 8.1 (Vapnik and Chervonenkis [152], Sauer [141], Shelah [143])**.** *Let $(X, \mathcal{R})$ be a range space. If $|\mathcal{R}| > \sum_{i=0}^{k-1} \binom{|X|}{i}$ then the VC-dimension of the range space is at least k. Conversely, if the VC-dimension of the range space is v then $|\mathcal{R}| \leq \sum_{i=0}^{v-1} \binom{|X|}{i} = O(|X|^v)$.*

In our examples, notice the discrepancy between the "identical" $O(|X|^2)$ bounds on the number of ranges and the different VC-dimensions (two and three respectively).

8.1.3 Epsilon Nets

A net is a subset of the ground set such that it hits every large range.

Definition 10 (ε-net)**.** Let $(X, \mathcal{R})$ be a range space and let ε be a real number in the $[0, 1)$ interval. A subset $N \subseteq X$ is an ε-net for the range space if for every $R \in \mathcal{R}$ such that $|R| > \varepsilon|X|$ we have that $N \cap R \neq \emptyset$.

In our algorithms and data structures, we consider nets of range spaces in $\mathbb{R}^d$. They are used as follows: construct an ε-net, partition the space into a small number of ranges so that no range of the partition is hit by the net. Then we have the guarantee that none of those ranges is large. For many problems, such partitions of space with a small number of small ranges usually lead to practical divide-and-conquer schemes. These partitions are called *cuttings* (see §8.1.5).

A very neat result is that, for constant VC-dimension, a reasonably small ε-net can be constructed efficiently by random sampling.

Lemma 8.2. *Let $(X, \mathcal{R})$ be a range space with VC-dimension v. Let N be a uniform random sample of X of size $\Omega_v(r \log r)$. Then N is a $\frac{1}{r}$-net for $(X, \mathcal{R})$ with probability $1 - r^{-\Omega(1)}$.*

See [52, Section 40.4] for more on range spaces and ε-nets.

8.1.4 Hyperplanes in Linear Dimension

Consider the range space where the ground set is a finite set of hyperplanes $\mathcal{H}$ in $\mathbb{R}^n$ and where the ranges are the subsets of hyperplanes that can be obtained by intersecting the ground set with a simplex.

By combining a theorem due to Blumer et al. [29] with the results of Meiser [113][1], it is possible to obtain good bounds on ε-nets constructed by random sampling. In Paper A we are interested in the dependency of those bounds on the ambiant dimension n. This is not explicitly tackled in Lemma 8.2 since here the VC-dimension of this range space depends on n.

Lemma 8.3. *For all real numbers $r > 1$ and $c \geq 1$, if we choose at least $cn^2 \log nr \log r$ hyperplanes of $\mathcal{H}$ uniformly at random and denote this selection $\mathcal{N}$ then for any simplex intersected by more than $\frac{|\mathcal{H}|}{r}$ hyperplanes of $\mathcal{H}$, with probability $1 - 2^{-\Omega(c)}$, at least one of the intersecting hyperplanes is contained in $\mathcal{N}$.*

The contrapositive states that if no hyperplane in $\mathcal{N}$ intersects a given simplex, then with high probability the number of hyperplanes of $\mathcal{H}$ intersecting the simplex is at most $\frac{|\mathcal{H}|}{r}$.

8.1.5 Cuttings

A cutting in $\mathbb{R}^d$ is a set of (possibly unbounded and/or non-full dimensional) bounded-complexity cells that together partition $\mathbb{R}^d$. For our purposes, a cell is of bounded complexity if its boundary is defined by a number of lines, pseudolines, or hyperplanes of some arrangement that solely depends on the dimension, and not on the size of the arrangement. A $\frac{1}{c}$-cutting of a set of n elements is a cutting with the constraint that each of its cells is intersected by at most $\frac{n}{c}$ elements.

In constant dimension, a straightforward way to construct a $\frac{1}{c}$-cutting for a set of hyperplanes is to first contruct a $\frac{1}{c}$-net for the range space consisting of hyperplanes as elements and simplices as ranges and then triangulate its arrangement as in §6.4.1. Because none of the simplices of the triangulation intersect the net, each simplex is intersected by at most a $\frac{1}{c}$-fraction of the hyperplanes. Through this construction we obtain $O(c^d \log^d c)$ simplicial cells each intersected by at most $\frac{n}{c}$ hyperplanes.

For hyperplanes in constant dimension, there exist various ways of constructing $\frac{1}{c}$-cuttings of size $O(c^d)$, thereby removing the polylogarithmic factor overhead (see for instance §8.2).

[1] Note that Meiser used an older result due to Haussler and Welzl [97] and got an extra $\log n$ factor in the size of the ε-net. We thank Hervé Fournier for pointing this out.

8.1.6 Algebraic Range Spaces

In Paper B (§B.2.1) we use a a similar technique to obtain a partition of $\mathbb{R}^2$ into $O(c^2 \log^2 c)$ pseudotrapezoidal cells so that each cell is intersected by at most a $\frac{1}{c}$-fraction of some input polynomial curves. Because those curves have bounded degree, we can design a range space with finite VC-dimension and exploit it.

In the same paper (§B.2.2) we use the technique of linearization [5, 157] in order to tackle a more general problem in any constant dimension: given a system of polynomial inequalities, we can define a single variable for each monomial so that each inequality becomes linear. This has the effect of greatly increasing the ambient dimension but allows to use the techniques that normally only apply to linear ranges.

8.1.7 Derandomization

The ε-net construction based on sampling described above can be made deterministic. Derandomization is usually achieved through the method of conditional probabilities of Raghavan [129] and Spencer [145] (as in [48]). This method yields a construction time that is linear in the size of the ground set for ε-nets and ε-cuttings with constant ε.

Lemma 8.4 (Matoušek [110, Corollary 4.5])**.** *Let $(X, \mathcal{R})$ be a range space of finite VC-dimension and let $r > 1$ be a constant. Under some reasonnable assumptions, we can compute a $\frac{1}{r}$-net for $(X, \mathcal{R})$ of size $O(r \log r)$ in $O(|X|)$ time.*

For other concrete examples of derandomization of nets and cuttings, we refer the reader to Matoušek [111], Chazelle and Matoušek [51] and Brönnimann et al. [34]. The Handbook has a nice overview of the subject [52, Section 40.7]. See also [52, Section 40.1] for more on randomized divide-and-conquer and [52, Section 40.6] for more on derandomization.

8.2 Hierarchical Cuttings

Compared to standard cuttings, hierarchical cuttings have the additional property that they can be composed without multiplying the hidden constant factors in the big-O notation [48]. In particular, they allow for $O(n^d)$-space

$O(\log n)$-query d-dimensional point location data structures (for constant d). This is exploited in Paper B (§B.2.2) and Paper C (all sections).

Definition 11 (Hierarchical Cutting)**.** Given n hyperplanes in $\mathbb{R}^d$, a ℓ-levels hierarchical cutting of parameter $r > 1$ for those hyperplanes is a sequence of ℓ levels labeled $0, 1, \ldots, \ell - 1$ such that[2]

- Level i has $O(r^{2i})$ cells,
- Each of those cells is further partitioned into $O(r^2)$ subcells,
- The collection of subcells is a $\frac{1}{r^{i+1}}$-cutting for the set of hyperplanes,
- The $O(r^{2(i+1)})$ subcells of level i are the cells of level $i + 1$.

It is clear from reading through the various references that those hierarchical cuttings can be constructed for arrangements of pseudolines with the same properties: In [48], Chazelle first proves a vertex-count estimation lemma that only relies on incidence properties of line arrangements [48, Lemma 2.1]. Then he summons a lemma from [109] that relies on the finite VC-dimension of the range space at hand [48, Lemma 3.1]. Here the ground set is the set of pseudolines and the ranges are the subsets induced by intersections with pseudosegments. The VC-dimension of this range space is easily shown to be finite and is known to be at most 8: every arrangement of 9 pseudolines contains a subset of 6 pseudolines in hexagonal formation [94], which cannot be shattered.[3] Finally, he proves a lemma for the efficient construction of sparse ε-nets whose correctness again only relies on incidence properties of line arrangements [48, Lemma 3.2]. Using those three lemmas together with bottom vertex triangulation (§6.4.1) he is able to prove his main result:

Lemma 8.5 (Chazelle [48, Theorem 3.3])**.** *Given n hyperplanes in $\mathbb{R}^d$, for any real parameter $r > 1$, we can construct a ℓ-levels hierarchical cutting of parameter r for those hyperplanes in time $O(nr^{\ell(d-1)})$.*

For pseudoline arrangements, bottom vertex triangulation can be traded for vertical decomposition (§6.4.2).

[2] In [48], Chazelle refers to this parameter as r_0 and uses r to mean r_0^ℓ. Here we drop the subscript for ease of presentation.

[3] This a quote from [32]. We could not access the original paper.

Lemma 8.6. *Given n pseudolines, for any real parameter $r > 1$, we can construct a ℓ-levels hierarchical cutting of parameter r for those pseudolines in time $O(nr^\ell)$.*

In particular, we will use those lemmas with $\ell = \lceil \log_r \frac{n}{t} \rceil$, for some parameter t, so that the last level of the hierarchy defines a $\frac{t}{n}$-cutting whose cells are each intersected by at most t pseudolines (or hyperplanes).

Note that in [48] the construction is described for constant parameter r. This restriction on the parameter is easily lifted: We can construct a hierarchical cutting with superconstant parameter r by constructing a hierarchical cutting with some appropriate constant parameter r', and then skip levels that we do not need. This is used in §C.2 to reduce the query time from $O(\log n)$ to $O(\frac{\log n}{\log \log n})$.

9

Existential Theory of the Reals

The problems we consider in Paper B require our algorithms to manipulate polynomial expressions and, potentially, their real roots. For that purpose, we rely on Collins's cylindrical algebraic decomposition (CAD) [54]. To understand the power of this method, and why it is useful for us, we give some background on the related concept of first-order theory of the reals.

Definition 12. A Tarski formula $\phi \in \mathbb{T}$ is a grammatically correct formula consisting of real variables ($x \in \mathbb{R}$), universal and existential quantifiers on those real variables ($\forall, \exists \colon \mathbb{R} \times \mathbb{T} \to \mathbb{T}$), the boolean operators of conjunction and disjunction ($\wedge, \vee \colon \mathbb{T}^2 \to \mathbb{T}$), the six comparison operators ($<, \leq, =, \geq, >, \neq \colon \mathbb{R}^2 \to \mathbb{T}$), the four arithmetic operators ($+, -, *, / \colon \mathbb{R}^2 \to \mathbb{R}$), the usual parentheses that modify the priority of operators, and constant real numbers ($\mathbb{R}$). A Tarski sentence is a fully quantified Tarski formula. The first-order theory of the reals (FOTR) is the set of true Tarski sentences.

Tarski [150] and Seidenberg [142] proved that FOTR is decidable. However, the algorithm resulting from their proof has nonelementary complexity. This proof, as well as other known algorithms, are based on quantifier elimination, that is, the translation of the input formula to a much longer quantifier-free formula, whose validity can be checked. There exists a family of formulas for which any method of quantifier elimination produces a doubly exponential size quantifier-free formula [57].

9.1 Cylindrical Algebraic Decomposition

Collins's CAD matches this doubly exponential complexity.

Theorem 9.1 (Collins [54])**.** FOTR *can be solved in* $2^{2^{O(n)}}$ *time in the real-RAM model, where* n *is the size of the input Tarski sentence.*

See Basu, Pollack, and Roy [23] for additional details, Basu, Pollack, and Roy [22] for a singly exponential algorithm when all quantifiers are existential (existential theory of the reals, $\exists\mathbb{R}$), Caviness and Johnson [43] for an anthology of key papers on the subject, and Mishra [116] for a review of techniques to compute with roots of polynomials.

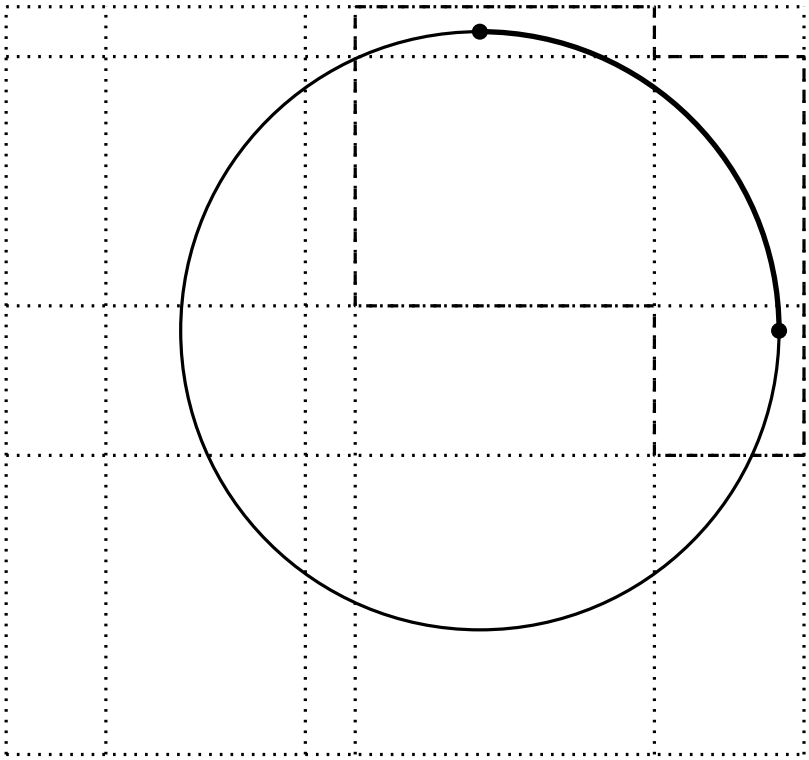
Collins's CAD solves any *geometric* decision problem that does not involve quantification over the integers in time doubly exponential in the problem size. This does not harm our results as we exclusively use this algorithm to solve constant size subproblems. Geometric is to be understood in the sense of Descartes and Fermat, that is, the geometry of objects that can be expressed with polynomial equations. In particular, it allows us to make the following computations in the real-RAM and bounded-degree ADT models:

1. Given a constant-degree univariate polynomial, count its real roots in $O(1)$ operations,
2. Sort $O(1)$ real numbers given implicitly as roots of some constant-degree univariate polynomials in $O(1)$ operations,
3. Given a point in the plane and an arrangement of a constant number of constant-degree polynomial planar curves, locate the point in the arrangement in $O(1)$ operations.

Instead of bounded-degree algebraic decision trees as the nonuniform model we could consider decision trees in which each decision involves a constant-size instance of the decision problem in the first-order theory of the reals. The depth of a bounded-degree algebraic decision tree simulating such a tree would only be blown up by a constant factor.

III

Algorithms



A

Solving k-SUM using Few Linear Queries

with Jean Cardinal and John Iacono

The k-SUM problem is given n input real numbers to determine whether any k of them sum to zero. The problem is of tremendous importance in the emerging field of complexity theory within P, and it is in particular open whether it admits an algorithm of complexity $O(n^c)$ with $c < \lceil \frac{k}{2} \rceil$. Inspired by an algorithm due to Meiser (1993), we show that there exist linear decision trees and algebraic computation trees of depth $O(n^3 \log^2 n)$ solving k-SUM. Furthermore, we show that there exists a randomized algorithm that runs in $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$ time, and performs $O(n^3 \log^2 n)$ linear queries on the input. Thus, we show that it is possible to have an algorithm with a runtime almost identical (up to the $+8$) to the best known algorithm but for the first time also with the number of queries on the input a polynomial that is independent of k. The $O(n^3 \log^2 n)$ bound on the number of linear queries is also a tighter bound than any known algorithm solving k-SUM, even allowing unlimited total time outside of the queries. By simultaneously achieving few queries to the input without significantly sacrificing runtime vis-à-vis known algorithms, we deepen the understanding of this canonical problem which is a cornerstone of complexity-within-P.

We also consider a range of tradeoffs between the number of terms involved in the queries and the depth of the decision tree. In particular, we prove that there exist $o(n)$-linear decision trees of depth $\tilde{O}(n^3)$ for the k-SUM problem.

A.1 Meiser Solves k-SUM

Our main contribution is an efficient implementation of an existing algorithm by Meiser [113] when applied to the k-SUM problem.

This section is divided into three subsections: §A.1.1 gives the outline of the algorithm and analyses its complexity in the linear decision tree model. §A.1.2 gives a second slightly more complex implementation and analysis of this algorithm in the word-RAM model. §A.1.3 gives a simple tweak that can be applied to those algorithms in order to reduce the complexity of the queries involved.

Missing details are found in §A.2.

A.1.1 Query Complexity

In this section and the next, we prove the following first result.

Contribution 1. *There exist linear decision trees of depth $O(n^3 \log^2 n)$ solving the k-SUM and the k-LDT problems. Furthermore, for the k-SUM problem there exists an $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$ Las Vegas algorithm in the word-RAM model expected to perform $O(n^3 \log^2 n)$ linear queries on the input. This algorithm also solves the k-LDT problem within the same time bounds provided the coefficients of the underlying linear equation are constant rational numbers.*

Algorithm outline

For a fixed set of hyperplanes $\mathcal{H}$ and given input vertex q in $\mathbb{R}^n$, Meiser's algorithm allows us to determine the cell of the arrangement $\mathcal{A}(\mathcal{H})$ that contains q in its interior (or that *is* q if q is a 0-cell of $\mathcal{A}(\mathcal{H})$), that is, the positions $\sigma(H, q) \in \{-, 0, +\}$ of q with respect to all hyperplanes $H \in \mathcal{H}$. In the k-SUM problem, the set $\mathcal{H}$ is the set of $\Theta(n^k)$ hyperplanes with equations of the form $x_{i_1} + x_{i_2} + \cdots + x_{i_k} = 0$. These equations can be modified accordingly for k-LDT.

We use Theorem 8.3 to design a prune and search algorithm for k-SUM as follows: (1) construct an ε-net $\mathcal{N}$, (2) compute the cell C of $\mathcal{A}(\mathcal{N})$ containing the input point q in its interior, (3) construct a simplex S inscribed in C and containing q in its interior, (4) recurse on the hyperplanes of $\mathcal{H}$ intersecting the interior of S.

Proceeding this way with a constant ε guarantees that at most a constant fraction ε of the hyperplanes remains after the pruning step, and thus the cumulative number of queries made to determine the enclosing cell at each step is $O(n^2 \log n \log |\mathcal{H}|)$ when done in a brute-force way. However, we still need to explain how to find a simplex S inscribed in C and containing q in its interior. This procedure corresponds to the well-known *bottom vertex decomposition* (or *triangulation*) of a hyperplane arrangement (see §6.4.1).

Finding a simplex

In order to simplify the exposition of the algorithm, we assume, without loss of generality, that the input numbers q_i all lie in the interval $[-1, 1]$. This assumption is justified by observing that we can normalize all the input numbers by the largest absolute value of a component of q. One can then see that every linear query on the normalized input can be implemented as a linear query on the original input. A similar transformation can be carried out for the k-LDT problem. This allows us to use bounding hyperplanes of equations $x_i = \pm 1, i \in [n]$. We denote by $\mathcal{B}$ this set of hyperplanes. Hence, if we choose a subset $\mathcal{N}$ of the hyperplanes, the input point is located in a bounded cell of the arrangement $\mathcal{A}(\mathcal{N} \cup \mathcal{B})$. Note that $|\mathcal{N} \cup \mathcal{B}| = O(|\mathcal{N}|)$ for all interesting values of ε.

We now explain how to construct S under this assumption. The algorithm can be sketched as follows. (Recall that $\sigma(H, p)$ denotes the relative position of p with respect to the hyperplane H.)

Algorithm 1 (Constructing S)**.**

input A point q in $[-1, 1]^n$, a set $\mathcal{I}$ of hyperplanes not containing q, and a set $\mathcal{E}$ of hyperplanes in general position containing q, such that the cell

$$C = \{\, p\colon\ \sigma(H, p) = \sigma(H, q) \text{ or } \sigma(H, p) = 0 \text{ for all } H \in (\mathcal{I} \cup \mathcal{E}) \,\}$$

is a bounded polytope. The value $\sigma(H, q)$ is known for all $H \in (\mathcal{I} \cup \mathcal{E})$.

output A simplex $S \in C$ that contains q in its interior (if it is not a point), and all vertices of which are vertices of C.

0. If $|\mathcal{E}| = n$, return q.

1. Determine a vertex ν of C.

2. Let q' be the projection of q along $\vec{\nu q}$ on the boundary of C. Compute $\mathcal{I}_\theta \subseteq \mathcal{I}$, the subset of hyperplanes in $\mathcal{I}$ containing q'. Compute $\mathcal{I}_\tau \subseteq \mathcal{I}_\theta$, a maximal subset of those hyperplanes such that $\mathcal{E}' = \mathcal{E} \cup \mathcal{I}_\tau$ is a set of hyperplanes in general position.

3. Recurse on q', $\mathcal{I}' = \mathcal{I} \setminus \mathcal{I}_\theta$, and $\mathcal{E}'$, and store the result in S'.

4. Return S, the convex hull of $S' \cup \{ \nu \}$.

Step **0** is the base case of the recursion: when there is only one point left, just return that point. This step uses no query.

We can solve step **1** by using linear programming with the known values of $\sigma(H, q)$ as linear constraints. We arbitrarily choose an objective function with a gradient non-orthogonal to all hyperplanes in $\mathcal{I}$ and look for the optimal solution. The optimal solution being a vertex of the arrangement, its coordinates are independent of q, and thus this step involves no query at all.

Step **2** prepares the recursive step by finding the hyperplanes containing q'. This can be implemented as a ray-shooting algorithm that performs a number of comparisons between projections of q on different hyperplanes of $\mathcal{I}$ without explicitly computing them. In §A.2.1, we prove that all such comparisons can be implemented using $O(|\mathcal{I}|)$ linear queries. Constructing $\mathcal{E}'$ can be done by solving systems of linear equations that do not involve q.

In step **3**, the input conditions are satisfied, that is, $q' \in [-1, 1]^n$, $\mathcal{I}'$ is a set of hyperplanes not containing q', $\mathcal{E}'$ is a set of hyperplanes in general position containing q', C' is a d-cell of C and is thus a bounded polytope. The value $\sigma(H, q')$ differs from $\sigma(H, q)$ only for hyperplanes that have been removed from $\mathcal{I}$, and for those $\sigma(H, q') = 0$, hence we know all necessary values $\sigma(H, q')$ in advance.

Since $|\mathcal{I}'| < |\mathcal{I}|$, $|\mathcal{E}'| > |\mathcal{E}|$, and $|\mathcal{I} \setminus \mathcal{I}'| - |\mathcal{E}' \setminus \mathcal{E}| \geq 0$, the complexity of the recursive call is no more than that of the parent call, and the maximal depth of the recursion is n. Thus, the total number of linear queries made to compute S is $O(n|\mathcal{I}|)$.

Hence given an input point $q \in [-1, 1]$, an arrangement of hyperplanes $\mathcal{A}(\mathcal{N})$, and the value of $\sigma(H, q)$ for all $H \in (\mathcal{N} \cup \mathcal{B})$, we can compute the desired simplex S by running Algorithm 1 on q, $\mathcal{I} = \{ H \in (\mathcal{N} \cup \mathcal{B}) \colon \sigma(H, q) \neq 0 \}$, and $\mathcal{E} \subseteq (\mathcal{N} \cup \mathcal{B}) \setminus \mathcal{I}$. This uses $O(n^3 \log n)$ linear queries. Figure A.1 illustrates a step of the algorithm.

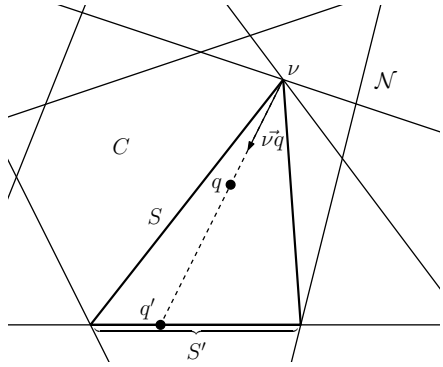


Figure A.1. Illustration of a step of Algorithm 1.

Assembling the pieces

Let us summarize the algorithm.

Algorithm 2.

input $q \in [-1, 1]^n$

1. Pick $O(n^2 \log n)$ hyperplanes of $\mathcal{H}$ at random and locate q in this arrangement. Call C the cell containing q.

2. Construct the simplex S containing q and inscribed in C, using Algorithm 1.

3. For every hyperplane of $\mathcal{H}$ containing S, output a solution.

4. Recurse on hyperplanes of $\mathcal{H}$ intersecting the interior of S.

The query complexity of step **1** is $O(n^2 \log n)$, and that of step **2** is $O(n^3 \log n)$. Steps **3** and **4** do not involve any query at all. The recursion depth is $O(\log |\mathcal{H}|)$, with $|\mathcal{H}| = O(n^k)$, hence the total query complexity of this algorithm is $O(n^3 \log^2 n)$. This proves the first part of Theorem 1.

We can also consider the overall complexity of the algorithm in the RAM model, that is, taking into account the steps that do not require any query, but for which we still have to process the set $\mathcal{H}$. Note that the complexity bottleneck of the algorithm are steps **3**-**4**, where we need to prune the list of hyperplanes according to their relative positions with respect to S. For this purpose, we simply maintain explicitly the list of all hyperplanes, starting

with the initial set corresponding to all k-tuples. Then the pruning step can be performed by looking at the position of each vertex of S relative to each hyperplane of $\mathcal{H}$. Because in our case hyperplanes have only k nonzero coefficients, this uses a number of integer arithmetic operations on $\tilde{O}(n)$ bits integers that is proportional to the number of vertices times the number of hyperplanes. (For the justification of the bound on the number of bits needed to represent vertices of the arrangement see §A.2.4.) Since we recurse on a fraction of the set, the overall complexity is $\tilde{O}(n^2|\mathcal{H}|) = \tilde{O}(n^{k+2})$. The next section is devoted to improving this running time.

A.1.2 Time Complexity

Proving the second part of Theorem 1 involves efficient implementations of the two most time-consuming steps of Algorithm 2. In order to efficiently implement the pruning step, we define an intermediate problem, that we call the *double k-SUM* problem.

Problem 15 (double k-SUM)**.** Given two vectors $\nu_1, \nu_2 \in [-1, 1]^n$, where the coordinates of ν_i can be written down as fractions whose numerator and denominator lie in the interval $[-M, M]$, enumerate all $i \in [n]^k$ such that

$$\left(\sum_{j=1}^{k} \nu_{1,i_j}\right)\left(\sum_{j=1}^{k} \nu_{2,i_j}\right) < 0.$$

In other words, we wish to list all hyperplanes of $\mathcal{H}$ intersecting the open line segment $\nu_1\nu_2$. We give an efficient output-sensitive algorithm for this problem.

Lemma A.1. *Double k-SUM can be solved in $O(n^{\lceil \frac{k}{2} \rceil} \log n \log M + Z)$ time in the word-RAM model, where Z is the size of the solution.*

Proof. If k is even, we consider all possible $\frac{k}{2}$-tuples of numbers in ν_1 and ν_2 and sort their sums in increasing order. This takes time $O(n^{\frac{k}{2}} \log n)$ and yields two permutations π_1 and π_2 of $[n^{\frac{k}{2}}]$. If k is odd, then we sort both the $\lceil \frac{k}{2} \rceil$-tuples and the $\lfloor \frac{k}{2} \rfloor$-tuples. For simplicity, we will only consider the even case in what follows. The odd case carries through.

We let $N = n^{\frac{k}{2}}$. For $i \in [N]$ and $m \in \{1, 2\}$, let $\Sigma_{m,i}$ be the sum of the $\frac{k}{2}$ components of the ith $\frac{k}{2}$-tuple in ν_m, in the order prescribed by π_m.

We now consider the two $N \times N$ matrices M_1 and M_2 giving all possible sums of two $\frac{k}{2}$-tuples, for both ν_1 with the ordering π_1 and ν_2 with the ordering π_2.

We first solve the k-SUM problem on ν_1, by finding the sign of all pairs $\Sigma_{1,i}+\Sigma_{1,j}$, $i,j \in [N]$. This can be done in time $O(N)$ by parsing the matrix M_1, just as in the standard k-SUM algorithm. We do the same with M_2.

The set of all indices $i,j \in [N]$ such that $\Sigma_{1,i}+\Sigma_{1,j}$ is positive forms a staircase in M_1. We sweep M_1 column by column in order of increasing $j \in [N]$, in such a way that the number of indices i such that $\Sigma_{1,i}+\Sigma_{1,j} > 0$ is growing. For each new such value i that is encountered during the sweep, we insert the corresponding $i' = \pi_2(\pi_1^{-1}(i))$ in a balanced binary search tree.

After each sweep step in M_1 — that is, after incrementing j and adding the set of new indices i' in the tree — we search the tree to identify all the indices i' such that $\Sigma_{2,i'}+\Sigma_{2,j'} < 0$, where $j' = \pi_2(\pi_1^{-1}(j))$. Since those indices form an interval in the ordering π_2 when restricted to the indices in the tree, we can search for the largest i'_0 such that $\Sigma_{2,i'_0} < -\Sigma_{2,j'}$ and retain all indices $i' \leq i'_0$ that are in the tree. If we denote by z the number of such indices, this can be done in $O(\log N + z) = O(\log n + z)$ time. Now all the pairs i', j' found in this way are such that $\Sigma_{1,i}+\Sigma_{1,j}$ is positive and $\Sigma_{2,i'}+\Sigma_{2,j'}$ is negative, hence we can output the corresponding k-tuples. To get all the pairs i', j' such that $\Sigma_{1,i}+\Sigma_{1,j}$ is negative and $\Sigma_{2,i'}+\Sigma_{2,j'}$ positive, we repeat the sweeping algorithm after swapping the roles of ν_1 and ν_2.

Every matching k-tuple is output exactly once, and every $\frac{k}{2}$-tuple is inserted at most once in the binary search tree. Hence the algorithm runs in the claimed time.

Note that we only manipulate rational numbers that are the sum of at most k rational numbers of size $O(\log M)$. □

Now observe that a hyperplane intersects the interior of a simplex if and only if it intersects the interior of one of its edges. Hence given a simplex S we can find all hyperplanes of $\mathcal{H}$ intersecting its interior by running the above algorithm $\binom{n}{2}$ times, once for each pair of vertices (ν_1, ν_2) of S, and take the union of the solutions. The overall running time for this implementation will therefore be $\tilde{O}(n^2(n^{\lceil \frac{k}{2} \rceil} \log M + Z))$, where Z is at most the number of

intersecting hyperplanes and M is to be determined later. This provides an implementation of the pruning step in Meiser's algorithm, that is, step **4** of Algorithm 2.

Corollary A.2. *In the word-RAM model, given a simplex S, we can compute the Z k-SUM hyperplanes intersecting its interior in $\tilde{O}(n^2(n^{\lceil \frac{k}{2} \rceil} \log M + Z))$ time, where $\log M$ is proportional to the number of bits necessary to represent S.*

In order to detect solutions in step **3** of Algorithm 2, we also need to be able to quickly solve the following problem.

Problem 16 (multiple k-SUM)**.** Given d points $\nu_1, \nu_2, \ldots, \nu_d \in \mathbb{R}^n$, where the coordinates of ν_i can be written down as fractions whose numerator and denominator lie in the interval $[-M, M]$, decide whether there exists a hyperplane with equation of the form $x_{i_1} + x_{i_2} + \cdots + x_{i_k} = 0$ containing all of them.

Here the standard k-SUM algorithm can be applied, taking advantage of the fact that the coordinates lie in a small discrete set.

Lemma A.3. *In the word-RAM model, k-SUM on n integers $\in [-V, V]$ can be solved in time $\tilde{O}(n^{\lceil \frac{k}{2} \rceil} \log V)$.*

Lemma A.4. *In the word-RAM model, multiple k-SUM can be solved in time $\tilde{O}(dn^{\lceil \frac{k}{2} \rceil + 2} \log M)$.*

Proof. Let $\mu_{i,j}$ and $\delta_{i,j}$ be the numerator and denominator of $\nu_{i,j}$ when written as an irreducible fraction. We define

$$\zeta_{i,j} = \nu_{i,j} \prod_{(i,j) \in [d] \times [n]} \delta_{i,j} = \frac{\mu_{i,j} \prod_{(i',j') \in [d] \times [n]} \delta_{i',j'}}{\delta_{i,j}}.$$

By definition $\zeta_{i,j}$ is an integer and its absolute value is bounded by $U = M^{n^2}$, that is, it can be represented using $O(n^2 \log M)$ bits. Moreover, if one of the hyperplanes contains the point $(\zeta_{i,1}, \zeta_{i,2}, \ldots, \zeta_{i,n})$, then it contains ν_i. Construct n integers of $O(dn^2 \log M)$ bits that can be written $\zeta_{1,j} + U, \zeta_{2,j} + U, \ldots, \zeta_{d,j} + U$ in base $2Uk + 1$. The answer to our decision problem is "yes" if and only if there exists k of those numbers whose sum is $kU, kU, \ldots, kU$.

We simply subtract the number $U, U, \ldots, U$ to all n input numbers to obtain a standard k-SUM instance on n integers of $O(dn^2 \log M)$ bits. □

We now have efficient implementations of steps **3** and **4** of Algorithm 2 and can proceed to the proof of the second part of Theorem 1.

Proof. The main idea consists in modifying the first iteration of Algorithm 2, by letting $\varepsilon = \Theta(n^{-\frac{k}{2}})$. Hence we pick a random subset $\mathcal{N}$ of $O(n^{k/2+2} \log n)$ hyperplanes in $\mathcal{H}$ and use this as an ε-net. This can be done efficiently, as shown in §A.2.3.

Next, we need to locate the input q in the arrangement induced by $\mathcal{N}$. This can be done by running Algorithm 2 on the set $\mathcal{N}$. From the previous considerations on Algorithm 2, the running time of this step is

$$O(n|\mathcal{N}|) = \tilde{O}(n^{k/2+4}),$$

and the number of queries is $O(n^3 \log^2 n)$.

Then, in order to prune the hyperplanes in $\mathcal{H}$, we have to compute a simplex S that does not intersect any hyperplane of $\mathcal{N}$. For this, we observe that the above call to Algorithm 2 involves computing a sequence of simplices for the successive pruning steps. We save the description of those simplices. Recall that there are $O(\log n)$ of them, all of them contain the input q and have vertices coinciding with vertices of the original arrangement $\mathcal{A}(\mathcal{H})$. In order to compute a simplex S avoiding all hyperplanes of $\mathcal{N}$, we can simply apply Algorithm 1 on the set of hyperplanes bounding the intersection of these simplices. The running time and number of queries for this step are bounded respectively by $n^{O(1)}$ and $O(n^2 \log n)$.

Note that the vertices of S are not vertices of $\mathcal{A}(\mathcal{H})$ anymore. However, their coordinates lie in a finite set (see §A.2.4)

Lemma A.5. *Vertices of S have rational coordinates whose fraction representations have their numerators and denominators absolute values bounded above by $C^{4n^5} n^{2n^5+n^3+\frac{n}{2}}$, where C is a constant.*

We now are in position to perform the pruning of the hyperplanes in $\mathcal{H}$ with respect to S. The number of remaining hyperplanes after the pruning is at most $\varepsilon n^k = O(n^{k/2})$. Hence from Corollary A.2, the pruning can be performed in time proportional to $\tilde{O}(n^{\lceil k/2 \rceil+7})$.

Similarly, we can detect any hyperplane of $\mathcal{H}$ containing S using the result of Lemma A.4 in time $\tilde{O}(n^{\lceil k/2 \rceil + 8})$. Note that those last two steps do not require any query.

Finally, it remains to detect any solution that may lie in the remaining set of hyperplanes of size $O(n^{k/2})$. We can again fall back on Algorithm 2, restricted to those hyperplanes. The running time is $\tilde{O}(n^{k/2+2})$, and the number of queries is still $O(n^3 \log^2 n)$.

Overall, the maximum running time of a step is $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$, while the number of queries is always bounded by $O(n^3 \log^2 n)$.

□

A.1.3 Query Size

In this section, we consider a simple blocking scheme that allows us to explore a tradeoff between the number of queries and the size of the queries.

Lemma A.6. *For any integer $b > 0$, an instance of the k-SUM problem on $n > b$ numbers can be split into $O(b^{k-1})$ instances on at most $k\lceil \frac{n}{b} \rceil$ numbers, so that every k-tuple forming a solution is found in exactly one of the subproblems. The transformation can be carried out in time $O(n \log n + b^{k-1})$.*

Proof. Given an instance on n numbers, we can sort them in $O(n \log n)$ time. Partition the sorted sequence into b consecutive blocks $B_1, B_2, \ldots, B_b$ of equal size. This partition can be associated with a partition of the real line into b intervals, say $I_1, I_2, \ldots, I_b$. Now consider the partition of $\mathbb{R}^k$ into grid cells defined by the kth power of the partition $I_1, I_2, \ldots, I_b$. The hyperplane of equation $x_1 + x_2 + \cdots + x_k = 0$ hits $O(b^{k-1})$ such grid cells. Each grid cell $I_{i_1} \times I_{i_2} \times \cdots \times I_{i_k}$ corresponds to a k-SUM problem on the numbers in the set $B_{i_1} \cup B_{i_2} \cup \ldots \cup B_{i_k}$ (note that the indices i_j need not be distinct). Hence each such instance has size at most $k\lceil \frac{n}{b} \rceil$. □

Combining Lemma A.6 and Theorem 1 directly yields our second contribution.

Contribution 2. *For any integer $b > 0$, there exists a $k\lceil \frac{n}{b} \rceil$-linear decision tree of depth $\tilde{O}(b^{k-4} n^3)$ solving the k-SUM problem. Moreover, this decision tree can be implemented as an $\tilde{O}(b^{\lfloor \frac{k}{2} \rfloor - 9} n^{\lceil \frac{k}{2} \rceil + 8})$ Las Vegas word-RAM algorithm.*

The following two corollaries are obtained by taking $b = \Theta(\text{polylog}(n))$, and $b = \Theta(n^\alpha)$, respectively

Corollary A.7. *There exists a $o(n)$-linear decision tree of depth $\tilde{O}(n^3)$ solving the k-SUM problem. This decision tree can be implemented as a $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$ Las Vegas word-RAM algorithm.*

Corollary A.8. *For any α such that $0 < \alpha < 1$, there exists a $O(n^{1-\alpha})$-linear decision tree of depth $\tilde{O}(n^{3+(k-4)\alpha})$ solving the k-SUM problem. This decision tree can be implemented as a $\tilde{O}(n^{(1+\alpha)\frac{k}{2}+8.5})$ Las Vegas word-RAM algorithm.*

Note that the latter query complexity improves on $\tilde{O}(n^{\frac{k}{2}})$ whenever $\alpha < \frac{k-6}{2k-8}$ and $k \geq 7$. By choosing $\alpha = \frac{k-6}{2k-8} - \frac{\beta}{k-4}$ we obtain $O(n^{1-\frac{k-6}{2k-8}+\frac{\beta}{k-4}})$-linear decision trees of depth $\tilde{O}(n^{\frac{k}{2}-\beta})$ for any $k \geq 7$. Hence for instance, we obtain $O(n^{\frac{3}{4}+\frac{\beta}{4}})$-linear decision trees of depth $\tilde{O}(n^{4-\beta})$ for the 8SUM problem.

A.2 Missing Details

This section regroups the missing details of the algorithms described in the previous one (§A.1).

In §A.2.1 we show how to keep the queries linear even though it is not evident at first sight that they are because of all the algebra generated by the recursive steps of the simplex construction. In §A.2.2 we show that our linear decision tree can be implemented in the same time on the algebraic computation tree model, even though some linear queries may involve all members of the input, and hence have superconstant cost. In §A.2.3 we explain how to efficiently implement uniform random sampling for the construction of large ε-nets. In §A.2.4 we bound the size of the numbers manipulated by the word-RAM implementation of our k-SUM algorithm.

A.2.1 Keeping Queries Linear in Algorithm 1

In Algorithm 1, we want to ensure that the queries we make in step **2** are linear and that the queries we will make in the recursion step remain linear too.

Lemma A.9. *Algorithm 1 can be implemented so that it uses $O(n|I|)$ linear queries.*

Proof. Let us first analyze what the queries of step **2** look like. In addition to the input point q we are given a vertex ν and we want to find the projection q' of q in direction $\vec{\nu q}$ on the hyperplanes of $\mathcal{I}_\theta$. Let the equation of H_i be $\Pi_i(x) = c_i + d_i \cdot x = 0$ where c_i is a scalar and d_i is a vector. The projection of q along $\vec{\nu q}$ on a hyperplane H_i can thus be written[1] $\rho(q, \nu, H_i) = \nu + \lambda_i \vec{\nu q}$ such that $\Pi_i(\nu + \lambda_i \vec{\nu q}) = c_i + d_i \cdot \nu + \lambda_i d_i \cdot \vec{\nu q} = 0$. Computing the closest hyperplane amounts to finding $\lambda_\theta = \min_{\lambda_i > 0} \lambda_i$. Since $\lambda_i = -\frac{c_i + d_i \cdot \nu}{d_i \cdot \vec{\nu q}}$ we can test whether $\lambda_i > 0$ using the linear query[2] $-\frac{d_i \cdot \vec{\nu q}}{c_i + d_i \cdot \nu} >^? 0$. Moreover, if $\lambda_i > 0$ and $\lambda_j > 0$ we can test whether $\lambda_i < \lambda_j$ using the linear query $\frac{d_i \cdot \vec{\nu q}}{c_i + d_i \cdot \nu} <^? \frac{d_j \cdot \vec{\nu q}}{c_j + d_j \cdot \nu}$. Step **2** can thus be achieved using $O(1)$ $(2k)$-linear queries per hyperplane of $\mathcal{N}$.

In step **4**, the recursive step is carried out on

$$q' = \nu + \lambda_\theta \vec{\nu q} = \nu - \frac{c_\theta + d_\theta \cdot \nu}{d_\theta \cdot \vec{\nu q}} \vec{\nu q},$$

hence, comparing λ'_i to 0 amounts to performing the query $-\frac{d_i \cdot \vec{\nu q'}}{c_i + d_i \cdot \nu'} >^? 0$, which is not linear in q. The same goes for comparing λ'_i to λ'_j with the query $\frac{d_i \cdot \vec{\nu q'}}{c_i + d_i \cdot \nu'} <^? \frac{d_j \cdot \vec{\nu q'}}{c_j + d_j \cdot \nu'}$.

However, we can multiply both sides of the inequality test by $d_\theta \vec{\nu q}$ to keep the queries linear as shown below. We must be careful to take into account the sign of the expression $d_\theta \vec{\nu q}$, this costs us one additional linear query.

This trick can be used at each step of the recursion. Let $q^{(0)} = q$, then we have

$$q^{(s+1)} = \nu^{(s)} - \frac{c_{\theta_s} + d_{\theta_s} \cdot \nu^{(s)}}{d_{\theta_s} \cdot \vec{\nu q}^{(s)}} \vec{\nu q}^{(s)}$$

and $(d_{\theta_s} \cdot \vec{\nu q}^{(s)}) q^{(s+1)}$ yields a vector whose components are linear in $q^{(s)}$. Hence, $(\prod_{k=0}^{s} d_{\theta_k} \cdot \vec{\nu q}^{(k)}) q^{(s+1)}$ yields a vector whose components are linear in q, and for all pairs of vectors d_i and $\nu^{(s+1)}$ we have that $(\prod_{k=0}^{s} d_{\theta_k} \cdot \vec{\nu q}^{(k)})(d_i \cdot \vec{\nu q}^{(s+1)})$ is linear in q.

Hence at the sth recursive step of the algorithm, we will perform at

[1] Note that we project from ν instead of q. We are allowed to do this since $\nu + \lambda_i \vec{\nu q} = q + (\lambda_i - 1)\vec{\nu q}$ and there is no hyperplane separating q from ν.

[2] Note that if $c_i + d_i \cdot \nu = 0$ then $\lambda_i = 0$, we can check this beforehand for free.

most $|\mathcal{N}|$ linear queries of the type

$$-\left(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)}\right) \frac{d_i \cdot \vec{\nu q}^{(s)}}{c_i + d_i \cdot \nu^{(s)}} \overset{?}{>} 0$$

$|\mathcal{N}| - 1$ linear queries of the type

$$\left(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)}\right) \frac{d_i \cdot \vec{\nu q}^{(s)}}{c_i + d_i \cdot \nu^{(s)}} \overset{?}{<} \left(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)}\right) \frac{d_j \cdot \vec{\nu q}^{(s)}}{c_j + d_j \cdot \nu^{(s)}}$$

and a single linear query of the type

$$d_{\theta_{s-1}} \cdot \vec{\nu q}^{(s-1)} \overset{?}{<} 0.$$

In order to detect all hyperplanes H_i such that $\lambda_i = \lambda_\theta$ we can afford to compute the query $f(q) > g(q)$ for all query $f(q) < g(q)$ that we issue, and vice versa.

Note that, without further analysis, the queries can become n-linear as soon as we enter the $\frac{n}{k}^{\text{th}}$ recursive step. □

A.2.2 Algebraic Computation Trees

The following theorem follows immediately from the analysis of the linearity of queries

Theorem A.10. *The algebraic computation tree complexity of k-LDT is $\tilde{O}(n^3)$.*

Proof. We go through each step of Algorithm 2. Indeed, each k-linear query of step **1** can be implemented as $O(k)$ arithmetic operations, so step **1** has complexity $O(|\mathcal{N}|)$. The construction of the simplex in step **2** must be handled carefully. What we need to show is that each n-linear query we use can be implemented using $O(k)$ arithmetic operations. It is not difficult to see from the expressions given in §A.2.1 that a constant number of arithmetic operations and dot products suffice to compute the queries. A dot product in this case involves a constant number of arithmetic operations because the d_i are such that they each have exactly k non-zero components. The only expression that involves a non-constant number of operations is the product $\prod_{k=0}^{s} d_{\theta_k} \cdot \vec{\nu q}^{(k)}$, but this is equivalent to $(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)})(d_{\theta_s} \cdot$

$\vec{\nu q}^{(s)}$) where the first factor has already been computed during a previous step and the second factor is of constant complexity. Since each query costs a constant number of arithmetic operations and branching operations, step **2** has complexity $O(n|\mathcal{N}|)$. Finally, steps **3** and **4** are free since they do not involve the input. The complexity of Algorithm 2 in this model is thus also $O(n^3 \log n \log |\mathcal{H}|)$. □

A.2.3 Uniform Random Sampling

Theorem 8.3 requires us to pick a sample of the hyperplanes uniformly at random. Actually the theorem is a little stronger; we can draw each element of $\mathcal{N}$ uniformly at random, only keeping distinct elements. This is not too difficult to achieve for k-LDT when the $\alpha_i, i \in [k]$ are all distinct: to pick a hyperplane of the form $\alpha_0 + \alpha_1 x_{i_1} + \alpha_2 x_{i_2} + \cdots + \alpha_k x_{i_k} = 0$ uniformly at random, we can draw each $i_j \in [n]$ independently and there are n^k possible outcomes. However, in the case of k-SUM, we only have $\binom{n}{k}$ distinct hyperplanes. A simple dynamic programming approach solves the problem for k-SUM. For k-LDT we can use the same approach, once for each class of equal α_i.

Lemma A.11. *Given $n \in \mathbb{N}$ and $(\alpha_0, \alpha_1, \ldots, \alpha_k) \in \mathbb{R}^{k+1}$, m independent uniform random draws of hyperplanes in $\mathbb{R}^n$ with equations of the form $\alpha_0 + \alpha_1 x_{i_1} + \alpha_2 x_{i_2} + \cdots + \alpha_k x_{i_k} = 0$ can be computed in time $O(mk^2 \log n)$ and preprocessing time $O(k^2 n)$.*

Proof. We want to pick an assignment $a = \{(\alpha_1, x_{i_1}), \ldots, (\alpha_k, x_{i_k})\}$ uniformly at random. Note that all x_i are distinct while the α_j can be equal.

Without loss of generality, suppose $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_k$. There is a bijection between assignments and lexicographically sorted k-tuples $((\alpha_1, x_{i_1}), \ldots, (\alpha_k, x_{i_k}))$.

Observe that x_{i_j} can be drawn independently of $x_{i_{j'}}$ whenever $\alpha_j \neq \alpha_{j'}$. Hence, it suffices to generate a lexicographically sorted $|\chi|$-tuple of x_i for each class χ of equal α_i.

Let $\omega(m, l)$ denote the number of lexicographically sorted l-tuples, where each element comes from a set of m distinct x_i. We have

$$\omega(m, l) = \begin{cases} 1 & \text{if } l = 0 \\ \sum_{i=1}^{m} \omega(i, l-1) & \text{otherwise.} \end{cases}$$

To pick such a tuple $(x_{i_1}, x_{i_2}, \ldots, x_{i_l})$ uniformly at random we choose $x_{i_l} = x_o$ with probability

$$P(x_{i_l} = x_o) = \begin{cases} 0 & \text{if } o > m \\ \frac{\omega(o,l-1)}{\omega(m,l)} & \text{otherwise} \end{cases}$$

that we append to a prefix $(l-1)$-tuple (apply the procedure recursively), whose elements come from a set of o symbols. If $l = 0$ we just return the empty tuple.

Obviously, the probability for a given l-tuple to be picked is equal to $\frac{1}{\omega(m,l)}$.

Let X denote the partiton of the α_i into equivalence classes, then the number of assignments is equal to $\prod_{\chi \in X} \omega(n, |\chi|)$. (Note that for k-SUM this is simply $\omega(n,k)$ since there is only a single class of equivalence.) For each equivalence class χ we draw independently a lexicographically sorted $|\chi|$-tuple on n symbols using the procedure above. This yields a given assignment with probability $\frac{1}{\prod_{\chi \in X} \omega(n,|\chi|)}$. Hence, this corresponds to a uniform random draw over the assignments.

It is a well known fact that $\omega(n,k) = \binom{n+k-1}{k-1}$, hence each number we manipulate fits in $O(k \log n)$ bits, that is, $O(k)$ words. Moreover $\omega(n,k) = \omega(n-1,k) + \omega(n-1,k-1)$ so each $\omega(m,l)$ can be computed using a single addition on numbers of $O(k)$ words.

For given n and k, there are at most nk values $\omega(m,l)$ to compute, and for a given k-LDT instance, it must be computed only once. One way to perform the random draws is to compute the cumulative distribution functions of the discrete distributions defined above, then to draw x_{i_l}, we use binary search to find a generated random integer of $O(k)$ words in the cumulative distribution function. Computing the values $\omega(m,l)$ and all cumulative distributions functions can be done as a preprocessing step in $O(k^2 n)$ time. Assuming the generation of a random sequence of words takes linear time, performing a random draw takes time $O(k^2 \log n)$.

□

A.2.4 Proof of Lemma A.5

Theorem A.12 (Cramer's rule)**.** *If a system of n linear equations for n unknowns, represented in matrix multiplication form $Ax = b$, has a unique*

solution $x = (x_1, x_2, \ldots, x_n)^T$ *then, for all* $i \in [n]$,

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where A_i *is* A *with the* i*th column replaced by the column vector* b*.*

Lemma A.13 (Meyer auf der Heide[114])**.** *The absolute value of the determinant of an* $n \times n$ *matrix* $M = M_{i=1\ldots n, j=1\ldots n}$ *with integer entries is an integer that is at most* $C^n n^{\frac{n}{2}}$*, where* C *is the maximum absolute value in* M*.*

Proof. The determinant of M must be an integer and is the volume of the hyperparalleliped spanned by the row vectors of M, hence

$$|\det(M)| \leq \prod_{i=1}^{n} \sqrt{\sum_{j=1}^{n} M_{i,j}^2} \leq \left(\sqrt{nC^2}\right)^n \leq C^n n^{\frac{n}{2}}.$$

□

Lemma A.14. *The determinant of an* $n \times n$ *matrix* $M = M_{i=1\ldots n, j=1\ldots n}$ *with rational entries can be represented as a fraction whose numerators and denominators absolute values are bounded above by* $(ND^{n-1})^n n^{\frac{n}{2}}$ *and* D^{n^2} *respectively, where* N *and* D *are respectively the maximum absolute value of a numerator and a denominator in* M*.*

Proof. Le $\delta_{i,j}$ denote the denominator of $M_{i,j}$. Multiply each row M_i of M by $\prod_j \delta_{i,j}$. Apply Lemma A.13. □

We can now proceed to the proof of Lemma A.5.

Proof. Coefficients of the hyperplanes of the arrangement are constant rational numbers, those can be changed to constant integers (because each hyperplane has at most k nonzero coefficients). Let C denote the maximum absolute value of those coefficients.

Because of Theorem A.12 and Lemma A.13, vertices of the arrangement have rational coordinates whose numerators and denominators absolute values are bounded above by $C^n n^{\frac{n}{2}}$.

Given simplices whose vertices are vertices of the arrangement, hyperplanes that define the faces of those simplices have rational coefficients whose numerators and denominators absolute values are bounded above by

$C^{2n^3} n^{n^3+\frac{n}{2}}$ by Theorem A.12 and Lemma A.14. (Note that some simplices might be not fully dimensional, but we can handle those by adding vertices with coordinates that are not much larger than that of already existing vertices).

By applying Theorem A.12 and Lemma A.14 again, we obtain that vertices of the arrangement of those new hyperplanes (and thus vertices of S) have rational coefficients whose numerators and denominators absolute values are bounded above by $C^{4n^5} n^{2n^5+n^3+\frac{n}{2}}$. □

B

Subquadratic Algorithms for Algebraic 3SUM

with Luis Barba, Jean Cardinal, John Iacono, Stefan Langerman, and Noam Solomon

The 3SUM problem asks if an input n-set of real numbers contains a triple whose sum is zero. We qualify such a triple as *degenerate* because the probability of finding one in a random input is zero. We consider the 3POL problem, an algebraic generalization of 3SUM where we replace the sum function by a constant-degree polynomial in three variables. The motivations are threefold. Raz, Sharir, and de Zeeuw gave an $O(n^{11/6})$ upper bound on the number of degenerate triples for the 3POL problem. We give algorithms for the corresponding problem of counting them. Grønlund and Pettie designed subquadratic algorithms for 3SUM. We prove that 3POL admits bounded-degree algebraic decision trees of depth $O(n^{12/7+\varepsilon})$, and we prove that 3POL can be solved in $O(n^2(\log\log n)^{3/2}/(\log n)^{1/2})$ time in the real-RAM model, generalizing their results. Finally, we shed light on the General Position Testing (GPT) problem: "Given n points in the plane, do three of them lie on a line?", a key problem in computational geometry: we show how to solve GPT in subquadratic time when the input points lie on a small number of constant-degree polynomial curves. Many other geometric degeneracy testing problems reduce to 3POL.

B.1 First Subquadratic Algorithms for 3POL

The results in this section have been published in Paper B. Our main contribution in this paper is the first subquadratic algorithm for the 3POL

problem (Problem 13).

This section is divided into four subsections: In §B.1.1, we design a bounded-degree algebraic decision tree of depth $O(n^{\frac{12}{7}+\epsilon})$ for explicit 3POL (Contribution 3), and in §B.1.2, we adapt this decision tree to run in time $O(n^2(\log\log n)^{\frac{3}{2}}/(\log n)^{\frac{1}{2}})$ in the real-RAM model (Contribution 4). In §B.1.3, we generalize the decision tree from §B.1.1 to work for 3POL with the same depth, up to constant factors (Contribution 5). Finally, in §B.1.4, we give a real-RAM implementation of this second decision tree to solve 3POL as fast as explicit 3POL, up to constant factors (Contribution 6).

Missing details are found in §B.2 and applications can be found in §B.3.

B.1.1 Nonuniform Algorithm for Explicit 3POL

We begin with the description of a nonuniform algorithm for explicit 3POL which we use later as a basis for other algorithms.

We recall the definition of the explicit 3POL problem.

Problem 14 (explicit 3POL)**.** Let $f \in \mathbb{R}[x, y]$ be a bivariate polynomial of constant degree, given three sets A, B, and C, each containing n real numbers, decide whether there exist $a \in A$, $b \in B$, and $c \in C$ such that $c = f(a, b)$.

We prove the following

Contribution 3. *Explicit 3POL can be solved in $O(n^{12/7+\varepsilon})$ time in the bounded-degree algebraic decision tree model.*

Idea We partition the sets A and B into small groups of consecutive elements. That way, we can divide the $A \times B$ grid into cells with the guarantee that each curve $c = f(x, y)$ intersects a small number of those cells. For each such curve and each cell it intersects, we search c among the values $f(a, b)$ for all (a, b) in a given intersected cell. We generalize Fredman's trick [77] — and how it is used in Grønlund and Pettie's paper [91] — to quickly obtain a sorted order on those values, which provides us a logarithmic search time for each cell. Below is a sketch of the algorithm.

Algorithm 3 (Nonuniform algorithm for explicit 3POL)**.**

input $A = \{a_1 < \cdots < a_n\}, B = \{b_1 < \cdots < b_n\}, C = \{c_1 < \cdots < c_n\} \subset \mathbb{R}$.

output *accept* if $\exists\,(a,b,c) \in A \times B \times C$ such that $c = f(a,b)$, *reject* otherwise.[1]

1. Partition the intervals $[a_1, a_n]$ and $[b_1, b_n]$ into blocks A_i^* and B_j^* such that $A_i = A \cap A_i^*$ and $B_j = B \cap B_j^*$ have size g.

2. Sort the sets $f(A_i \times B_j) = \{\, f(a,b) \colon (a,b) \in A_i \times B_j \,\}$ for all A_i, B_j. This is the only step that is nonuniform.

3. For each $c \in C$,

3.1. For each cell $A_i^* \times B_j^*$ intersected by the curve $c = f(x,y)$,

3.1.1. Binary search for c in the sorted set $f(A_i \times B_j)$. If c is found, *accept* and halt.

4. *reject* and halt.

Like in Grønlund and Pettie's $\tilde{O}(n^{3/2})$ decision tree for 3SUM [91], the key is to give an efficient implementation of step **2**.

$A \times B$ grid partitioning Let $A = \{\, a_1 < a_2 < \cdots < a_n \,\}$ and $B = \{\, b_1 < b_2 < \cdots < b_n \,\}$. For some positive integer g to be determined later, partition the interval $[a_1, a_n]$ into n/g blocks $A_1^*, A_2^*, \ldots, A_{n/g}^*$ such that each block contains g numbers in A. Do the same for the interval $[b_1, b_n]$ with the numbers in B and name the blocks of this partition $B_1^*, B_2^*, \ldots, B_{n/g}^*$. For the sake of simplicity, and without loss of generality, we assume here that g divides n. We continue to make this assumption in the following sections. To each of the $(n/g)^2$ pairs of blocks A_i^* and B_j^* corresponds a cell $A_i^* \times B_j^*$. By definition, each cell contains g^2 pairs in $A \times B$. For the sake of notation, we define $A_i = A \cap A_i^* = \{\, a_{i,1} < a_{i,2} < \cdots < a_{i,g} \,\}$ and $B_j = B \cap B_j^* = \{\, b_{j,1} < b_{j,2} < \cdots < b_{j,g} \,\}$. Figure B.1 depicts this construction.

The following two lemmas result from this construction:

Lemma B.1. *For a fixed value $c \in C$, the curve $c = f(x,y)$ intersects $O(\frac{n}{g})$ cells. Moreover, those cells can be found in $O(\frac{n}{g})$ time.*

Proof. Since f has constant degree, the curve $c = f(x,y)$ can be partitioned into a constant number of xy-monotone arcs. Split the curve into

[1] Note that it is easy to modify the algorithm to count or report the solutions. In the latter case, the algorithm becomes output sensitive.

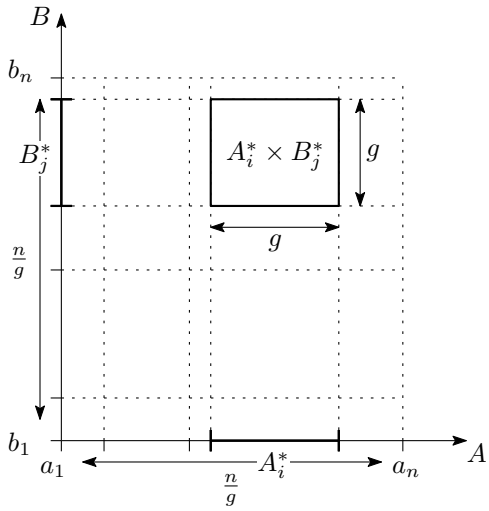


Figure B.1. The partitioning of $A \times B$. There are n/g columns A_i^*, n/g rows B_j^*, and $(n/g)^2$ cells $A_i^* \times B_j^*$. There are n^2 points in $A \times B$. Each column contains the ng points in $A_i \times B$, each row contains the ng points in $A \times B_j$, and each cell contains the g^2 points in $A_i \times B_j$.

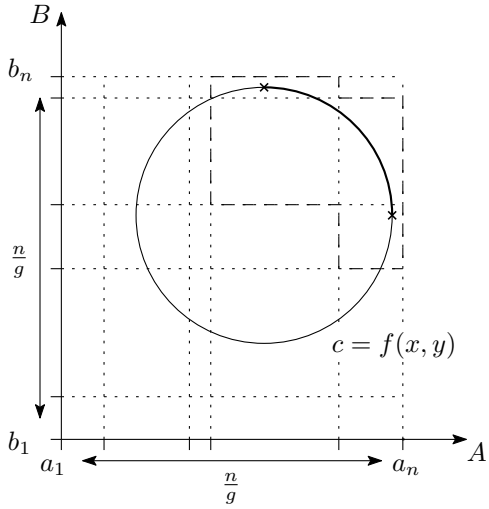


Figure B.2. An xy-monotone arc of the two-dimensional polynomial curve of equation $c = f(x, y)$. This arc intersects a staircase of at most $2\frac{n}{g} - 1$ cells in the grid. When f has constant degree, the defined curve can be partitioned into $O(1)$ such arcs.

x-monotone pieces, then each x-monotone piece into y-monotone arcs. The endpoints of the xy-monotone arcs are the intersections of $f(x,y) = c$ with its derivatives $f'_x(x,y) = 0$ and $f'_y(x,y) = 0$. By Bézout's theorem, there are $O(\deg(f)^2)$ such intersections and so $O(\deg(f)^2)$ xy-monotone arcs. Figure B.2 shows that each such arc intersects $O(\frac{n}{g})$ cells since the cells intersected by a xy-monotone arc form a staircase in the grid. This proves the first part of the lemma.

To prove the second part, notice that for each connected component of $c = f(x,y)$ intersecting at least one cell of the grid either: (1) it intersects a boundary cell of the grid, or (2) it is a (singular) point or contains vertical and horizontal tangency points.[2] The cells intersected by $c = f(x,y)$ are computed by exploring the grid from $O(\frac{n}{g})$ starting cells. Start with an empty set. Find and add all boundary cells containing a point of the curve. Finding those cells is achieved by solving the Tarski sentence $\exists x \exists y(c = f(x,y) \wedge x \in A_i^* \wedge y \in B_j^*)$, for each cell $A_i^* \times B_j^*$ on the boundary. This takes $O(\frac{n}{g})$ time. Find and add the cells containing singular points and tangency points of $c = f(x,y)$. Finding those cells is achieved by first finding the constant number of vertical and horizontal slabs $A_i^* \times \mathbb{R}$ and $\mathbb{R} \times B_j^*$ containing such points:

$$\exists x \exists y(c = f(x,y) \wedge (f'_x(x,y) = 0 \vee f'_y(x,y) = 0) \wedge x \in A_i^*),$$
$$\exists x \exists y(c = f(x,y) \wedge (f'_x(x,y) = 0 \vee f'_y(x,y) = 0) \wedge y \in B_j^*).$$

This takes $O(\frac{n}{g})$ time. Then for each pair of vertical and horizontal slab containing such a point, check that the cell at the intersection of the slabs also contains such a point:

$$\exists x \exists y(c = f(x,y) \wedge (f'_x(x,y) = 0 \vee f'_y(x,y) = 0) \wedge x \in A_i^* \wedge y \in B_j^*).$$

This takes $O(1)$ time. Note that we can always assume the constant-degree polynomials we manipulate are square-free, as making them square-free is trivial [159]: since $\mathbb{R}[x]$ and $\mathbb{R}[y]$ are unique factorization domains, let $Q = P/\gcd(P, P'_x; x)$ and $\mathrm{sf}(P) = Q/\gcd(P, P'_y; y)$, where $\gcd(P, Q; z)$ is the greatest common divisor of P and Q when viewed as polynomials in $R[z]$ where R is a unique factorization domain and $\mathrm{sf}(P)$ is the square-free part of P. The set now contains, for each component of each type, at least one

[2] Note that vertical and horizontal lines fall in both categories.

cell intersected by it. Initialize a list with the elements of the set. While the list is not empty, remove any cell from the list, add each of the eight neighbouring cells to the set and the list, if it contains a point of $c = f(x, y)$ — this can be checked with the same sentences as in the boundary case — and if it is not already in the set. This costs $O(1)$ per cell intersected. The set now contains all cells of the grid intersected by $c = f(x, y)$. □

Lemma B.2. *If the sets A, B, C can be preprocessed in $S_g(n)$ time so that, for any given cell $A_i^* \times B_j^*$ and any given $c \in C$, testing whether $c \in f(A_i \times B_j) = \{ f(a, b) \colon (a, b) \in A_i \times B_j \}$ can be done in $O(\log g)$ time, then, explicit 3POL can be solved in $S_g(n) + O(\frac{n^2}{g} \log g)$ time.*

Proof. We need $S_g(n)$ preprocessing time plus the time required to search each of the n numbers $c \in C$ in each of the $O(\frac{n}{g})$ cells intersected by $c = f(x, y)$. Each search costs $O(\log g)$ time. We can compute the cells intersected by $c = f(x, y)$ in $O(\frac{n}{g})$ time by Lemma B.1. □

Remark We do not give a $S_g(n)$-time real-RAM algorithm for preprocessing the input, but only a $S_g(n)$-depth bounded-degree ADT. In fact, this preprocessing step is the only nonuniform part of Algorithm 3. A real-RAM implementation of this step is given in §B.1.2.

Preprocessing All that is left to prove is that $S_g(n)$ is subquadratic for some choice of g. To achieve this we sort the points inside each cell using Fredman's trick [77]. Grønlund and Pettie [91] use this trick to sort the sets $A_i + B_j = \{ a + b \colon (a, b) \in A_i \times B_j \}$ with few comparisons: sort the set $D = (\cup_i [A_i - A_i]) \cup (\cup_j [B_j - B_j])$, where $A_i - A_i = \{ a - a' \colon (a, a') \in A_i \times A_i \}$ and $B_j - B_j = \{ b - b' \colon (b, b') \in B_j \times B_j \}$, using $O(n \log n + |D|)$ comparisons, then testing whether $a + b \leq a' + b'$ can be done using the free (already computed) comparison $a - a' \leq b' - b$. We use a generalization of this trick to sort the sets $f(A_i \times B_j)$. For each B_j, for each pair $(b, b') \in B_j \times B_j$, define the curve $\gamma_{b,b'} = \{ (x, y) \colon f(x, b) = f(y, b') \}$. Define the sets $\gamma^0_{b,b'} = \gamma_{b,b'}$, $\gamma^-_{b,b'} = \{ (x, y) \colon f(x, b) < f(y, b') \}$, and $\gamma^+_{b,b'} = \{ (x, y) \colon f(x, b) > f(y, b') \}$. The following lemma — illustrated by Figures B.3 and B.4 — follows by definition:

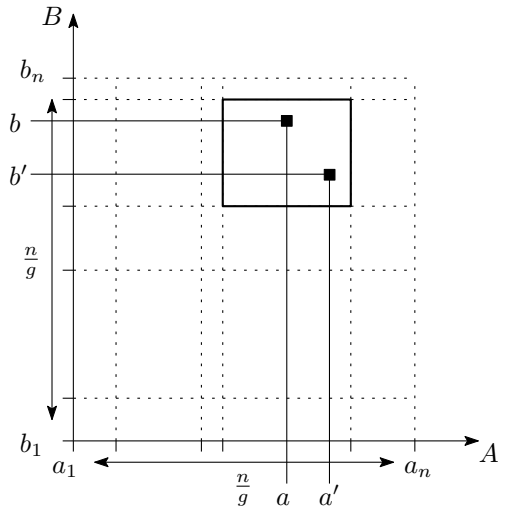


Figure B.3. For each cell, we sort the points it contains with comparisons. The points (a, b) and (a', b') are compared using the comparison $f(a, b) \leq f(a', b')$.

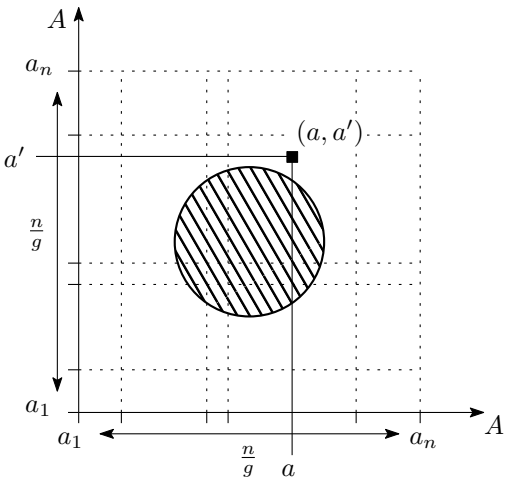


Figure B.4. The disk is the semi-algebraic set $\{(x, y) \colon f(x, b) \leq f(y, b')\}$. Here (a, a') lies outside this semi-algebraic set which implies that $f(a, b) > f(a', b')$.

Lemma B.3. *Given a cell $A_i^* \times B_j^*$ and two pairs $(a,b),(a',b') \in A_i \times B_j$, deciding whether $f(a,b) < f(a',b')$ (respectively $f(a,b) = f(a',b')$ and $f(a,b) > f(a',b')$) amounts to deciding whether the point (a,a') is contained in $\gamma^-_{b,b'}$ (respectively $\gamma^0_{b,b'}$ and $\gamma^+_{b,b'}$).*

There are $N \coloneqq \frac{n}{g} \cdot g^2 = ng$ pairs $(a,a') \in \cup_i[A_i \times A_i]$ and there are N pairs $(b,b') \in \cup_j[B_j \times B_j]$. Sorting the $f(A_i \times B_j)$ for all (A_i, B_j) amounts to solving the following problem:

Problem 17 (Polynomial Batch Range Searching)**.** Given N points and N polynomial curves in $\mathbb{R}^2$, locate each point with respect to each curve.

We can now refine the description of step **2** in Algorithm 3

Algorithm 4 (Sorting the $f(A_i \times B_j)$ with a nonuniform algorithm)**.**

input $A = \{ a_1 < a_2 < \cdots < a_n \}, B = \{ b_1 < b_2 < \cdots < b_n \} \subset \mathbb{R}$

output The sets $f(A_i \times B_j)$, sorted.

2.1. Locate each point $(a,a') \in \cup_i[A_i \times A_i]$ w.r.t. each $\gamma_{b,b'}, (b,b') \in \cup_j[B_j \times B_j]$.

2.2. Sort the sets $f(A_i \times B_j)$ using the information retrieved in step **2.1**.

Note that this algorithm is nonuniform: step **2.2** costs at least quadratic time in the real-RAM model, however, this step does not need to query the input at all, as all the information needed to sort is retrieved during step **2.1**. Step **2.2** incurs no cost in our nonuniform model.

To implement step **2.1**, we use a modified[3] version of the $N^{\frac{4}{3}}2^{O(\log^* N)}$ algorithm of Matoušek [109] for Hopcroft's problem. In § B.2.1, we prove the following upper bound:

Lemma B.4. *Polynomial Batch Range Searching can be solved in $O(N^{\frac{4}{3}+\varepsilon})$ time in the real-RAM model when the input curves are the $\gamma_{b,b'}$.*

Analysis Combining Lemma B.2 and Lemma B.4 yields a $O((ng)^{4/3+\varepsilon} + n^2 \log g/g)$-depth bounded-degree ADT for explicit 3POL. By optimizing over g, we get $g = \Theta(n^{2/7-\varepsilon})$, and the previous expression simplifies to $O(n^{12/7+\varepsilon})$, proving Contribution 3.

[3]The original algorithm relies on hierarchical cuttings which cannot be implemented in the bounded-degree ADT model.

B.1.2 Uniform Algorithm for Explicit 3POL

We again consider the explicit 3POL problem

Problem 14 (explicit 3POL)**.** Let $f \in \mathbb{R}[x, y]$ be a bivariate polynomial of constant degree, given three sets A, B, and C, each containing n real numbers, decide whether there exist $a \in A$, $b \in B$, and $c \in C$ such that $c = f(a, b)$.

We build on the nonuniform algorithm described in §B.1.1 to prove the following

Contribution 4. *Explicit 3POL can be solved in* $O(\frac{n^2(\log\log n)^{3/2}}{(\log n)^{1/2}})$ *time in the real-RAM model.*

We generalize again Grønlund and Pettie [91]. The algorithm we present is derived from the first subquadratic algorithm in their paper.

Idea We want the implementation of step **2** in Algorithm 3 to be uniform, because then, the whole algorithm is. We use the same partitioning scheme as before except we choose g to be much smaller. This allows to store all permutations on g^2 items in a lookup table, where g is chosen small enough to make the size of the lookup table $\Theta(n^\varepsilon)$. The preprocessing part of the previous algorithm is replaced by $g^2!$ calls to an algorithm that determines for which cells a given permutation gives the correct sorted order. This preprocessing step stores a constant-size[4] pointer from each cell to the corresponding permutation in the lookup table. Search can now be done efficiently: when searching a value c in $f(A_i \times B_j)$, retrieve the corresponding permutation on g^2 items from the lookup table, then perform binary search on the sorted order defined by that permutation. The sketch of the algorithm is exactly Algorithm 3. The only differences with respect to §B.1.1 are the choice of g and the implementation of step **2**.

$A \times B$ grid partitioning We use the same partitioning scheme as before, hence Lemma B.1 and Lemma B.2 hold. We just need to find a replacement for Lemma B.4.

[4] In the real-RAM and word-RAM models.

Preprocessing For their simple subquadratic 3SUM algorithm, Grønlund and Pettie [91] explain that for a permutation to give the correct sorted order for a cell, that permutation defines a *certificate* — a set of inequalities — that the cell must verify. They cleverly note — using Fredman's Trick [77] as in Chan [44] and Bremner et al. [33] — that the verification of a single certificate by all cells amounts to solving a red/blue point dominance reporting problem. We generalize their method. For each permutation $\pi\colon [g^2] \to [g]^2$, where $\pi = (\pi_r, \pi_c)$ is decomposed into row and column functions $\pi_r, \pi_c\colon [g^2] \to [g]$, we enumerate all cells $A_i^* \times B_j^*$ for which the following *certificate* holds:

$$f\Big(a_{i,\pi_r(1)}, b_{j,\pi_c(1)}\Big) \leq f\Big(a_{i,\pi_r(2)}, b_{j,\pi_c(2)}\Big) \leq \cdots \leq f\Big(a_{i,\pi_r(g^2)}, b_{j,\pi_c(g^2)}\Big).$$

Remark Since some entries may be equal, to make sure each cell corresponds to exactly one certificate, we replace $\leq$ symbols by choices of $g^2 - 1$ symbols in $\{=, <\}$. Each permutation π gets a certificate for each of those choices. This adds a 2^{g^2-1} factor to the number of certificates to test, which will eventually be negligible. Note that some of those 2^{g^2-1} certificates are equivalent. We need to skip some of them, as otherwise we might output some cells more than once, and then there will be no guarantee with respect to the output size. For example, the certificate $f(a_{i,9}, b_{j,5}) = f(a_{i,6}, b_{j,7}) < \cdots < f(a_{i,4}, b_{j,4})$ is equivalent to the certificate $f(a_{i,6}, b_{j,7}) = f(a_{i,9}, b_{j,5}) < \cdots < f(a_{i,4}, b_{j,4})$. Among equivalent certificates, we only consider the certificate whose permutation π precedes the others lexicographically. In the previous example, $((6,7),(9,5),\ldots,(4,4)) \prec ((9,5),(6,7),\ldots,(4,4))$ hence we would only process the second certificate. For the sake of simplicity, we will write inequality when we mean either strict inequality or equation, and "$\leq$" when we mean either "$<$" or "$=$".

Fredman's Trick This is where Fredman's Trick comes into play. By Lemma B.3, each inequality $f(a_{i,\pi_r(t)}, b_{j,\pi_c(t)}) \leq f(a_{i,\pi_r(t+1)}, b_{j,\pi_c(t+1)})$ of a certificate can be checked by computing the relative position of the point $(a_{i,\pi_r(t)}, a_{i,\pi_r(t+1)})$ with respect to the curve $\gamma_{b_{j,\pi_c(t)}, b_{j,\pi_c(t+1)}}$ in $\mathbb{R}^2$. For a given certificate, for each A_i and each B_j, define

$$p_i = \Big(\Big(a_{i,\pi_r(1)}, a_{i,\pi_r(2)}\Big), \ldots, \Big(a_{i,\pi_r(g^2-1)}, a_{i,\pi_r(g^2)}\Big)\Big),$$

$$q_j = \Big(f\Big(x, b_{j,\pi_c(1)}\Big) \leq f\Big(y, b_{j,\pi_c(2)}\Big), \ldots, f\Big(x, b_{j,\pi_c(g^2-1)}\Big) \leq f\Big(y, b_{j,\pi_c(g^2)}\Big)\Big).$$

A certificate is verified by a cell $A_i \times B_j$ if and only if, for all $t \in [g^2 - 1]$, the point $p_{i,t}$ verifies the inequality $q_{j,t}$. Enumerating all cells $A_i \times B_j$ for which the certificate holds therefore amounts to solving the following problem:

Problem 18 (Polynomial Dominance Reporting (PDR))**.** Given N k-tuples p_i of points in $\mathbb{R}^2$ and N k-tuples q_j of bivariate polynomial inequalities of degree at most Δ, output all pairs (p_i, q_j) where, for all $t \in [k]$, the point $p_{i,t}$ verifies the inequality $q_{j,t}$.

In § B.2.2, we explain how to solve PDR efficiently and prove the following:

Lemma B.5. *We can output all ℓ such pairs in time $2^{O(k)} N^{2-\frac{4}{\Delta^2+3\Delta+2}+\varepsilon} + O(\ell)$.*

We can now give a uniform implementation of step **2** in Algorithm 3:

Algorithm 5 (Sorting the $f(A_i \times B_j)$ with a uniform algorithm)**.**

input $A = \{ a_1 < a_2 < \cdots < a_n \}, B = \{ b_1 < b_2 < \cdots < b_n \} \subset \mathbb{R}$

output The sets $f(A_i \times B_j)$, sorted.

2.1. Initialize a table that will contain all $g^2!$ permutations on g^2 elements.

2.2. For each permutation $\pi \colon [g^2] \to [g]^2$,

2.2.1. Append permutation π to the table,

2.2.2. For each choice of $g^2 - 1$ symbols in $\{ =, < \}$,

2.2.2.1. If there is any "=" symbol that corresponds to a lexicographically decreasing pair of tuples of indices in π, skip this choice of symbols.

2.2.2.2. Solve the PDR instance associated with A, B, π and the choice of symbols.

2.2.2.3. For each output pair (i, j), store a pointer to the last entry in the table.

Analysis Plugging in $k = g^2 - 1$, $N = \frac{n}{g}$, iterating over all permutations ($\sum_\pi \ell = (n/g)^2$), and adding the binary search step we get that explicit 3POL can be solved in time

$$(g^2!)2^{g^2}2^{O(g^2)}(n/g)^{2-\frac{4}{\deg(f)^2+3\deg(f)+2}+\varepsilon} + O((n/g)^2) + O(n^2 \log g/g).$$

The first two terms correspond to the complexity of step **2** in Algorithm 3, and the last term corresponds to the complexity of step **3** in Algorithm 3. To get subquadratic time we can set $g = c_{\deg(f)}\sqrt{\log n/\log\log n}$, because then for some appropriate choice of the constant factor $c_{\deg(f)}$, $(g^2)!2^{g^2}2^{O(g^2)} = n^\delta$ where $\delta < 4/(\deg(f)^2 + 3\deg(f) + 2) - \varepsilon$, making the first term negligible. The complexity of the algorithm is dominated by $O(n^2 \log g/g) = O(n^2(\log\log n)^{3/2}/(\log n)^{1/2})$. This proves Theorem 4.

B.1.3 Nonuniform Algorithm for 3POL

In this section, we extend the nonuniform algorithm given for explicit 3POL in §B.1.1 to work for the more general 3POL problem.

We recall the definition of the 3POL problem.

Problem 13 (3POL)**.** Let $F \in \mathbb{R}[x, y, z]$ be a trivariate polynomial of constant degree, given three sets A, B, and C, each containing n real numbers, decide whether there exist $a \in A$, $b \in B$, and $c \in C$ such that $F(a, b, c) = 0$.

We prove the following:

Contribution 5. *3POL can be solved in $O(n^{12/7+\varepsilon})$ time in the bounded-degree algebraic decision tree model.*

Idea The idea is the same as for explicit 3POL. Partition the plane into $A_i^* \times B_j^*$ cells. Note that for a fixed $c \in C$, the curve $F(x, y, c)$ intersects $O(\frac{n}{g})$ cells $A_i^* \times B_j^*$. The algorithm is the following: (1) for each cell $A_i^* \times B_j^*$, sort the real roots of the $F(a, b, z) \in \mathbb{R}[z]$ taking the union over all $(a, b) \in A_i \times B_j$, (2) for each $c \in C$, for each cell $A_i^* \times B_j^*$ intersected by $F(x, y, c)$, binary search on the sorted order computed in step (1) to find c. Step (2) costs $O(n^2 \log g/g)$. It only remains to implement step (1) efficiently.

$A \times B$ partition We use the same partitioning scheme as before. Hence, counterparts of Lemma B.1 and Lemma B.2 hold

Lemma B.6. *For a fixed $c \in C$, the curve $F(x, y, c) = 0$ intersects $O(\frac{n}{g})$ cells. Moreover, those cells can be computed in $O(\frac{n}{g})$ time.*

Lemma B.7. *If the sets A, B, C can be preprocessed in $S_g(n)$ time so that, for any given cell $A_i^* \times B_j^*$ and any given $c \in C$, testing whether*

$c \in \{ z\colon \exists (a,b) \in A_i \times B_j$ such that $F(a,b,z) = 0 \}$ can be done in $O(\log g)$ time, then, 3POL can be solved in $S_g(n) + O(\frac{n^2}{g} \log g)$ time.

Interleavings Let $\mathcal{P} = (P_1, P_2, \ldots, P_m)$ be a tuple of m (nonzero) univariate polynomials. Let $\{ p_{i,1} < \cdots < p_{i,\Delta_i} \}$ be the set of real roots of P_i (*without* multiplicities). Let $I = ((i_1, j_1), \ldots, (i_\Delta, j_\Delta))$ be a tuple of pairs of positive integers. We say that $\mathcal{P}$ realizes I if and only if I is a permutation of $\{ (i,j)\colon i \in [m], j \in [\Delta_i] \}$, and for all $t \in [\Delta - 1]$, $p_{i_t,j_t} \leq p_{i_{t+1},j_{t+1}}$. When used in this context, we call I an *interleaving*. Note that (1) the first condition implies $\Delta = \sum_{i=1}^{m} \Delta_i$, (2) a tuple of polynomials realizes at least one interleaving, (3) a tuple of polynomials realizes more than one interleaving if some of the polynomials have common real roots. We denote by $\mathcal{I}(\mathcal{P})$ the set of interleavings realized by $\mathcal{P}$.

$A \times A$ (b,b')-partitions For a fixed pair $(b,b') \in B \times B$, we partition $\mathbb{R}^2$ into (b,b')-cells that encode equivalence classes. Each cell $\mathcal{C}$ is mapped to a unique interleaving I, and if we take any two points (a_1, a_1') and (a_2, a_2') inside $\mathcal{C}$, I is realized by both polynomial tuples $(F(a_1,b,z), F(a_1',b',z))$ and $(F(a_2,b,z), F(a_2',b',z))$. It is possible that a degenerate case arises where we cannot associate an interleaving to $\mathcal{C}$ because one of the polynomials is the zero polynomial. We can easily tackle these degeneracies, because, if any point (a,a') is contained in such a cell, we can immediately answer the instance with the affirmative. Identifying the interleaving associated with each cell of each (b,b')-partition, then locating each (a,a') inside each (b,b')-partition provides the answers to all questions of the form "Is the kth real root of $F(a,b,z)$ greater than the ℓth real root of $F(a',b',z)$?", for some $(a,b), (a',b') \in A_i \times B_j$. Those answers are all we need to binary search for c in the union of the real roots of the $F(a,b,z) \in \mathbb{R}[z]$ in time $O(\log g)$. Note again that in the nonuniform setting, we do not sort the roots explicitly, but we must be able to recover the order from the previous computation steps.

$\gamma_{b,b'}$ and δ_b curves We consider the set of interleavings $\mathcal{I}$ that the polynomial tuple $(F(x,b,z), F(y,b',z))$ realizes, where z is a variable, and x and y are parameters. We identify four types of event that can happen when the parameters x and y vary continuously (ignoring zero polynomials): (1) a real root of P_i and a real root of P_j that were previously distinct become equal,

for some P_i and P_j in $\mathcal{P}$, (2) a real root of P_i and a real root of P_j that were previously equal become distinct, for some P_i and P_j in $\mathcal{P}$, (3) a real root appears in one of the polynomials, and (4) a real root disappears in one of the polynomials. Note that many of those events can happen concurrently. By definition of an interleaving, those events are the only ones that can cause $\mathcal{I}$ to change.

To handle events of the types (1) and (2), we redefine the curves $\gamma_{b,b'}$ from §B.1.1:

$$\gamma_{b,b'} = \{\, (x, y) \colon \exists z \text{ such that } F(x, b, z) = F(y, b', z) = 0 \,\},$$

that is, $(a, a') \in \gamma_{b,b'}$ if and only if $F(a, b, z)$ and $F(a', b', z)$ have at least one common root.[5] Note that this curve is defined by the equation

$$\operatorname{res}(F(x, b, z), F(y, b, z); z) = 0,$$

that is, the set of pairs (x, y) for which the resultant (in z) of $F(x, b, z)$ and $F(y, b, z)$ vanishes. This resultant is a polynomial in $\mathbb{R}[x, y]$ of degree $O(\deg(F)^2)$ and can be computed in constant time [56]. The following lemma follows by continuity of the manipulated curve:

Lemma B.8. *Let (a_1, a_1') and (a_2, a_2') be two points in the plane such that there does not exist an interleaving that both $(F(a_1, b, z), F(a_1', b', z))$ and $(F(a_2, b, z), F(a_2', b', z))$ realize. Moreover, suppose that those two points belong to a connected region in the plane such that for any point (a, a') in that region, the number of real roots of $F(a, b, z)$ and $F(a', b', z)$ is fixed (and finite). Then the interior of any continuous path from (a_1, a_1') to (a_2, a_2') lying in this connected region must intersect $\gamma_{b,b'}$.*

Proof. Let I_1 be an interleaving realized by $(F(a_1, b, z), F(a_1', b', z))$ and let I_2 be an interleaving realized by $(F(a_2, b, z), F(a_2', b', z))$. Because the number of real roots of the polynomials $F(x, b, z)$ and $F(y, b', z)$ is fixed for any point (x, y) lying in the connected region, I_1 and I_2 differ by a nonzero number of swaps. Moreover, by contradiction, there is a swap that is common to every choice of I_1 and I_2. Since there is a common swap, for some $i, j \in [\deg(F)]$ and without loss of generality, the ith root of $F(a_1, b, z)$ is smaller than the jth root of $F(a_1', b', z)$ whereas the ith root of $F(a_2, b, z)$

[5] Note that Raz, Sharir, and de Zeeuw [133] use the same points and curves.

is larger than the jth root of $F(a_2', b', z)$. By continuity, on any continuous path from (a_1, a_1') and (a_2, a_2') there is a point (a, a') such that the ith root of $F(a, b, z)$ is equal to the jth root of $F(a', b', z)$. This point cannot be an endpoint of the path, hence, the interior of the path intersects $\gamma_{b,b'}$. □

The contrapositive states that, if there exists a continuous path from (a_1, a_1') to (a_2, a_2') whose interior does not intersect the curve $\gamma_{b,b'}$, then there exists an interleaving realized by both $(F(a_1, b, z), F(a_1', b', z))$ and $(F(a_2, b, z), F(a_2', b', z))$.

To handle events of the types (3) and (4), we define the curve

$$\delta_b = \{\, (x, z) \colon\ F(x, b, z) = 0 \,\},$$

which lies in the xz-plane.

Lemma B.9. *We can partition the x axis of the xz-plane into a constant number of intervals so that for each interval the number of real roots of $F(a, b, z)$ is fixed for all a in this interval.*

Proof. We partition the xz-plane into a constant number of vertical slabs and lines. The x coordinates of vertical tangency points and singular points of δ_b are the values a for which a real root of $F(a, b, z) = 0$ appears or disappears. The number of singular and vertical tangency points of δ_b is quadratic in $\deg(F)$. For each of those points, draw a vertical line that contains the point. Those vertical lines partition the xz-plane into slabs and lines. The number of vertical lines we draw is constant because the degree of F is constant. Figure B.5 depicts this drawing. The projection of the vertical lines on the x axis produce the desired partition (with roughly half of the intervals being singletons). Let us name those lines δ_b-lines for further reference. □

We can do a symmetric construction for $F(y, b', z)$ in the zy-plane and get horizontal $\delta_{b'}$-lines.

Lemma B.10. *We can partition the y axis of the zy-plane into a constant number of intervals so that for each interval the number of real roots of $F(a', b', z)$ is fixed for all a' in this interval.*

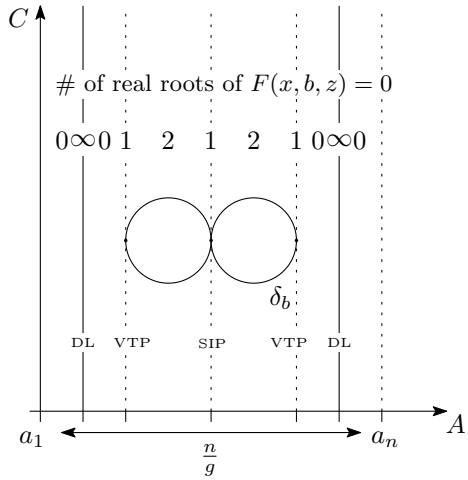


Figure B.5. The vertical tangency points (VTP), self-intersection points (SIP) and degenerate lines (DL) of δ_b partition the A axis into intervals. For all x of the same interval, the polynomial $F(x, b, z) \in \mathbb{R}[z]$ has a fixed number of real roots.

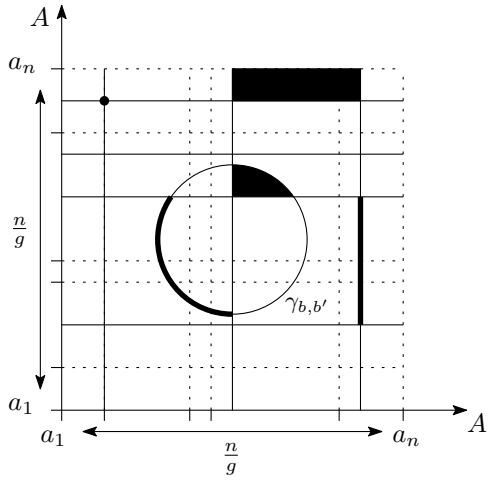


Figure B.6. Cells obtained after partitioning the plane using the curve $\gamma_{b,b'}$ and the δ_b and $\delta_{b'}$-lines. The five darkened regions highlight examples of (b, b')-cells.

Cells of the (b, b')-partition For a given $(b, b') \in B^2$, let $\Gamma_{b,b'}$ be the set containing the curve $\gamma_{b,b'}$, the vertical δ_b-lines and the horizontal $\delta_{b'}$-lines. The arrangement $\mathcal{A}(\Gamma_{b,b'})$ of those constant-degree polynomial curves partitions $\mathbb{R}^2$ into a constant-size set $\mathcal{C}(\Gamma_{b,b'})$ of (b, b')-*cells*. Let $P = \cup_{\gamma \in \Gamma_{b,b'}} \gamma$ and $\mathcal{C} = \emptyset$. Add all vertices of $\mathcal{A}(\Gamma_{b,b'})$ to $\mathcal{C}$. Add each connected component of $P \setminus \mathcal{C}$ to $\mathcal{C}$. Add each connected component of $\mathbb{R}^2 \setminus P$ to $\mathcal{C}$. Finally $\mathcal{C}(\Gamma_{b,b'}) = \mathcal{C}$. Those cells can be connected regions, pieces of the curve $\gamma_{b,b'}$, pieces of the δ_b- and $\delta_{b'}$-lines (vertical and horizontal line segments), and intersections and self-intersection of those curves (vertices). This partitioning scheme is depicted in figure B.6. By construction, all (b, b')-cells have the following *invariant property*

Definition 13. A (b, b')-cell has the invariant property if, for all points (a, a') in that cell, (1) the number of real roots of $F(a, b, z)$ is fixed, (2) the number of real roots of $F(a', b', z)$ is fixed, and (3) either, at least one of $F(a, b, z)$ or $F(a', b', z)$ is the zero polynomial, or the sorted order of the real roots of $F(a, b, z)$ and $F(a', b', z)$ is fixed, that is, $\mathcal{I}((F(a, b, z), F(a', b', z)))$ is fixed.

Lemma B.11. *All (b, b')-cells have the invariant property.*

Proof. First, (1) and (2) hold for all (b, b')-cells because of the partition induced by the δ_b-lines and the $\delta_{b'}$-lines. Second, (3) holds for all (b, b')-cells that are not contained in $\gamma_{b,b'}$ since (1) and (2) hold for those cells and because of the partition induced by $\gamma_{b,b'}$ (see Lemma B.8). Third, (3) holds for all (b, b')-cells that are both contained in $\gamma_{b,b'}$ and some δ_b- or $\delta_{b'}$-line because one of the associated polynomials must be the zero polynomial.

Finally, if a (b, b')-cell is contained in $\gamma_{b,b'}$ but not in any of the δ_b- or $\delta_{b'}$-lines, we make a simple observation. This cell has two distinct neighbouring connected regions lying on each of its sides. We just showed that those two neighbouring cells have the invariant property. The union of this piece of $\gamma_{b,b'}$ with its two neighbouring cells is a connected region as in Lemma B.8. Hence, the ordering of any two real roots cannot swap along the piece of $\gamma_{b,b'}$, as this would otherwise contradict Lemma B.8. Hence, (3) holds for those pieces of $\gamma_{b,b'}$. □

This lemma implies that, provided we compute in which (b, b')-cells each (a, a') point lies, we only need to probe a single point per (b, b')-cell to reveal

the sorted permutation associated with each cell of the $A \times B$ partition.

Preprocessing Locate all points $(a, a') \in A_i \times A_i$ for all A_i with respect to all $\gamma_{b,b'}$ curves, all vertical lines derived from δ_b and all horizontal lines derived from $\delta_{b'}$ for all $(b, b') \in B_j \times B_j$ for all B_j. As in §B.1.1, this can be done in a single batch using the algorithm described in §B.2.1, and the following generalization of Lemma B.13:

Lemma B.12. *Define*

$$\hat{\gamma}_{a,a'} = \{\, (x, y) \colon \ \operatorname{res}(F(a, x, z), F(a', y, z); z) = 0 \,\},$$
$$\hat{\delta}_a\text{-}lines = \{\, (x, y) \colon \ \operatorname{res}(F(a, x, z), F'_x(a, x, z); z) = 0 \,\},$$
$$\hat{\delta}_{a'}\text{-}lines = \{\, (x, y) \colon \ \operatorname{res}(F(a', y, z), F'_y(a', y, z); z) = 0 \,\},$$
$$\hat{\Gamma}_{a,a'} = \hat{\gamma}_{a,a'} \ \cup \ \hat{\delta}_a\text{-}lines \ \cup \ \hat{\delta}_{a'}\text{-}lines.$$

Locating (a, a') with respect to $\Gamma_{b,b'}$ amounts to locating (b, b') with respect to $\hat{\Gamma}_{a,a'}$.

This gives us the information needed for the binary search in step (2).

Analysis The analysis mirrors the explicit case (described immediately after Lemma B.4). Combining Lemma B.7 and Lemma B.4 yields a bounded-degree ADT of depth $O((ng)^{4/3+\varepsilon} + n^2 \log g/g)$ for 3POL. By optimizing over g, we get $g = \Theta(n^{2/7-\varepsilon})$, and the previous expression simplifies to $O(n^{12/7+\varepsilon})$, proving Theorem 5.

B.1.4 Uniform Algorithm for 3POL

In this section, we combine the uniform algorithm for explicit 3POL given in §B.1.2 with the nonuniform algorithm for 3POL given in §B.1.3 to obtain a uniform subquadratic algorithm for 3POL.

We recall the definition of the 3POL problem.

Problem 13 (3POL)**.** Let $F \in \mathbb{R}[x, y, z]$ be a trivariate polynomial of constant degree, given three sets A, B, and C, each containing n real numbers, decide whether there exist $a \in A$, $b \in B$, and $c \in C$ such that $F(a, b, c) = 0$.

We prove the following:

Contribution 6. *3POL can be solved in $O(\frac{n^2 (\log\log n)^{3/2}}{(\log n)^{1/2}})$ time in the real-RAM model.*

Idea In the uniform algorithm for explicit 3POL of §B.1.2, we partition the set $A \times B$ into very small sets $A_i \times B_j$, sort the sets $f(A_i \times B_j)$ using the dominance reporting algorithm of §B.2.2 then binary search on those sorted sets in order to find a matching c. Here we devise a similar scheme with the only difference that the sets to sort are the unions of the real roots of the univariate polynomials $F(a, b, z) \in \mathbb{R}[z]$ over all $(a, b) \in A_i \times B_j$. The main difficulty resides in implementing the equivalent of the certificates of §B.1.2 to reuse the dominance reporting algorithm of §B.2.2. We show how to implement those certificates using the $\gamma_{b,b'}$ and δ_b curves defined in §B.1.3.

$A \times B$ partition We use the same partitioning scheme as all previous algorithms, hence Lemma B.6 and Lemma B.7 hold. We apply the same certificate verification scheme as in §B.1.2, hence, the dominance reporting algorithm of §B.2.2 and the analysis in §B.1.2 still apply.

Preprocessing The preprocessing algorithm is essentially the same as Algorithm 5 with more complex certificates. We explain how to construct those new certificates. The first part of the explanation consists in generalizing the definition of a certificate. The rest of the explanation focuses on the implementation of the verification of those certificates via Polynomial Dominance Reporting.

The certificates For a fixed pair (a, b), $F(a, b, z) \in \mathbb{R}[z]$ is a polynomial in z of degree at most $\deg(F)$. Hence, $F(a, b, z)$ has at most $\deg(F)$ real roots. For each cell $A_i^* \times B_j^*$, let

$$A_i \times B_j = \{\, (a_{i,1}, b_{j,1}), (a_{i,1}, b_{j,2}), \ldots, (a_{i,2}, b_{j,1}), (a_{i,2}, b_{j,2}), \ldots, (a_{i,g}, b_{j,g}) \,\}.$$

Let $\rho\colon [g]^2 \to \{\, 0, 1, \ldots, \deg(F) \,\}$ be a function that maps a pair (k, l) to the number of real roots of $F(a_{i,k}, b_{j,l}, z)$. Let $\Sigma_\rho = \sum_{(i,j)\in[g]^2} \rho(i, j) \leq \deg(F)g^2$. Given a function ρ, let $\pi\colon [\Sigma_\rho] \to [g]^2 \times \{\, 0, 1, \ldots, \deg(F) \,\}$ be a permutation of the union of the real roots of all g^2 polynomials

$$F(a_{i,1}, b_{j,1}, z), F(a_{i,1}, b_{j,2}, z), \ldots, F(a_{i,2}, b_{j,1}, z), \ldots, F(a_{i,g}, b_{j,g}, z),$$

where the number of real roots of each polynomial is prescribed by ρ. Decompose $\pi = (\pi_r, \pi_c, \pi_s)$ into row, column and real root number functions

$\pi_r, \pi_c \colon [\Sigma_\rho] \to [g]$, and $\pi_s \colon [\Sigma_\rho] \to \{0, 1, \ldots, \deg(F)\}$. Let $\diamond(a, b, s)$ denote the sth real root of $F(a, b, z)$. To fix the permutation of the union of the real roots of all g^2 polynomials, we define the following interleaving certificate with $\Sigma_\rho - 1$ inequalities, for each possible function ρ and permutation π

$$\Phi_{\rho,\pi} = \diamond(a_{i,\pi_r(1)}, b_{j,\pi_c(1)}, \pi_s(1)) \leq \cdots \leq \diamond(a_{i,\pi_r(\Sigma_\rho)}, b_{j,\pi_c(\Sigma_\rho)}, \pi_s(\Sigma_\rho)).$$

To fix the number of real roots each of the g^2 polynomials can have, we define the following cardinality certificate for each function ρ

$$\Psi_\rho = \bigwedge_{(k,l)\in[g]^2} F(a_{i,k}, b_{j,l}, z) \text{ has } \rho(k, l) \text{ real roots.}$$

For each possible function ρ and permutation π we define the certificate $\Upsilon_{\rho,\pi} = \Psi_\rho \wedge \Phi_{\rho,\pi}$ that fixes both the number of real roots each polynomial has and the permutation of those real roots. The total number of certificates $\Upsilon_{\rho,\pi}$ is $\sum_{\rho \colon [g]^2 \to \{0,1,\ldots,\deg(F)\}} \Sigma_\rho!$ which is of the order of $(g^2)^{O(g^2)}$.

Finally, we need to handle the edge cases where a polynomial $F(a, b, z)$ is the zero polynomial. In that case, $F(a, b, z)$ cancels for all $z \in \mathbb{R}$. Hence, all planar curves $F(x, y, c) = 0$ go through (a, b) and we can immediately accept the 3POL instance. To capture those edge cases, we will check the following certificate before running the main algorithm

$$\aleph = \bigvee_{(k,l)\in[g]^2} F(a_{i,k}, b_{j,l}, z) \text{ is the zero polynomial.}$$

We can check if $\aleph$ holds for any cell $A_i \times B_j$ in $O(n \log n)$ time. For each $b \in B$ binary search for a $a \in A$ that lies on a vertical line component of δ_b.

If this certificate is verified we accept and halt. Otherwise we can safely run the main algorithm.

$A \times A$ (b, b')-partitions For each B_j and for each $(b, b') \in B_j^2$ compute a partition of the $A \times A$ grid according to the (b, b')-cells defined by $\Gamma_{b,b'}$ — see §B.1.3. For each (b, b')-cell of that partition, pick a sample point (a, a'), compute the interleaving $\mathcal{I}((F(a, b, z), F(a', b', z)))$. Store that information for future lookup. All this takes $O(ng)$ time.

PDR instance for Ψ_ρ For a fixed pair (a, b), suppose $F(a, b, z)$ has r real roots. Then a must lie in one of the open intervals or be one of the breaking points defined by the VTP, SIP and DL of δ_b that fixes the number of real roots of $F(a, b, z)$ to r. Hence Ψ_ρ can be rewritten as follows

$$\Psi_\rho = \bigwedge_{(k,l)\in[g]^2} \left(\bigvee_{[u,v]\in\mathcal{I}_{\rho(k,l)}} u < a_{i,k} < v \right) \bigvee \left(\bigvee_{w\in\mathcal{B}_{\rho(k,l)}} a_{i,k} = w \right)$$

where $\mathcal{I}_{\rho(k,l)}$ denotes the set of intervals fixing the number of real roots of $F(a_{i,k}, b_{j,l}, z)$ to $\rho(k, l)$, and $\mathcal{B}_{\rho(k,l)}$ denotes the set of breaking points fixing the number of real roots of $F(a_{i,k}, b_{j,l}, z)$ to $\rho(k, l)$.

The PDR algorithm can only check conjunctions of polynomial inequalities. However, we can transform Ψ_ρ into disjunctive normal form (DNF) by splitting the certificate into distinct branches, each consisting of a conjunction of polynomial inequalities. Since the number of intervals and breaking points considered above is constant for each pair (k, l), the number of branches to test is $2^{O(g^2)}$.

For each A_i we have thus a single vector of reals

$$p_i = (a_{i,1}, a_{i,1}, a_{i,2}, a_{i,2}, \ldots, a_{i,g}, a_{i,g}),$$

and for each B_j we have $2^{O(g^2)}$ vectors of linear inequalities

$$q_j = (x \bowtie_{u_{1,1}} u_{1,1}, x \bowtie_{v_{1,1}} v_{1,1}, x \bowtie_{u_{1,2}} u_{1,2}, x \bowtie_{v_{1,2}} v_{1,2}, \ldots \\ \ldots, x \bowtie_{u_{g,g}} u_{g,g}, x \bowtie_{v_{g,g}} v_{g,g},),$$

where each $(\bowtie_{u_{k,l}}, u_{k,l}, \bowtie_{v_{k,l}}, v_{k,l})$ is an element of

$$\{\, (>, u, <, v) \colon\ (u, v) \in \mathcal{I}_{\rho(k,l)} \,\} \cup \{\, (=, w, =, w) \colon\ w \in \mathcal{B}_{\rho(k,l)} \,\}.$$

For a fixed function ρ, the sets of vectors p_i and q_j is a valid PDR instance of size $N = (n/g) \cdot 2^{O(g)} = n2^{O(g)}$ and with parameter $k = 2g^2$ that will output all cells $A_i^* \times B_j^*$ such that $F(a_{i,k}, b_{j,l}, z) \in \mathbb{R}[z]$ has exactly $\rho(k, l)$ real roots for all $(a_{i,k}, a_{j,l}) \in A_i \times B_j$.

PDR instance for $\Phi_{\rho,\pi}$ For fixed pairs (a, b) and (a', b'), suppose the s-th real root of $F(a, b, z)$ is smaller or equal to the q-th real root of $F(a, b, z)$.

Then, (a, a') must lie in a (b, b')-cell that orders the s-th root of $F(x, b, z)$ before the q-th root of $F(y, b', z)$ for all points (x, y) in that cell.

Hence $\Phi_{\rho,\pi}$ can be rewritten as follows

$$\Phi_{\rho,\pi} = \bigwedge_{t\in[\Sigma_\rho-1]} \bigvee_{C\in\mathcal{C}_{\rho,\pi,t}} (a_{i,\pi_r(t)}, a_{i,\pi_r(t+1)}) \in C$$

where $\mathcal{C}_{\rho,\pi,t}$ denotes the set of (b, b')-cells fixing to $\rho(\pi_r(t), \pi_c(t))$ the number of real roots of $F(a_{i,\pi_r(t)}, b_{j,\pi_c(t)}, z)$, fixing to $\rho(\pi_r(t+1), \pi_c(t+1))$ the number of real roots of $F(a_{i,\pi_r(t+1)}, b_{j,\pi_c(t+1)}, z)$, and ordering the $\pi_s(t)$-th root of $F(a_{i,\pi_r(t)}, b_{j,\pi_c(t)}, z)$ before the $\pi_s(t+1)$-th root of $F(a_{i,\pi_r(t+1)}, b_{j,\pi_c(t+1)}, z)$.

The PDR algorithm can only check conjunctions of polynomial inequalities. However, we can transform $\Phi_{\rho,\pi}$ in DNF as we did for Ψ_ρ. Again the number of cells considered above is constant for each t, the description of each cell is constant, hence, the number of branches to test is $2^{O(g^2)}$.

For each A_i we have thus a single vector of 2-dimensional points

$$p_i = (\underbrace{\ldots, (a_{i,\pi_r(1)}, a_{i,\pi_r(2)}), \ldots}_{\omega}, \ldots, \underbrace{\ldots, (a_{i,\pi_r(\Sigma_\rho-1)}, a_{i,\pi_r(\Sigma_\rho)}), \ldots}_{\omega}),$$

where ω is the size of the largest description of a (b, b')-cell C, and for each B_j we have $2^{O(g^2)}$ vectors of polynomial inequalities,

$$q_j = (\underbrace{\ldots, h_{1,\vartheta}(x, y) \bowtie_{1,\vartheta} 0, \ldots}_{\vartheta\in[\omega]}, \ldots, \underbrace{\ldots, h_{\Sigma_\rho-1,\vartheta}(x, y) \bowtie_{\Sigma_\rho-1,\vartheta} 0 \ldots}_{\vartheta\in[\omega]}),$$

where each $(\ldots, h_{t,\vartheta}(x, y) \bowtie_{t,\vartheta} 0, \ldots)$ is an element of $\{\, \mathrm{desc}(C) \colon\ C \in \mathcal{C}_{\rho,\pi,t} \,\}$, where $\mathrm{desc}(C)$ is the description of the cell C given as a certificate of belonging to C in the form of a Tarski sentence. The description of each (b, b')-cell is padded with its last component so that it has length ω.

For a fixed function ρ, for a fixed function π, the sets of vectors p_i and q_j is a valid PDR instance of size $N = n2^{O(g)}$ and with parameter $k = \Theta(g^2)$ that will output all cells $A_i^* \times B_j^*$ such that the number of real roots of $F(a_{i,\pi_r(t)}, b_{j,\pi_c(t)}, z)$ is $\rho(\pi_r(t), \pi_c(t))$, the number of real roots of $F(a_{i,\pi_r(t+1)}, b_{j,\pi_c(t+1)}, z)$ is
$\rho(\pi_r(t+1), \pi_c(t+1))$, and the $\pi_s(t)$-th root of $F(a_{i,\pi_r(t)}, b_{j,\pi_c(t)}, z)$ comes before the $\pi_s(t+1)$-th root of $F(a_{i,\pi_r(t+1)}, b_{j,\pi_c(t+1)}, z)$, for all $t \in [\Sigma_\rho - 1]$.

PDR instance for $\Upsilon_{\rho,\pi}$ We can combine the certificates given above for Ψ_ρ and $\Phi_{\rho,\pi}$ to obtain the ones for $\Upsilon_{\rho,\pi}$: concatenate the p_i and q_j together (add a dummy y variable for the p_i and q_j of Ψ_ρ). For a fixed function ρ, for a fixed function π, the sets of vectors p_i and q_j is a valid PDR instance of size $N = n2^{O(g)}$ and with parameter $k = \Theta(g^2)$ that will output all cells $A_i^* \times B_j^*$ such that $F(a_{i,k}, b_{j,l}, z) \in \mathbb{R}[z]$ has exactly $\rho(k,l)$ real roots for all $(a_{i,k}, a_{j,l}) \in A_i \times B_j$, and the $\pi_s(t)$-th root of $F(a_{i,\pi_r(t)}, b_{j,\pi_c(t)}, z)$ comes before the $\pi_s(t+1)$-th root of $F(a_{i,\pi_r(t+1)}, b_{j,\pi_c(t+1)}, z)$ for all $t \in [\Sigma_\rho - 1]$. The rest of the analysis in §B.1.2 applies. This proves Theorem 6.

B.2 Subproblems

In §B.2.1 we prove Lemma B.4 necessary for the implementation of our nonuniform algorithms. In §B.2.2 we prove Lemma B.5 necessary for the implementation of our uniform algorithms.

B.2.1 Polynomial Batch Range Searching

In this section we present a uniform algorithm that computes the relative position of M points with respect to N $\gamma_{b,b'}$ curves. We call such a problem an (M,N)-problem. When $M = N$ the complexity of the algorithm is $O(N^{4/3+\varepsilon})$. The algorithm gives the output in "concise form": it outputs a set of $(\Pi_\alpha, \Gamma_\beta, \sigma)$ triples where Π_α is a subset of input points, Γ_β is a subset of input curves, and $\sigma \in \{-, 0, +\}$ indicates the relative position of all points in Π_α with respect to all curves in Γ_β. Note that if one is only interested in incident point-curve pairs, the algorithm can explicitly report all of them in $O(N^{4/3+\varepsilon})$ time, because there are at most $O(N^{4/3})$ such pairs and because they can easily be filtered from the output.

Tools The proof of Lemma B.4 involves stantard computational geometry tools: vertical decomposition of an arrangement of polynomial curves (see §**??**), ε-nets (see §8.1.3) and cuttings (see §8.1.5). cand derandomization.

Proof of Lemma B.4. Fix some constant $r \geq 2$. If $M < r^2$ or $N < r$, solve by brute-force in $O(M+N)$ time. Otherwise, consider the range space defined by $\gamma_{b,b'}$ curves and y-axis aligned trapezoidal patches whose top and bottom sides are pieces of $\gamma_{b,b'}$ curves. This range space has constant VC-dimension. Compute an $\frac{1}{r}$-net of size $O(r \log r)$ for the input curves with

respect to this range space. Compute the vertical decomposition Ξ of the arrangement of this $\frac{1}{r}$-net. This decomposition is a $\frac{1}{r}$-cutting: it partitions $\mathbb{R}^2$ into $O(r^2 \log^2 r)$ cells of constant complexity each of which intersects at most $\frac{N}{r}$ input curves. Note that some of those cells are degenerate trapezoidal patches. The decomposition contains vertices, line segments, and curve segments as cells, each of which could contain input points and be intersected or contained by an input curve. Denote by Π_C the set of points contained in the cell $C \in \Xi$. Partition each Π_C into $\left\lceil \frac{|\Pi_C|}{Mr^{-2}} \right\rceil$ disjoint subsets of size at most $\frac{M}{r^2}$. All of this can be done in $O(M+N)$ time. The last step consists in solving $O(r^2 \log^2 r)$ $(\frac{M}{r^2}, \frac{N}{r})$-problems, that is, solving the problem recursively for the points and curves intersecting each cell. Each recursive call is done by swapping the roles of the points and curves using a form of duality to be described below. The whole algorithm can be formally described as follows,

Algorithm 6 (Polynomial Batch Range Searching)**.**

input A set Π of M points (a, a'), A set Γ of N curves $\gamma_{b,b'}$.

output A set of triples $(\Pi_\alpha, \Gamma_\beta, \sigma)$ covering $\Pi \times \Gamma$ such that, for any triple $(\Pi_\alpha, \Gamma_\beta, \sigma)$, for all points (a, a') in Π_α and all curves γ in Γ_β, $(a, a') \in \gamma^\sigma$.

0. If $M < r^2$ or $N < r$, solve the problem by brute force in $O(M+N)$ time.

1. Compute an $\frac{1}{r}$-net of size $O(r \log r)$ for the input curves.

2. Compute the vertical decomposition Ξ of the arrangement of this $\frac{1}{r}$-net.

3. Denote by Π_C the set of points contained in the cell $C \in \Xi$. Partition each Π_C into $\left\lceil \frac{|\Pi_C|}{Mr^{-2}} \right\rceil$ disjoint subsets $\Pi_{C,i}$ of size at most $\frac{M}{r^2}$.

4. For each cell C of the vertical decomposition,

4.1. For each subset $\Pi_{C,i}$ of points contained in that cell,

4.1.1. Solve an $(\frac{N}{r}, \frac{M}{r^2})$-problem on the curves intersecting that cell and the points in $\Pi_{C,i}$, swapping the roles of lines and curves via duality.

4.2. For each curve γ not intersecting C,

4.2.1. Compute the location $\sigma_{C,\gamma}$ of any point in C with respect to γ.

4.3. Output $(\{\,\gamma\colon\ \sigma_{C,\gamma} = -\,\}, \Pi_C, -)$.

4.4. Output $(\{\,\gamma\colon\ \sigma_{C,\gamma} = +\,\}, \Pi_C, +)$.

4.5. Output $(\{\,\gamma\colon\ \sigma_{C,\gamma} = 0\,\}, \Pi_C, 0)$.

Correctness We want to locate each point with respect to each curve. When considering a curve-cell pair, there are two cases: either the curve intersects the cell, or it does not. For the first case we locate each point in the cell with respect to the curve in one of the recursive steps. For the second case, the relative position of all points in the cell with respect to the curve is the same, it suffices thus to locate one of those points with respect to the curve to get the location of all the points in $O(1)$ time. Each recursive call divides M and N by some constant strictly greater than one, hence, the base case is reached in each of the paths of the recursion tree and the algorithm always terminates.

Analysis For c_1 some constant and bounding $c_1 r^2 \log^2 r$ above by $c_2 r^{2+\varepsilon}$ for some large enough constant c_2, the complexity $T(M,N)$ of an (M,N)-problem is thus

$$T(M,N) \leq c_2 r^{2+\varepsilon}\, T\left(\frac{M}{r^2}, \frac{N}{r}\right) + O(M+N).$$

The complexity $T(N,M)$ of a (N,M)-problem is the same as the complexity $T(M,N)$ of an (M,N)-problem by the following point-curve duality result whose proof is straightforward

Lemma B.13. *Define*

$$\hat{\gamma}_{a,a'} = \{\,(x,y)\colon\ f(a,x) = f(a',y)\,\},$$

then, locating (a,a') with respect to $\gamma_{b,b'}$ amounts to locating (b,b') with respect to $\hat{\gamma}_{a,a'}$.

By doing alternately one step in the primal with the points (a,a') and the curves $\gamma_{b,b'}$, then a second step with the dual points (b,b') and the dual

curves $\hat{\gamma}_{a,a'}$, we get the following recurrence

$$T(M,N) \leq c_2^2 r^{4+\varepsilon}\, T\left(\frac{M}{r^3}, \frac{N}{r^3}\right) + c_2 r^{2+\varepsilon}\, O\left(\frac{M}{r^2} + \frac{N}{r}\right) + O(M+N)$$
$$\leq c_2^2 r^{4+\varepsilon}\, T\left(\frac{M}{r^3}, \frac{N}{r^3}\right) + O(M+N)$$

Hence, for some large enough constant c_3,

$$T(N,N) = T(N) \leq c_3 r^{4+\varepsilon}\, T\left(\frac{N}{r^3}\right) + O(N)$$
$$\leq O\left(N^{\log_{r^3}(c_3 r^{4+\varepsilon})}\right)$$
$$\leq O(N^{\frac{4}{3}+\varepsilon}).$$

□

B.2.2 Polynomial Dominance Reporting

We combine a standard dominance reporting algorithm [127] with Matoušek's algorithm [109] to prove Lemma B.5. For a pair of blue and red points in $\mathbb{R}^k$, we say that the blue point *dominates* the red point if for all indices $i \in [k]$ the ith coordinate of the blue point is greater or equal to the ith coordinate of the red point. The standard algorithm in [127] solves the following problem:

Problem 19. Given N blue and M red points in $\mathbb{R}^k$, report all bichromatic dominating pairs.

Our problem is significantly more complicated and general. Instead of blue points we have blue k-tuples p_i of 2-dimensional points, instead of red points we have red k-tuples q_j of bivariate polynomial inequalities, and we want to report all bichromatic pairs (p_i, q_j) such that, for all $t \in [k]$, the point $p_{i,t}$ verifies the inequality $q_{j,t}$. The standard algorithm essentially works by a combination of divide and conquer and prune and search, using a one-dimensional cutting (median selection) to split a problem into subproblems. We generalize the standard algorithm by using higher dimensional cuttings, in a way similar to Matoušek's algorithm [109]. For the analysis, we generalize Chan's analysis of the standard algorithm when k is not constant [44].

Proof of Lemma B.5. We use the Veronese embedding [95, 96]. Since the polynomials have constant degree, we can trade polynomial inequalities for linear inequalities by lifting to a space of higher — but constant — dimension. The degree of each polynomial is at most Δ. There are exactly $d = \binom{\Delta+2}{2} - 1$ different bivariate monomials of degree at most Δ.[6] To each monomial we associate a variable in $\mathbb{R}^d$. By this association, points in the plane are mapped to points in $\mathbb{R}^d$ and bivariate polynomial inequalities are mapped to d-variate linear inequalities.

By abuse of notation, let p_i denote the tuple p_i where each 2-dimensional point has been replaced by its d-dimensional counterpart, and let q_i denote the tuple q_i where each bivariate polynomial inequality has been replaced by its d-variate linear counterpart. We have N k-tuples p_i and M k-tuples q_j. The algorithm checks each of the k components of the tuples in turn and can be described recursively as follows for some positive integer $r > 1$:

Algorithm 7 (Polynomial Dominance Reporting)**.**

input N k-tuples p_i of d-dimensional points, M k-tuples q_j of d-variate linear inequalities.

output All (p_i, q_j) pairs such that, for all $t \in [k]$, the point $p_{i,t}$ verifies the inequality $q_{j,t}$.

1. If $k = 0$, then output all pairs (p_i, q_j) and halt.

2. If $N < r^d$ or $M < r$, solve the problem by brute force in $O((N+M)k)$ time.

3. We now only consider the kth component of each input k-tuple and call these *active* components. To each active d-variate linear inequality corresponds a defining hyperplane in $\mathbb{R}^d$. Construct, as in [109], a hierarchical cutting of $\mathbb{R}^d$ using $O(r^d)$ simplicial cells such that each simplicial cell is intersected by at most $\frac{M}{r}$ of the defining hyperplanes. This construction also gives us for each simplicial cell of the cutting the list of defining hyperplanes intersecting it. This takes $O(Mr^{d-1})$ time. Locate each active point inside the hierarchical cutting in time $O(N \log r)$. Let S be a simplicial cell of the hierarchical cutting. Denote by Π_S the set of active points in S. Partition each Π_S into

[6] Not including the independent monomial, namely, 1.

$\left\lceil \frac{|\Pi_S|}{Nr^{-2}} \right\rceil$ disjoint subsets of size at most $\frac{N}{r^d}$. For each simplicial cell, find the active inequalities whose corresponding geometric object (hyperplane, closed or open half-space) contains the cell. This takes $O(Mr^d)$ time. The whole step takes $O(N \log r + Mr^d)$ time.

4. For each of the $O(r^d)$ simplicial cells, recurse on the at most $\frac{N}{r^d}$ k-tuples p_i whose active point is inside the simplicial cell and the at most $\frac{M}{r}$ k-tuples q_j whose active inequality's defining hyperplane intersects the simplicial cell.

5. For each of the $O(r^d)$ simplicial cells, recurse on the at most $\frac{N}{r^d}$ $(k-1)$-prefixes of k-tuples p_i whose active point is inside the simplicial cell and the $(k-1)$-prefixes of k-tuples q_j whose active inequality's corresponding geometric object contains the simplicial cell.

Correctness In each recursive call, either k is decremented or M and N are divided by some constant strictly greater than one, hence, one of the conditions in steps **1** and **2** is met in each of the paths of the recursion tree and the algorithm always terminates. Step **5** is correct because it only recurses on (p_i, q_j) pairs whose suffix pairs are dominating. The base case in step **1** is correct because the only way for a pair (p_i, q_j) to reach this point is to have had all k components checked in step **5**. The base case in step **2** is correct by definition. Each dominating pair is output exactly once because the recursive calls of step **4** and **5** partition the set of pairs (p_i, q_j) that can still claim to be candidate dominating pairs.

Analysis For $k, N, M \geq 0$, the total complexity $T_k(N, M)$ of computing the inclusions for the first k components, excluding the output cost (steps **1** and **2**), is bounded, in general, by

$$T_k(N,M) \leq \underbrace{O(r^d)\, T_{k-1}(N,M)}_{\text{Step } \mathbf{5}} + \underbrace{O(r^d)\, T_k\left(\frac{N}{r^d}, \frac{M}{r}\right)}_{\text{Step } \mathbf{4}} + \underbrace{O(N+M)}_{\text{Step } \mathbf{3}},$$

and, in particular, by $T_k(N, M) = 0$ when $k = 0$, $T_k(N, M) = O(Nk)$ when $M < r$, and $T_k(N, M) = O(Mk)$ when $N < r^d$.

By point-hyperplane duality, $T_k(N, M) = T_k(M, N)$, hence, we can execute step **4** on dual linear inequalities and dual points to balance the

recurrence. For some constant $c_1 \geq 1$,

$$T_k(N, M) \leq c_1 r^{2d}\, T_{k-1}(N, M) + c_1 r^{2d}\, T_k\left(\frac{N}{r^{d+1}}, \frac{M}{r^{d+1}}\right) + c_1(N + M).$$

For simplicity, we ignore some problem-size reductions occuring in this balancing step.

Let $T_k(N) = T_k(N, N)$ denote the complexity of solving the problem when $M = N$, excluding the output cost. Hence, we have

$$T_k(N) \leq c_1 r^{2d}\, T_{k-1}(N) + c_1 r^{2d}\, T_k\left(\frac{N}{r^{d+1}}\right) + c_1 N, \tag{B.1}$$

and, in particular, $T_k(N) = 0$ when $k = 0$, and $T_k(N) = O(k)$ when $N < r^{d+1}$.

To get rid of the parameter k and progress into the analysis of the recurrence, Chan makes an ingenious change of variable [44]. With hindsight, choose $b = r^{d+1}$ and let

$$T(N') = \max\Big\{\, T_k(N) \colon\ k \geq 0, \quad N \geq 1, \ \text{ and } \ b^k N \leq N' \,\Big\}. \tag{B.2}$$

For the rest of the discussion, we shorten the notation to

$$T(N') = \max_{b^k N \leq N'} T_k(N).$$

By combining (B.1) and (B.2) we obtain

$$T(N') = \max_{b^k N \leq N'} T_k(N) \leq c_1 \max_{b^k N \leq N'} \left[r^{2d}\, T_{k-1}(N) + r^{2d}\, T_k\left(\frac{N}{r^{d+1}}\right) + N \right].$$

The maximum of a sum is always bounded by the sum of the maxima of its terms, hence,

$$T(N') \leq \max_{b^k N \leq N'} \Big[c_1 r^{2d}\, T_{k-1}(N) \Big] + \max_{b^k N \leq N'} \left[c_1 r^{2d}\, T_k\left(\frac{N}{r^{d+1}}\right) \right] + \max_{b^k N \leq N'} [c_1 N].$$

Looking at each term separately, by definition of $T(N')$, we have

$$\max_{b^k N \leq N'} T_{k-1}(N) = \max_{b^{k-1} N \leq \frac{N'}{b}} T_{k-1}(N) = T\left(\frac{N'}{b}\right) = T\left(\frac{N'}{r^{d+1}}\right),$$

$$\max_{b^k N \leq N'} T_k\left(\frac{N}{r^{d+1}}\right) = \max_{b^k \frac{N}{r^{d+1}} \leq \frac{N'}{r^{d+1}}} T_k\left(\frac{N}{r^{d+1}}\right) = T\left(\frac{N'}{r^{d+1}}\right),$$

$$\max_{b^k N \leq N'} N = \max_{N \leq \frac{N'}{b^k}} N = \frac{N'}{b^k} \leq N',$$

which, when combined with the previous inequality, produce the following recurrence

$$T(N') \leq 2c_1 r^{2d}\, T\left(\frac{N'}{r^{d+1}}\right) + c_1 N'.$$

Powers of r^{d+1} We claim that if N' is a power of r^{d+1}, then $T(N') \leq c_2[N'^{\alpha} - N']$ for some constants $\alpha > 1$ and $c_2 \geq 1$. We prove by induction that this (educated) guess is indeed correct. For $N' = 1$, we have

$$T(1) = \max_{b^k N \leq 1} T_k(N) = T_0(1) = 0 \leq c_2[1^{\alpha} - 1].$$

For $N' \geq r^{d+1}$ a power of r^{d+1}, and assuming the claim holds for all smaller powers:

$$\begin{aligned} T(N') &\leq 2c_1 r^{2d} c_2 \left[\left(\frac{N'}{r^{d+1}}\right)^{\alpha} - \frac{N'}{r^{d+1}}\right] + c_1 N' \\ &\leq c_2 N'^{\alpha}\left[\frac{2c_1 r^{2d}}{(r^{d+1})^{\alpha}}\right] - c_2 N'\left[2c_1 r^{d-1} - \frac{c_1}{c_2}\right]. \end{aligned}$$

We want

$$\frac{c_1 r^{2d}}{(r^{d+1})^{\alpha}} \leq \frac{1}{2} \qquad \text{and} \qquad 2c_1 r^{d-1} - \frac{c_1}{c_2} \geq 1.$$

For the first inequality, we can set the left hand side to be equal to $c_1 r^{-\epsilon'} = \frac{1}{2}$ with some small $\epsilon' = \frac{1+\log c_1}{\log r}$. Hence, $2d - \alpha(d+1) = -\epsilon'$, and for $\epsilon = \frac{\epsilon'}{d+1}$, we get $\alpha = \frac{2d}{d+1} + \epsilon$. The second inequality is equivalent to $2r^{d-1} \geq \frac{1}{c_1} + \frac{1}{c_2}$, which always holds since $r > 1, d \geq 1, c_1 \geq 1, c_2 \geq 1$.

We now have

$$T(N') = O\left(N'^{\frac{2d}{d+1}+\epsilon}\right),$$

where $\epsilon = \frac{1+\log c_1}{(d+1)\log r}$ can be chosen arbitrarily small by picking an arbitrarily large $r = (2c_1)^{1/\epsilon(d+1)}$.

Remark The choice $b = r^{d+1}$ gives a simpler analysis. Although giving more freedom to the value of b — as in Chan's paper — yields a slightly better relation between ϵ and r, namely $r > c_1^{1/\epsilon(d+1)}$, it does not get rid of the dependency of ϵ in r, unless $c_1 = 1$.

General case When $N' \geq 2$ is not a power of r^{d+1}, we use the fact that $T(N') \leq T(N'+1)$ by definition,

$$\begin{aligned}
T(N') &= T\Big((r^{d+1})^{\log_{r^{d+1}} N'}\Big) \\
&\leq T\Big((r^{d+1})^{\lfloor \log_{r^{d+1}} N' \rfloor + 1}\Big) \\
&= O\Big((r^{d+1})^{(\lfloor \log_{r^{d+1}} N' \rfloor + 1)(\frac{2d}{d+1}+\epsilon)}\Big) \\
&= O\Big((r^{d+1})^{\frac{2d}{d+1}+\epsilon} (r^{d+1})^{\lfloor \log_{r^{d+1}} N' \rfloor \frac{2d}{d+1}+\epsilon}\Big) \\
&= O\Big((r^{d+1})^{\lfloor \log_{r^{d+1}} N' \rfloor \frac{2d}{d+1}+\epsilon}\Big) \\
&= O\Big((r^{d+1})^{\log_{r^{d+1}} N' \frac{2d}{d+1}+\epsilon}\Big) \\
&= O\Big(N'^{\frac{2d}{d+1}+\epsilon}\Big).
\end{aligned}$$

Finally We can now bound $T_k(N)$ using the upper bound for $T(N')$,

$$T_k(N) \leq \max_{b^{k_i} N_i \leq b^k N} T_{k_i}(N_i) = T(b^k N) = O\Big((b^k N)^{\frac{2d}{d+1}+\epsilon}\Big) = 2^{O(k)} N^{\frac{2d}{d+1}+\epsilon}.$$

Hence, $T_k(N) = 2^{O(k)} N^{\frac{2d}{d+1}+\epsilon_r}$, and since $d = \binom{\Delta+2}{2} - 1$, we have

$$T_k(N) = 2^{O(k)} N^{2-\frac{4}{\Delta^2+3\Delta+2}+\epsilon_r}.$$

To that complexity we add a constant time unit for each output pair in steps **1** and **2**. □

B.3 Applications

To illustrate the expressive power of 3POL, we give some geometric applications in the sections that follow. We show the following:

1. GPT can be solved in subquadratic time provided the input points lie on few parameterized constant-degree polynomial curves.

2. In the plane, given three sets C_i of n unit circles and three points p_i such that a circle $c \in C_i$ contains p_i, deciding whether there exists $(a, b, c) \in C_1 \times C_2 \times C_3$ such that $a \cap b \cap c \neq \emptyset$ can be done in subquadratic time.
3. Given n input points in the plane, deciding whether any triple spans a unit triangle can be done in subquadratic time, provided the input points lie on few parameterized constant-degree polynomial curves.

B.3.1 GPT for Points on Curves

The following is a corollary of Theorem 2.16 in Raz, Sharir and de Zeeuw [133]

Corollary B.14 (Raz, Sharir and de Zeeuw [133])**.** *Any n points on an irreducible algebraic curve of degree d in $\mathbb{C}^2$ determine $\tilde{O}_d(n^{\frac{11}{6}})$ proper collinear triples, unless the curve is a line or a cubic.*

An interesting application of our results is the existence of subquadratic nonuniform and uniform algorithms for the computational version of this corollary.

Problem 20 (GPT on curves)**.** Let C_1, C_2, C_3 be three (not necessarily distinct) parameterized constant-degree polynomial curves in $\mathbb{R}^2$, so that each C_i can be written $(g_i(t), h_i(t))$ for some polynomials of constant degree g_i, h_i. Given three n-sets $S_1 \subset C_1, S_2 \subset C_2, S_3 \subset C_3$, decide whether there exist any collinear triple of points in $S_1 \times S_2 \times S_3$.

Theorem B.15. *GPT on curves reduces linearily to 3POL.*

Proof. For each set S_i, construct the set $T_i = \{\, t\colon\ p \in S_i, p = (g_i(t), h_i(t)) \,\}$. Testing whether there exists a collinear triple in $S_1 \times S_2 \times S_3$ amounts to testing whether any determinant

$$\begin{vmatrix} g_1(t_1) & h_1(t_1) & 1 \\ g_2(t_2) & h_2(t_2) & 1 \\ g_3(t_3) & h_3(t_3) & 1 \end{vmatrix}$$

equals zero. This determinant is a trivariate constant-degree polynomial in $\mathbb{R}[t_1, t_2, t_3]$. Solving the original problem amounts thus to deciding whether this polynomial cancels for any triple $(t_1, t_2, t_3) \in T_1 \times T_2 \times T_3$. □

Note that a similar polynomial predicate exists for testing collinearity in higher dimension.

Lemma B.16. *Let $p = (p_1, p_2, \ldots, p_d)$, $q = (q_1, q_2, \ldots, q_d)$, and $r = (r_1, r_2, \ldots, r_d)$ be three points in $\mathbb{R}^d$, then p, q, and r are collinear if and only if*

$$\left[\sum_{i=1}^{d}(p_i - r_i)(q_i - p_i)\right]^2 - \left[\sum_{i=1}^{d}(p_i - r_i)^2\right]\left[\sum_{i=1}^{d}(q_i - p_i)^2\right] = 0.$$

Proof. Let $a = (p_1, p_2, \ldots, p_d)$, $b = (q_1, q_2, \ldots, q_d)$, and $c = (r_1, r_2, \ldots, r_d)$ be three points in $\mathbb{R}^d$. The points p, q, and r are collinear if and only if $r = p + \lambda(q - p)$ for some unique $\lambda \in \mathbb{R}$, that is

$$\begin{aligned}
&(p - r) + \lambda(q - p) = 0,\\
\Rightarrow\quad &\forall i \in [d]\colon\ (p_i - r_i) + \lambda(q_i - p_i) = 0,\\
\Rightarrow\quad &\sum_{i=1}^{d}\left[(p_i - r_i) + \lambda(q_i - p_i)\right]^2 = 0,\\
\Rightarrow\quad &\sum_{i=1}^{d}\left[(q_i - p_i)^2\lambda^2 + 2(p_i - r_i)(q_i - p_i)\lambda + (p_i - r_i)^2\right] = 0,\\
\Rightarrow\quad &\underbrace{\left[\sum_{i=1}^{d}(q_i - p_i)^2\right]}_{A}\lambda^2 + \underbrace{\left[2\sum_{i=1}^{d}(p_i - r_i)(q_i - p_i)\right]}_{B}\lambda + \underbrace{\left[\sum_{i=1}^{d}(p_i - r_i)^2\right]}_{C} = 0,\\
\Rightarrow\quad &\lambda = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}.
\end{aligned}$$

For λ to exist and be unique $B^2 - 4AC$ must be zero. Hence, p, q, and r are collinear if and only if

$$\left[2\sum_{i=1}^{d}(p_i - r_i)(q_i - p_i)\right]^2 - 4\left[\sum_{i=1}^{d}(p_i - r_i)^2\right]\left[\sum_{i=1}^{d}(q_i - p_i)^2\right] = 0.$$

□

Moreover, the improvement that we obtain in the time complexity of 3POL can be exploited to boost the number of curves we pick the points from.

Theorem B.17. *Let $C_1, C_2, \ldots, C_k$ be $k = o\Big((\log n)^{\frac{1}{6}}/(\log\log n)^{\frac{1}{2}}\Big)$ (not necessarily distinct) constant-degree polynomial curves in $\mathbb{R}^d$. Given k n-sets $S_{i_j} \subset C_{i_j}$, deciding whether there exists any collinear triple of points in any triple of sets $S_{i_1} \times S_{i_2} \times S_{i_3}$ can be solved in subquadratic time.*

Proof. Solve a 3POL instance for each choice of $S_{i_1} \times S_{i_2} \times S_{i_3}$. Since there are $o\Big((\log n)^{\frac{1}{2}}/(\log\log n)^{\frac{3}{2}}\Big)$ such choices, the theorem follows. □

B.3.2 Incidences on Unit Circles

Raz, Sharir and Solymosi [137] mention the following problem as a special case of the framework they introduce. Let p_1, p_2, p_3 be three distinct points in the plane, and, for $i = 1, 2, 3$, let $\mathcal{C}_i$ be a family of n unit circles (a circle of radius 1) that pass through p_i. Their goal is to obtain an upper bound on the number of *triple points*, which are points that are incident to a circle of each family. They prove:

Theorem B.18. *Let p_1, p_2, p_3 be three distinct points in the plane, and, for $i = 1, 2, 3$, let $\mathcal{C}_i$ be a family of n unit circles that pass through p_i. Then the number of points incident to a circle of each family is $O(n^{11/6})$.*

They observe that the following dual formulation is equivalent to their original problem:

Theorem B.19. *Let C_1, C_2, C_3 be three unit circles in $\mathbb{R}^2$, and, for each $i = 1, 2, 3$, let S_i be a set of n points lying on C_i. Then the number of unit circles, spanned by triples of points in $S_1 \times S_2 \times S_3$, is $O(n^{11/6})$.*

Our new algorithms indeed allow us to solve the decision version of their problems in subquadratic time.

Problem 21. Let C_1, C_2, C_3 be three unit circles in $\mathbb{R}^2$ with centers c_i, and, for each $i = 1, 2, 3$, let $S_i = \{(x_{i,1}, y_{i,1}), (x_{i,2}, y_{i,2}), \ldots, (x_{i,n}, y_{i,n})\}$ be a set of n points lying on C_i. Decide whether any triple of points $(p_1, p_2, p_3) \in S_1 \times S_2 \times S_3$ spans a unit circle.

Theorem B.20. *Problem 21 can be solved in $O(n^2(\log\log n)^{\frac{3}{2}}/(\log n)^{\frac{1}{2}})$ time.*

Proof. Without loss of generality, assume all input points lie on the right y-monotone arc of their respective circle. All other seven cases can be handled similarly. We can also assume that no input point is the top or bottom vertex of its circle, rotating the plane if necessary.

Given three points p_1, p_2, and p_3, let

$$x = \|p_1 - p_2\|,\ X = x^2,\ y = \|p_1 - p_3\|,\ Y = y^2,\ z = \|p_2 - p_3\|,\ Z = z^2.$$

Testing if the three points p_1, p_2, and p_3 span a unit circle amounts to testing whether

$$X^2 + Y^2 + Z^2 - 2XY - 2XZ - 2YZ + XYZ = 0.$$

The fact that the input points lie on the right y-monotone arc of unit circles of centers c_1, c_2, c_3 allows us to get down to a single variable per point. Let $c_i = (c_i^x, c_i^y)$ and $t_{i,j} = \sqrt{\frac{1-x_{i,j}+c_i^x}{1+x_{i,j}-c_i^x}}$. Then the jth input point of the ith circle can be expressed as

$$p_{i,j} = (x_{i,j}, y_{i,j}) = c_i + \left(\frac{1 - t_{i,j}^2}{1 + t_{i,j}^2}, \frac{2t_{i,j}}{1 + t_{i,j}^2}\right).$$

Combining those two observations with some algebraic manipulations, one can show that there exists some trivariate polynomial F of degree at most 24 that cancels on t_1, t_2, t_3 when the points $c_1 + \left(\frac{1-t_1^2}{1+t_1^2}, \frac{2t_1}{1+t_1^2}\right)$, $c_2 + \left(\frac{1-t_2^2}{1+t_2^2}, \frac{2t_2}{1+t_2^2}\right)$, and $c_3 + \left(\frac{1-t_3^2}{1+t_3^2}, \frac{2t_3}{1+t_3^2}\right)$ span a unit circle.

Hence, the polynomial F together with the sets $\{t_{1,1}, t_{1,2}, \ldots, t_{1,n}\}$, $\{t_{2,1}, t_{2,2}, \ldots, t_{2,n}\}$, and $\{t_{3,1}, t_{3,2}, \ldots, t_{3,n}\}$ give an instance of 3POL we can solve in subquadratic time with our new algorithms.

Unfortunately, the computation $\sqrt{\cdot}$ is not allowed in our model, and so, we cannot compute $t_{i,j}$. However, we can generalize the 3POL problem to make it fit:

Problem 22 (Modified 3POL)**.** Let $F \in \mathbb{R}[x, y, z]$ be a trivariate polynomial of constant degree, given three sets A, B, and C, each containing n real numbers, decide whether there exist $a \in A$, $b \in B$, and $c \in C$ such that

$$\exists t_1, t_2, t_3 (t_1^2 = a \wedge t_2^2 = b \wedge t_3^2 = c \wedge F(t_1, t_2, t_3) = 0).$$

Hence, the polynomial F together with the sets $\{ t_{1,1}^2, t_{1,2}^2, \ldots, t_{1,n}^2 \}$, $\{ t_{2,1}^2, t_{2,2}^2, \ldots, t_{2,n}^2 \}$, and $\{ t_{3,1}^2, t_{3,2}^2, \ldots, t_{3,n}^2 \}$ give an instance of this variant of 3POL. Note that those sets are computable in our models.

We can tweak our algorithms so that they work for this new version of 3POL. We prefix each decision we make on the first-order theory of the reals with an existential quantifier and a condition of the type $t_i^2 = x$, with x the square of t_i, when we reference t_i in the formula we test. This new algorithm answers positively if and only if the original problem contains a triple of points spanning a unit circle.

In general, any constant-degree polynomial curve can be decomposed in a constant number of pieces as above. Each point on this curve can be given a parameterization that might involve roots of its coordinates. Those can be taken care of by appropriately augmenting the Tarski sentences in our algorithm with equations that encode those roots for free. □

B.3.3 Points Spanning Unit Triangles

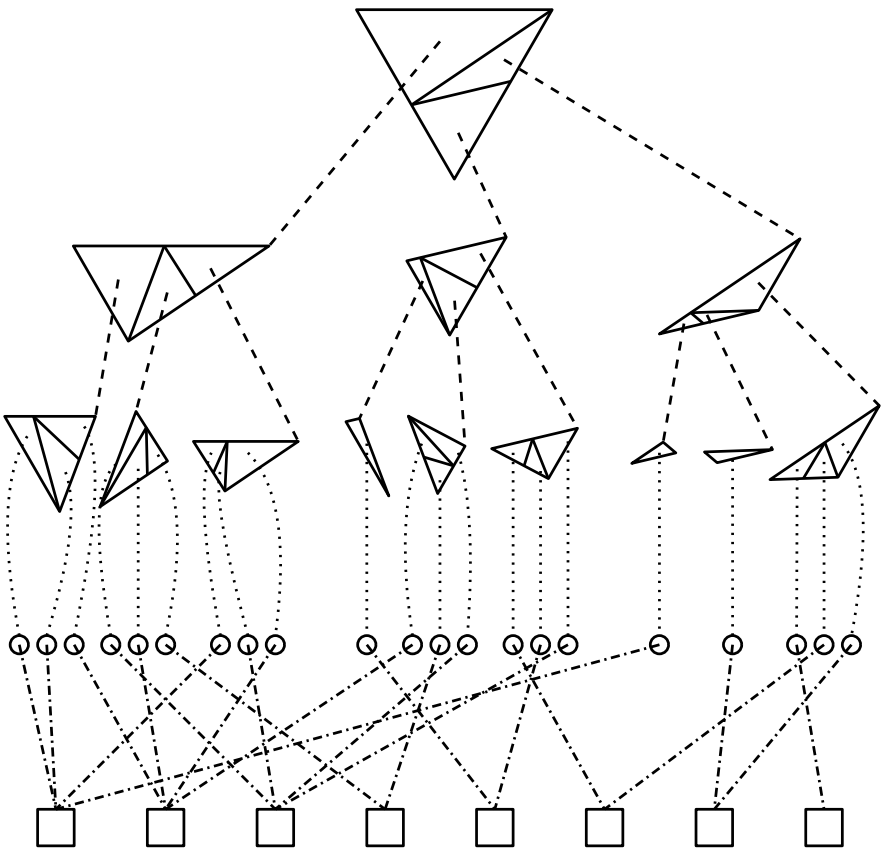
A similar problem, namely counting the number of input point triples spanning an area S triangle (provided they lie on a few curves), can also easily be reduced to 3POL. The polynomial to look at in this case is

$$F(x, y, z) = X^2 + Y^2 + Z^2 - 2XY - 2XZ - 2YZ + 16S^2.$$

Note that when the input points lie in the plane, the number of solutions is more than quadratic [132, 135].

IV

Data Structures



C

Subquadratic Encodings for Point Configurations

with Jean Cardinal, Timothy Chan, John Iacono, and Stefan Langerman

For many algorithms dealing with sets of points in the plane, the only relevant information carried by the input is the combinatorial configuration of the points: the orientation of each triple of points in the set (clockwise, counterclockwise, or collinear). This information is called the *order type* of the point set. In the dual, realizable order types and abstract order types are combinatorial analogues of line arrangements and pseudoline arrangements. Too often in the literature we analyze algorithms in the real-RAM model for simplicity, putting aside the fact that computers as we know them cannot handle arbitrary real numbers without some sort of encoding. Encoding an order type by the integer coordinates of a realizing point set is known to yield doubly exponential coordinates in some cases. Other known encodings can achieve quadratic space or fast orientation queries, but not both. In §C.1, we give a compact encoding for abstract order types that allows an efficient query of the orientation of any triple: the encoding uses $O(n^2)$ bits and an orientation query takes $O(\log n)$ time in the word-RAM model with word size $w \geq \log n$. This encoding is space-optimal for abstract order types. We show how to shorten the encoding to $O(n^2(\log\log n)^2/\log n)$ bits for realizable order types, giving the first subquadratic encoding for those order types with fast orientation queries. In §C.2, we further refine our encoding to attain $O(\log n/\log\log n)$ query time at the expense of a negligibly larger space requirement. In the realizable case, we show that

all those encodings can be computed efficiently. In §C.3, we generalize our results to the encoding of point configurations in higher dimension.

C.1 Encoding Order Types via Hierarchical Cuttings

To make our statements clear, we use the concept of an encoding. We recall its definition

Definition 2. For fixed k and given a function $f : [n]^k \to [O(1)]$, we define a $(S(n),Q(n))$-encoding of f to be a string of $S(n)$ bits such that, given this string and any $i \in [n]^k$, we can compute $f(i)$ in $Q(n)$ time in the word-RAM model with word size $w \geq \log n$.

In this section, we use this definition with f being some order type,[1] $k = 3$ and the codomain of f being $\{-, 0, +\}$. For the rest of the discussion, we assume the word-RAM model with word size $w \geq \log n$ and the standard arithmetic and bitwise operators. We prove our main theorems for the two-dimensional case:

Contribution 7. *All abstract order types have an encoding using $O(n^2)$ bits of space and allowing for queries in $O(\log n)$ time.*

Contribution 8. *All realizable order types have a $O(\frac{n^2(\log\log n)^2}{\log n})$-bits encoding allowing for queries in $O(\log n)$ time.*

Contribution 17. *In the real-RAM model and the constant-degree algebraic decision tree model, given n real-coordinate input points in $\mathbb{R}^2$ we can compute the encoding of their order type as in Theorems 7 and 8 in $O(n^2)$ time.*

For instance, Theorem 8 implies that for any set of points $\{p_1, \ldots, p_n\}$, there exists a string of $O(n^2(\log\log n)^2/\log n)$ bits such that given this string and any triple of indices $(a, b, c) \in [n]^3$ we can compute the value of $\chi(a, b, c) = \nabla(p_a, p_b, p_c)$ in $O(\log n)$ time.

Throughout the rest of this paper, we assume that we can access some arrangement of pairwise intersecting lines or pseudolines that realizes the

[1] Technically, we encode the orientation predicate of some realizing arrangement of the order type and skip the isomorphism. If desired, a canonical labeling of the arrangement can be produced in $O(n^2)$ time for abstract and realizable order types (see Lemma 7.1).

order type we want to encode. We thus exclusively focus on the problem of encoding the order type of a given arrangement. This does not pose a threat against the existence of an encoding. However, we have to be more careful when we bound the preprocessing time required to compute such an encoding. This is why, in Theorem 17, we specify the model of computation and how the input is given.

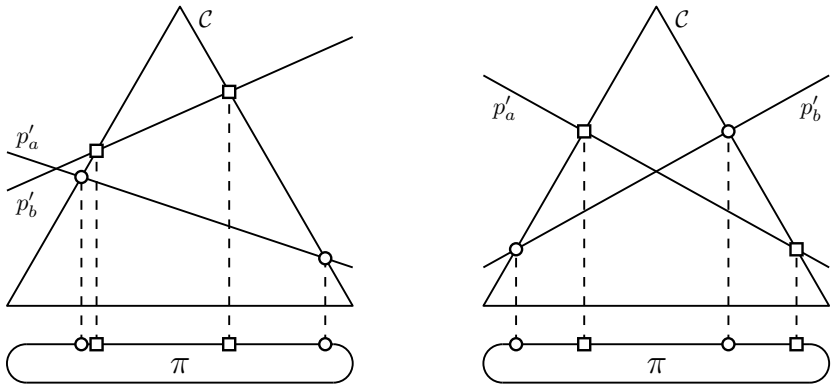
Idea We want to preprocess n pseudolines $\{p'_1, p'_2, \ldots, p'_n\}$ in the plane so that, given three indices a, b, and c, we can compute their orientation, that is, whether the intersection $p'_a \cap p'_b$ lies above, below or on p'_c. Our data structure builds on cuttings as follows: Given a cutting Ξ and the three indices, we can locate the intersection of p'_a and p'_b with respect to Ξ. The location of this intersection is a cell of Ξ. The next step is to decide whether p'_c lies above, lies below, contains or intersects that cell. In the first three cases, we are done. Otherwise, we can answer the query by recursing on the subset of pseudolines intersecting the cell containing the intersection. We build on hierarchical cuttings to control the size of each such subproblem.

Intersection Location When the arrangement consists of straight lines, locating the intersection $p'_a \cap p'_b$ in Ξ is trivial if we know the real parameters of p'_a and p'_b and of the descriptions of the subcells of Ξ. However, in our model we are not allowed to store real numbers. To circumvent this annoyance, and to handle arrangements of pseudolines, we make an observation illustrated by Figure C.1.

Definition 14. Given a set of pseudolines, a pseudocircle is a simple closed curve such that each pseudoline properly intersects it in exactly two points. A pseudodisk is a region bounded by a pseudocircle.

Observation C.1. *Two pseudolines p'_a and p'_b intersect in the interior of a pseudodisk $\mathcal{C}$ if and only if the intersections of p'_a and p'_b with $\mathcal{C}$ alternate on the boundary of $\mathcal{C}$.*

We construct Ξ so that its cells are pseudodisks for the pseudolines that intersect them. Hence, this observation gives us a way to encode the location of the intersection of p'_a and p'_b in Ξ using only bits. We formalize this observation with the following definition:



a. $p'_a \cap p'_b \cap \mathcal{C} = \emptyset$. **b.** $p'_a \cap p'_b \cap \mathcal{C} \neq \emptyset$.

Figure C.1. Cyclic permutations (π).

Definition 15 (Cyclic Permutation)**.** The *cyclic permutation* of a full-dimensional cell of a cutting is the cyclic ordering of the pseudolines crossing its boundary.

For an arrangement of pseudolines, we use the standard vertical decomposition (§6.4.2) to construct a hierarchical cutting (§8.2). This decomposition partitions the space into trapezoidal cells. For an arrangement of lines, we can use the standard bottom-vertex triangulation (§6.4.1) instead, which allows us to generalize our results to higher dimensions in §C.3. In the plane, the bottom-vertex triangulation partitions the space into triangular cells. Note that non-full-dimensional cells are easier to encode. For a 0-dimensional cell and a pseudoline, we store whether the pseudoline lies above, lies below, or contains the 0-dimensional cell. For a 1-dimensional cell, a pseudoline could also intersect the interior of the cell, but in only one point. The pseudolines intersecting that cell define an (acyclic) permutation with potentially several intersections at the same position. This information suffices to answer location queries for those cells, and the space taken is not more than that necessary for full-dimensional cells. When two pseudolines intersect in a 1-dimensional cell or contain the same 0-dimensional cell, they appear simultaneously in the cyclic permutation of an adjacent 2-dimensional cell if they intersect its interior. If that is the case, the location of the intersection of those two pseudolines in the cutting is the non-full-dimensional cell. A constant number of bits can be added to the encoding each time we need to

know the dimension of the cell we encode.

Encoding We encode the order type of an arrangement via hierarchical cuttings as defined in [48] (see §8.2). Given n pseudolines in the plane and some fixed parameters r and ℓ, compute a ℓ-levels hierarchical cutting of parameter r for those pseudolines as in Lemma 8.6. This hierarchical cutting consists of ℓ levels labeled $0, 1, \ldots, \ell - 1$. Level i has $O(r^{2i})$ cells. Each of those cells is further partitioned into $O(r^2)$ subcells. The $O(r^{2(i+1)})$ subcells of level i are the cells of level $i + 1$. Each cell of level i is intersected by at most $\frac{n}{r^i}$ pseudolines, and hence each subcell is intersected by at most $\frac{n}{r^{i+1}}$ pseudolines.

We compute and store a combinatorial representation of the hierarchical cutting as follows: For each level of the hierarchy, for each cell in that level, for each pseudoline intersecting that cell, for each subcell of that cell, we store two bits to indicate the location of the pseudoline with respect to that subcell, that is, whether the pseudoline lies above (`00`), lies below (`01`), intersects the interior of that subcell (`10`), or contains the subcell (`11`). When a pseudoline intersects the interior of a 2-dimensional subcell, we also store the two indices of that pseudoline in the cyclic permutation of that subcell, beginning at an arbitrary location in, say, clockwise order. If the intersected subcell is 1-dimensional instead, we store the index of the pseudoline in the acyclic permutation of that subcell, beginning at an arbitrary endpoint. If two pseudolines intersect in the interior of a 1-dimensional subcell or on the boundary of a 2-dimensional subcell, they share the same index in the permutation of that subcell.

This representation takes $O(\frac{n}{r^i} + \frac{n}{r^{i+1}} \log \frac{n}{r^{i+1}})$ bits per subcell of level i by storing for each pseudoline its location and, when needed, the permutation indices of its intersections with the subcell. At the last level of the hierarchy, let $t = \frac{n}{r^\ell}$ denote an upper bound on the number of pseudolines intersecting each subcell. For each of those $O(r^{2\ell}) = O(\frac{n^2}{t^2})$ subcells we store a pointer to a lookup table of size $O(t^3)$ that allows to answer the query of the orientation of any triple of pseudolines intersecting that subcell.

Storing the permutation at each subcell would suffice to answer all queries that do not reach the last level of the hierarchy. However, for those queries to be answered efficiently, we need to have access to all bits belonging to a given pseudoline without having to read the bits of the others. One solution is to

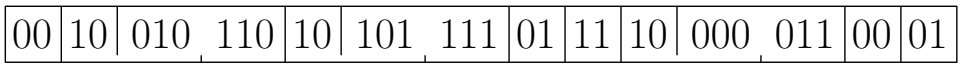


Figure C.2. A trace $\textsc{Tr}(\mathcal{C}, p')$. The cell $\mathcal{C}$ has eight subcells. Each subcell is intersected by at most four pseudolines. The pseudoline p' lies above two of them, lies below two of them, contains one of them, and intersects three of them at indices $(2, 6)$, $(5, 7)$, and $(0, 3)$.

augment each subcell with a hash table that translates pseudoline indices of the parent cell into pseudoline indices of the subcell. Another cleaner solution is to use the Zone Theorem (Theorem 6.2): by constructing the hierarchical cutting via decompositions of subsets of the input pseudolines, we can bound the number of subcells of a given cell a given pseudoline intersects by $O(r)$.[2] Hence, the number of bits stored for a single intersecting cell-pseudoline pair at level i is bounded by $|\textsc{Tr}_i| = O(r^2 + r \log \frac{n}{r^{i+1}})$. This bound allows us to store all bits belonging to a given cell-pseudoline pair $(\mathcal{C}, p')$ in a contiguous block of memory, denoted by $\textsc{Tr}(\mathcal{C}, p')$, whose address in memory is easy to compute (as detailed later on). The overall number of bits stored stays the same up to a constant factor. We call $\textsc{Tr}(\mathcal{C}, p')$ the *trace* of p' in $\mathcal{C}$. Figure C.2 depicts an example trace.

For queries that reach the last level of the hierarchy, storing an individual lookup table for each leaf would cost too much as soon as $t = \omega(1)$. However, as long as t is small enough, each order type is shared by many leaves, and we can thus save space. Formally, let $\nu(t)$ denote the number of order types of size t, which is $\nu(t) = 2^{\Theta(t^2)}$ for abstract order types [73] and $\nu(t) = 2^{\Theta(t \log t)}$ for realizable order types [16, 87]. At most $\nu(t)$ distinct lookup tables are needed to answer the queries on the subcells of the last level of the hierarchy. Hence, the pointers have size $|\textsc{Pointer}| = O(\log \nu(t))$ and the total space needed for the lookup tables is $O(t^3 \nu(t))$. For each leaf, we store a canonical labeling of size $|\textsc{Labeling}| = O(t \log t)$ on the pseudolines that intersect it, as in Lemma 7.1. We use that labeling to order the queries in the associated lookup table.[3]

[2] A reason to prefer this solution is that it enables the query time reduction in the next section.

[3] Note that the use of this canonical labeling is not necessary if t is constant, because then we can afford to use one lookup table per leaf. For superconstant t, the canonical labeling is necessary to get the construction time down to $O(n^2)$ as explained later. Space

Layout For completeness, we detail precisely how bits of the encoding are laid out in memory to allow an efficient decoding. Indeed, the data structure we encode is a tree and many space-efficient layouts exists for those when their nodes each have the same size. Here however, node size shrinks as we go down the hierarchy. We spend a few paragraphs showing it is still possible to address all components in a time- and space-efficient way. As this is fairly straightforward to adapt for the encodings in §C.2 and §C.3, we do not give the details for those sections.

The encoding is the concatenation of the parameters n, r, and t, the cells of the hierarchy, and the lookup tables. We order the cells of the hierarchy in a depth-first manner: a cell of level i is denoted by $\mathcal{C}_{0,j_1,j_2,\dots,j_i}$ with $j_1, j_2, \dots, j_i \in \{0, 1, \dots, O(r^2)\}$. The root cell is $\mathcal{C}_0$ and the cell $\mathcal{C}_{0,j_1,j_2,\dots,j_i,j_{i+1}}$ is the $(j_{i+1}+1)$-th subcell of the cell $\mathcal{C}_{0,j_1,j_2,\dots,j_i}$. Cells are then ordered lexicographically. For each leaf cell we store its pointer and canonical labeling. For each internal cell $\mathcal{C}$ we store the traces $\textsc{Tr}(\mathcal{C}, p')$ in a certain permutation of the pseudolines p'. To order the pseudolines, we use the first index of each pseudoline in the cyclic permutation of the cell (for the root cell $\mathcal{C}_0$ this is the index given by the input permutation). Note that those indices are between 0 and $2\lfloor \frac{N}{r^i} \rfloor - 1$ for a cell of level i. We allocate twice the required space for traces and canonical labelings to avoid defining a mapping between the indices of a cell and the indices of each of its subcells.

For the root cell of the hierarchy $\mathcal{C}_0$ representing the entire space and containing all the intersections of the arrangement, $\textsc{Addr}(\mathcal{C}_0)$ is the first free address after the encoding of the parameters n, r, and t.

The address $\textsc{Addr}(\mathcal{C}_{0,j_1,j_2,\dots,j_{i-1},0})$ of the first subcell of $\mathcal{C}_{0,j_1,j_2,\dots,j_{i-1}}$ is offset by twice the space taken by the traces for that cell

$$\textsc{Addr}(\mathcal{C}_{0,j_1,j_2,\dots,j_{i-1},0}) = \underbrace{\textsc{Addr}(\mathcal{C}_{0,j_1,j_2,\dots,j_{i-1}})}_{\text{Address of the parent cell}} + \underbrace{2\left\lfloor \frac{n}{r^{i-1}} \right\rfloor |\textsc{Tr}_{i-1}|}_{\text{Traces of the parent cell}} .$$

Let c_h be the constant hidden by the $O(r^2)$ of the hierarchical cutting.

and query complexity are not affected by this design choice (other than constant factors).

The space taken by a subtree rooted at a node of level i is bounded by

$$|\textsc{Subtree}_i| = \overbrace{2\left\lfloor \frac{n}{r^i} \right\rfloor |\textsc{Tr}_i|}^{\text{Traces at the root}} + \overbrace{2c_h \sum_{k=1}^{\ell-i-1} r^{2k} \left\lfloor \frac{n}{r^{i+k}} \right\rfloor |\textsc{Tr}_{i+k}|}^{\text{Traces of the subtrees}} + \underbrace{c_h r^{2(\ell-i)} |\textsc{Pointer}|}_{\text{Pointers}} + \underbrace{2c_h r^{2(\ell-i)} |\textsc{Labeling}|}_{\text{Canonical labelings}}.$$

The address of any other subcell $\textsc{Addr}(\mathcal{C}_{0,j_1,j_2,\ldots,j_{i-1},j_i})$ is offset by the space taken by the subtrees of its siblings $0, 1, \ldots, j_i - 1$

$$\textsc{Addr}(\mathcal{C}_{0,j_1,j_2,\ldots,j_{i-1},j_i}) = \underbrace{\textsc{Addr}(\mathcal{C}_{0,j_1,j_2,\ldots,j_{i-1},0})}_{\text{Address of the first sibling}} + \underbrace{j_i \cdot |\textsc{Subtree}_i|}_{\text{Left siblings subtrees}}.$$

The address of the trace $\textsc{Tr}(\mathcal{C}, p'_a)$, where $0 \le a \le 2\lfloor \frac{N}{r^i} \rfloor - 1$ is the first intersection index of p'_a in the cyclic permutation of the level-i cell $\mathcal{C}$, is

$$\textsc{Addr}(\mathcal{C}, p'_a) = \textsc{Addr}(\mathcal{C}) + a \cdot |\textsc{Tr}_i|.$$

Since we do not store any trace for the leaves, define $|\textsc{Tr}_\ell| = 0$. The pointer and canonical labeling of the leaf $\mathcal{C}_{0,j_1,j_2,\ldots,j_\ell}$ are concatenated at position $\textsc{Addr}(\mathcal{C}_{0,j_1,j_2,\ldots,j_\ell})$ and the lookup tables are concatenated at position $\textsc{Addr}(\mathcal{C}_0) + |\textsc{Subtree}_0|$.

This layout makes traversing the hierarchy from root to leaf efficient: The address of the root cell is discovered after parsing the parameters n, r, and t. The address of the first subcell of a parent cell is computed in constant time from the address of the parent cell. The size of the hierarchy is computed in time proportional its height. The size of a subtree of level $i+1$ is derived in constant time from the size of a subtree of level i. The address of any subcell of level i is computed in constant time from its index, the address of the parent cell, and the size of a subtree of level i. The address of a trace is computed in constant time from the address of its cell and the index of its pseudoline. The address of the pointer and canonical labeling of a leaf is the address of the corresponding cell. The address of a lookup table is computed in constant time given the address of the root cell, the size of the hierarchy, the size of a lookup table, and the leaf pointer.

During a traversal, pseudoline indices of the parent cell are mapped to indices in the subcell in constant time by using the first intersection index

of each pseudoline in that subcell. A final index mapping happens when translating leaf indices to lookup table indices using the canonical labeling of the leaf.

Space Complexity We first prove a general bound on the space taken by the encoding of a ℓ-level hierarchical cutting of parameter $r \geq 2$. For the space taken by the lookup tables, their associated pointers and canonical labelings at the leaves, and the parameters of the hierarchy n, r and t, the analysis is immediate.

Let $H_r^\ell(n) \in \mathbb{N}$ be the maximum amount of space (bits), over all arrangements of n pseudolines, taken by the $\ell \in \mathbb{N}$ levels of a hierarchy with parameter $r \in (1, +\infty)$.

Lemma C.2. *For $r \geq 2$ and $t = \frac{n}{r^\ell}$ we have*

$$H_r^\ell(n) = O\left(\frac{n^2}{t}(\log t + r)\right).$$

Proof. By definition, summing over all subcells, we have

$$H_r^\ell(n) = O\left(\sum_{i=0}^{\ell-1}\left(r^{2i} \cdot r^2 \cdot \left(\frac{n}{r^i} + \frac{n}{r^{i+1}} \log \frac{n}{r^{i+1}}\right)\right)\right).$$

Note that we obtain the same bound by summing over all traces (of intersecting cell-pseudoline pairs)

$$H_r^\ell(n) = O\left(n\sum_{i=0}^{\ell-1}\left(r^i \cdot \left(r^2 + r \log \frac{n}{r^{i+1}}\right)\right)\right).$$

Multiply any of the previous equations by $\frac{n}{tr^\ell} = 1$ to obtain

$$H_r^\ell(n) = O\left(\frac{n^2}{t}\sum_{i=0}^{\ell-1}\left(\frac{1}{r^{\ell-i-1}} \cdot \left(r + \log \frac{n}{r^{i+1}}\right)\right)\right).$$

We use the equivalence $\frac{n}{r^{i+1}} = tr^{\ell-i-1}$ to replace the last term in the previous equation

$$H_r^\ell(n) = O\left(\frac{n^2}{t}\sum_{i=0}^{\ell-1}\left(\frac{1}{r^{\ell-i-1}} \cdot (r + \log t + (\ell - i - 1)\log r)\right)\right).$$

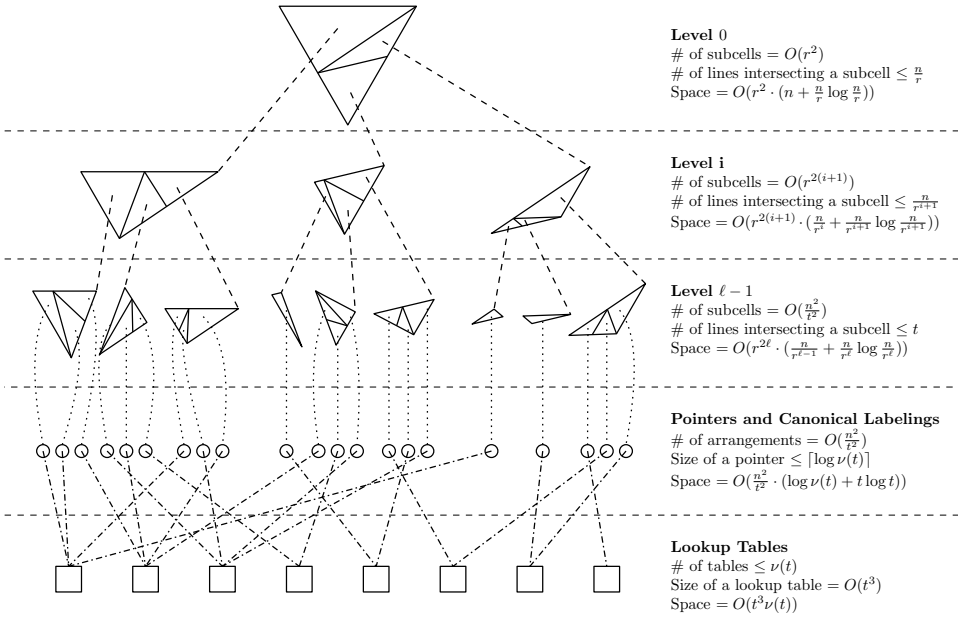


Figure C.3. Space analysis.

We reverse the summation by redefining $i \leftarrow \ell - i - 1$ and group the terms

$$H_r^\ell(n) = O\left(\frac{n^2}{t}\left((\log t + r)\sum_{i=0}^{\ell-1}\frac{1}{r^i} + \log r \sum_{i=0}^{\ell-1}\frac{i}{r^i}\right)\right).$$

Using the following inequalities (the geometric series and a multiple of its derivative):

$$\sum_{i=0}^{k} x^i \leq \frac{1}{1-x} \qquad \text{and} \qquad \sum_{i=0}^{k} i x^i \leq \frac{x}{(1-x)^2}, \qquad \forall k \in \mathbb{N}, \forall x \in (0,1),$$

we conclude that

$$H_r^\ell(n) = O\left(\frac{n^2}{t}\left(\left(1 + \frac{1}{r-1}\right)(\log t + r) + \left(1 + \frac{2r-1}{r^2-2r+1}\right)\frac{\log r}{r}\right)\right),$$

and the statement follows from $r \geq 2$. □

Figure C.3 sketches the different components of the encoding and shows the space taken by each of them. To that we must add the space taken by

the parameters of the hierarchy n, r and t if those are not implicitly known (here we assume the dimension $d = 2$ is implicitly known). We have thus the following bound:

Lemma C.3. *The space taken by the encoding described in §C.1 is proportional to*

$$\underbrace{\log ntr}_{\textit{Parameters}} + \underbrace{\frac{n^2}{t}(\log t + r)}_{\textit{Traces}} + \underbrace{\frac{n^2}{t^2}(\log \nu(t) + t \log t)}_{\textit{Pointers and Canonical Labelings}} + \underbrace{t^3 \nu(t)}_{\textit{Lookup Tables}} .$$

We pick r constant for both abstract and realizable order types. We have $\nu(t) = 2^{\Theta(n^2)}$ for abstract order types, hence we choose $t = \sqrt{\delta \log n}$ for small enough δ and the third term in Lemma C.3 dominates with n^2. Note how the quadratic bottleneck of this encoding is the storage of the order type pointers at the leaves of the hierarchy. We have $\nu(t) = 2^{\Theta(n \log n)}$ for realizable order types, hence we choose $t = \delta \log n / \log\log n$ for small enough δ and the second and third term in Lemma C.3 dominate with $n^2(\log\log n)^2 / \log n$. This proves the space constraints in Theorems 7 and 8.

Correctness and Query Complexity Given our encoding and three pseudoline indices a, b, c we answer a query as follows: We start by decoding the parameters n, r, and t. In our model, this can be done in $O(\log^* n + \log^* r + \log^* t)$ time, where $\log^*$ is the iterated logarithm (as in [109]).[4] Let $\mathcal{C} = \mathcal{C}_0$. First, find the subcell $\mathcal{C}'$ of $\mathcal{C}$ containing $p'_a \cap p'_b$ by testing for each subcell whether p'_a and p'_b alternate in its cyclic permutation. This can be done in $O(r^2)$ time by scanning $\textsc{Tr}(\mathcal{C}, p'_a)$ and $\textsc{Tr}(\mathcal{C}, p'_b)$ in parallel. Note that non-full dimensional cells and subcells are easier to test. Next, if p'_c does not properly intersect $\mathcal{C}'$, answer the query accordingly. If on the other hand p'_c does properly intersect the subcell we recurse on $\mathcal{C}'$. This can be tested by scanning $\textsc{Tr}(\mathcal{C}, p'_c)$ in $O(r^2)$ time. Note that in case that

[4] Logarithmic space and constant decoding time is trivial when $w = \Theta(\log n)$. If w is too large, encode n in binary using $\lceil \log n + 1 \rceil$ bits, $\lceil \log n + 1 \rceil$ using $\lceil \log \lceil \log n + 1 \rceil + 1 \rceil$ bits, $\lceil \log \lceil \log n + 1 \rceil + 1 \rceil$ using $\lceil \log \lceil \log \lceil \log n + 1 \rceil + 1 \rceil + 1 \rceil$ bits, etc. until the number to encode is smaller than a constant which we encode in unary with `1`'s. Prepend a `1` to the largest number and `0` to all the others. Concatenate those numbers from smallest to largest. Total space is $O(\log n)$ bits and decoding n can be done in $O(\log^* n)$ time in the word-RAM model with $w \geq \log n$. As an alternative, logarithmic space and logarithmic decoding time is also trivially achievable with no constraint on w.

the subcell is non-full-dimensional we can already answer the query. When we reach the relative interior of a subcell of the last level of the hierarchy without having found a satisfactory answer, we can answer the query by table lookup in constant time. This works as long as each order type identifier for at most t pseudolines fits in a constant number of words, which is the case for the values of t we defined. The layout described earlier makes all memory address computations of this query algorithm take constant time. The total query time is thus proportional to $r^2 \log_r n$ in the worst case, which is logarithmic since r is constant. This proves the query time constraints in Theorems 7 and 8.

With the hope of getting faster queries we could pick $r = \Theta(\log t)$ to reduce the depth of the hierarchy, without changing the space requirements by more than a constant factor. However, if no additional care is taken, this would slow the queries down by a $\Theta(\log^2 t / \log\log t)$ factor because of the scanning approach taken when locating the intersection $p'_a \cap p'_b$. We show how to handle small but superconstant r properly in the next section.

Preprocessing Time For a set of n points in the plane, or an arrangement of n lines in the dual, we can construct the encoding of their order type in quadratic time in the real-RAM and constant-degree algebraic computation tree models. We prove Theorem 17.

Proof. Using Lemma 8.6, a hierarchical cutting can be computed in $O(nr^\ell)$ time in the dual plane. All traces $\textsc{Tr}(\mathcal{C}, p')$ can be computed from the cutting in the same time. The lookup tables and leaf-table pointers can be computed in $O(n^2 + t^3\nu(t))$ time as follows: For each subcell $\mathcal{C}$ among the $\frac{n^2}{t^2}$ subcells of the last level of the hierarchy, compute a canonical labeling and representation of the lines intersecting $\mathcal{C}$ in $O(t^2)$ time as in Lemma 7.1. Insert the canonical representation in some trie in $O(t^2)$ time. If the canonical representation was not already in the trie, create a lookup table with the answers to all $O(t^3)$ queries on those lines and attach a pointer to that table in the trie. This happens at most $\nu(t)$ times. In the encoding, store the canonical labeling and this new pointer or the pointer that was already in the trie for the subcell $\mathcal{C}$. All parts of the encoding can be concatenated together in time proportional to the size of the encoding. □

C.2 Sublogarithmic Query Complexity

We further refine the encoding introduced in the previous section so as to reduce the query time by a $\log \log n$ factor. We do so using specificities of the word-RAM model that allow us to preprocess computations on inputs of small but superconstant size. This refinement is applicable to both abstract and realizable order types, and leads to an improvement of our main theorems for the two-dimensional case:

Contribution 9. *All abstract order types have an encoding using $O(n^2)$ bits of space and allowing for queries in $O(\frac{\log n}{\log \log n})$ time.*

Contribution 10. *All realizable order types have a $O(\frac{n^2 \log^\epsilon n}{\log n})$-bits encoding allowing for queries in $O(\frac{\log n}{\log \log n})$ time.*

Contribution 18. *In the real-RAM model and the constant-degree algebraic decision tree model, given n real-coordinate input points in $\mathbb{R}^2$ we can compute the encoding of their order type as in Theorems 9 and 10 in $O(n^2)$ time.*

Idea A natural idea is to pick r to be small but superconstant to reduce the number of levels of the hierarchy and thus the query time. As already pointed out, this has the drawback of increasing the time complexity of the intersection location primitive from constant to $\Theta(r^2)$. Since this primitive is used at each level of the hierarchy, the $\Theta(\log r)$ factor saved by having less levels is lost.

To get past this difficulty, the trick is to encode approximations of the traces $\textsc{Tr}(\mathcal{C}, p')$ to still allow constant intersection location. We call those approximations *signatures* and denote them by $\textsc{Sig}(\mathcal{C}, p')$. We define those signatures so that they approximately encode the cyclic permutation of the intersections around each subcell. By carefully choosing some parameters, we are able to fit two of those signatures in a single word of memory. We can then precompute the output to all possible inputs for the intersection location primitive and put them in a small table.

Because of this size reduction, distinct pseudolines could be mapped to identical signatures. Those ambiguous situations can be deterministically handled using an additional lookup table. Because those situations rarely arise, this table also is small.

Once we have located the intersection $p'_a \cap p'_b$, we still need to deal with p'_c. We change the layout of a trace to locate the subcell containing $p'_a \cap p'_b$ with respect to p'_c in constant time. In case p'_c properly intersects the subcell, we need to recurse on the subcell. To do that, we need to map the pseudoline indices a, b, and c to the indices those pseudolines have in that subcell. This is done with one indirection: For each of the three pseudolines we identify the index of the subcell in the list of subcells intersected by that pseudoline. This is implemented as a rank operation on a list of $O(r^2)$ bits. Given that index, we can find the intersection indices of that pseudoline in the cyclic permutation of the subcell in constant time.

In what follows, we describe five new structures: the signatures, the intersection oracle, the disambiguation table, the augmented traces, and the subcell mapper. The first reduces the bitsize of the original traces so that the second can be implemented in constant time. The third handles bad cases that arise because of this size reduction. The fourth defines a new layout for the traces that includes the signature. The fifth allows to implement the parent-to-subcell pseudoline index mapping in constant time using this new layout.

Signatures Fix a small constant α and define $r = \Theta(\log^\alpha n)$.[5] We encode a ℓ-levels hierarchical cutting of parameter r. Note that we can construct a hierarchical cutting with superconstant r by constructing a hierarchical cutting with some appropriate constant parameter r', and then skip levels that we do not need. As in §C.1, we store a combinatorial representation of this hierarchical cutting. We make some tweaks to this representation.

We augment the traces of §C.1 with a *signature*. The trace $\textsc{Tr}(\mathcal{C}, p')$ of a cell-pseudoline pair is composed of two parts: The incidence bits that tell us for each subcell of the cell whether the pseudoline lies above, lies below, intersects or contains it, and the cyclic permutation bits used to locate the intersection of two pseudolines inside the cell. The first part uses $\Theta(r^2)$ bits. The second part uses $\Theta(r \log \frac{N}{r^{i+1}})$ bits for a cell of level i.

To construct the signature $\textsc{Sig}(\mathcal{C}, p')$, we keep the $\Theta(r^2) = \Theta(\log^{2\alpha} n)$ incidence bits because they fit in sublogarithmic space for sufficiently small

[5] The exact bound for how small α must be depends on hidden constant factors in the Zone Theorem and in the definition of the word size. In particular, we must have $\alpha \leq \frac{1}{2}$ when $w = \Theta(\log n)$.

α. The second part would use superlogarithmic space if handled as before. We thus replace the $\Theta(r \log \frac{n}{r^{i+1}})$ bits of the cyclic permutation by a well chosen approximation.

Let $\beta = 2^{\Theta(\log^\alpha n)}$ and denote by $n_i = n/r^i$ an upper bound on the number of pseudolines intersecting a cell of level i. For each subcell of level i, partition its cyclic permutation into $\beta_i \leq \min\{\beta, n_{i+1}\}$ blocks of at most $\lceil n_{i+1}/\beta_i \rceil$ intersections. For each pseudoline intersecting a cell we store the block indices that that pseudoline touches instead of storing the cyclic permutation indices.[6] Hence, the second part of each signature only uses $O(r \log \beta) = O(\log^{2\alpha})$ bits.

Intersection Oracle We construct a lookup table to compute in constant time, for any given cell of any given level, the subcell in which $q_i \cap q_j$ lies. For that we need a general observation on the precomputation of functions on small universes.

Observation C.4. *In the word-RAM model with word size $w \geq \log n$, for any word-to-word function $f : [2^w] \to [2^w]$, we can build a lookup table of total bitsize $2^g h$ for all 2^g inputs $x \in [2^g]$ of bitsize $g \leq w$, mapping to images $y \in [2^h]$ of bitsize $h \leq w$, in time $2^g T(g)$ where $T(g)$ is the complexity of computing $f(x)$, $x \in [2^g]$. The image of bitsize h of any input of bitsize g can then be retrieved in $O(1)$ time by a single lookup (since inputs and outputs fit in a single word). In particular, the preprocessing time $2^g T(g)$ and the space $2^g h$ are sublinear as long as $T(g) = g^{O(1)}$ and $g + \log h = o(\log n)$.*

In other words, any polynomial time computable word-to-word function can be precomputed in sublinear time and space for all inputs and outputs of sublogarithmic size.

Since each pseudoline signature fits in $O(r^2 + r \log \beta) = O(\log^{2\alpha} n)$ bits, and since the number of subcells of each cell is $O(\log^{2\alpha} n)$, we can choose an appropriate α so as to satisfy the requirements given above: take $\alpha < \frac{1}{2}$ so that two pseudoline signatures have a combined bitsize of $g = o(\log n)$. The output size is the bitsize of a subcell identifier, which is $h \leq 2\alpha \log \log n = o(\log n)$. In some cases, the block indices stored in the

[6] For β a power of two, this can be implemented by truncating the original cyclic permutation indices.

signatures will not contain enough information to point to a unique output. In those cases we store a special value that indicates ambiguity of the input.

We can thus precompute the function that sends two pseudoline signatures to either the subcell containing their intersection or to some special value in case of an ambiguous input. Since we compute the function for all members of its universe, we can implement the lookup table using direct addressing into an array.

Note that the output of this oracle is the same no matter what level or cell we consider: for non-ambiguous inputs, all the information required to locate $q_i \cap q_j$ is included in the input. We thus only need a single lookup table, and the space needed is proportional to

$$2^g h = 2^{\Theta(\log^{2\alpha} n)}.$$

Disambiguation An input for the intersection oracle is ambiguous if and only if at least one boundary intersection of each input pseudoline appears in the same cyclic permutation block of the cell that contains their intersection. Thus, ambiguous inputs rarely occur: less than the number of blocks times the number of pairs of boundary intersections in a block, that is, less than $\beta \cdot (n_{i+1}/\beta)^2 = \frac{n^2}{r^{2(i+1)}}/\beta$ times per subcell of level i of the hierarchy. When $\beta \geq n_{i+1}$ all ambiguity is lifted so we can ignore those cases.

The disambiguation table is a hash table storing the answer to all ambiguous inputs for all cells of each level $i \in \{0, 1, \ldots, \ell - 1\}$. This table maps a triple of a cell $\mathcal{C}_{0,j_1,j_2,\ldots,j_i}$, a pseudoline p'_a, and a pseudoline p'_b to the index $j_{i+1} \in \{0, 1, \ldots, O(r^2)\}$ of the subcell $\mathcal{C}_{0,j_1,j_2,\ldots,j_i,j_{i+1}}$ containing the intersection $p'_a \cap p'_b$. Summing over all subcells of each level, we obtain that the number of entries in this table is bounded above by

$$\sum_{i=0}^{\ell-1} r^{2i} \cdot r^2 \cdot \frac{n^2}{r^{2(i+1)}}/\beta = \frac{n^2 \ell}{2^{\Theta(\log^{\alpha} n)}}.$$

The number of bits of each entry is at most $\ell\lceil \log cr^2 \rceil + 2 \log n = O(\log n)$ for the key and $\lceil \log cr^2 \rceil$ for the value. Both get absorbed by the $2^{-\Theta(\log^{\alpha} n)}$ factor in the number of entries, so we can keep the same expression for the number of bits used by the disambiguation table.

Since the keys and values fit in a constant number of words, we can guarantee worst case constant query time using cuckoo hashing [124] or

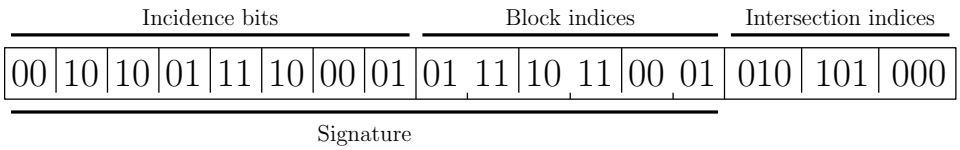


Figure C.4. An augmented trace $\textsc{Tr}'(\mathcal{C}, p')$ for the same cell-pseudoline pair as in Figure C.2.

perfect hashing [78]. In both cases the construction of the table is randomized and takes expected linear time in the number of entries. Perfect hashing has the advantage that we can drop the entry keys since we only query this table for existing entries. Note that this is the only part of the construction that is randomized.

Augmented Traces The augmented traces are simply the concatenation of the signature and the first intersection indices as depicted in Figure C.4. This layout allows for constant time access to the signature. Given a subcell index we can test its incidence bits in constant time. Given an intersected subcell index we can access the first intersection index of the pseudoline in that subcell in constant time.

As discussed earlier, the first part of the signature uses $O(r^2)$ bits. We already noted that the second part of the signature uses $O(r \log \beta) = O(\log^{2\alpha} n)$ bits. However, a better bound to use for the analysis of the total space is $O(r \log \beta_i) = O(r \log n_{i+1})$ bits, which is proportional to the number of bits needed for the first intersection indices.

Summing over all pseudolines and all intersected cells of each level, the space used for the augmented traces is proportional to

$$n \sum_{i=0}^{\ell-1} r^i \cdot \left(r^2 + r \log n_{i+1}\right) = O\left(\frac{n^2}{t}(\log t + r)\right),$$

as in Lemma C.2. Since $r = \Theta(\log^\alpha n)$ the bound becomes

$$O\left(\frac{n^2}{t}(\log t + \log^\alpha n)\right).$$

Subcell Mapper As before, let c_h be the constant hidden by the $O(r^2)$ of the hierarchical cutting, and let $c_z = 9.5$ be the constant in Theorem 6.2.

We need a way to map a subcell index $0 \leq j_i \leq c_h r^2 - 1$ to the index $0 \leq j'_i \leq \lfloor c_z r \rfloor - 2$ this subcell has in the list of subcells intersected by a given pseudoline. If we can achieve this in constant time, then we can also access the first intersection index this pseudoline has in this subcell.

It is not hard to see that this mapping operation boils down to computing $\mathsf{rank}_{\mathsf{10}}(j_i)$ where the data array is composed of the incidence bits of the given pseudoline. Hence this can be solved by adding an auxiliary rank-select data structure to each trace [24, 131]. Another solution is to reuse Observation C.4 to construct a lookup table for all $2^{\Theta(r^2)}$ possible incidence vectors.

With the first solution, the space used by the traces stays the same up to a constant factor. With the second solution, the space used is dominated by the space taken by the intersection oracle. Hence, we do not bother including it in the space analysis.

All that is left to do is to properly solve the subproblems spawned by the last level of the hierarchy. This is done exactly as in the previous section.

Leaves of the Hierarchy As before, we have $O(\frac{n^2}{t^2})$ subproblems of size t to encode. We follow the solution previously described to obtain: $O(\frac{n^2}{t^2})$ pointers of size $\lceil \log \nu(t) \rceil$, $O(\frac{n^2}{t^2})$ canonical labelings of size $\Theta(t \log t)$, and $\nu(t)$ lookup tables of size $O(t^3)$. This is sufficient to take care of each of those subproblems in constant time. The total space usage for those leaves is unchanged and stays proportional to

$$\frac{n^2}{t^2} \cdot (t \log t + \log \nu(t)) + t^3 \nu(t).$$

Space Complexity Summing all terms together and adding the space taken by the parameters of the hierarchy n, r and t, we obtain:

Lemma C.5. *The space taken by the encoding described in §C.2 is proportional to*

$$\overbrace{\log ntr}^{\textit{Parameters}} + \overbrace{\frac{n^2}{t}(\log t + \log^\alpha n) + 2^{\Theta(\log^{2\alpha} n)}}^{\textit{Traces}} + \overbrace{2^{\Theta(\log^{2\alpha} n)}}^{\textit{Intersection Oracle}} + \underbrace{\frac{n^2 \ell}{2^{\Theta(\log^\alpha n)}}}_{\textit{Disambiguation Table}} + \underbrace{\frac{n^2}{t^2}(\log \nu(t) + t \log t) + t^3 \nu(t)}_{\textit{Leaves}}.$$

As before, we take $t = \sqrt{\delta \log n}$ for abstract order types and $t = \delta \log n / \log\log n$ for realizable ones. Taking δ to be sufficiently small, the space taken by the leaves of the hierarchy is thus $\Theta(n^2)$ for abstract order types and dominated by the term $\frac{n^2}{t}\log t$ in the case of realizable order types. Setting $\alpha < \frac{1}{2}$ guarantees that the space taken by the intersection oracle is subpolynomial, and that the space taken by the traces is subquadratic. The space taken by the disambiguation table is in $O(\frac{n^2}{\log^c n})$ for all c and is thus dominated by the other terms.

For abstract order types, all those terms are subquadratic, except for the pointers at the leaves. The total space usage for abstract order types is thus dominated by this term and is quadratic. For realizable order types, the total space is dominated by the term $\frac{n^2}{t}\log^\alpha n$. Indeed, we can take α as small as desired to make the factor $\log^\alpha n = O(\log^\varepsilon n)$. This proves the space constraints in Theorems 9 and 10. Unfortunately, the present solution incurs a nonabsorbable extra $\log^\varepsilon n$ factor in the realizable case. Note that a $\log\log\log n$ factor can be squeezed from the query time without increasing the space usage by choosing $r = \Theta(\log\log n)$ instead.

Correctness and Query Complexity Given a query p'_a, p'_b, p'_c and a cell $\mathcal{C}$, the subcell $\mathcal{C}'$ containing $p'_a \cap p'_b$ is found in constant time via the intersection oracle and, if necessary, the disambiguation table. The location of that subcell with respect to p'_c can then be retrieved by a single lookup in the incidence bits of $\textsc{Tr}'(\mathcal{C}, p'_c)$. In case of recursion, we can compute the address of the traces $\textsc{Tr}'(\mathcal{C}', p'_a)$, $\textsc{Tr}'(\mathcal{C}', p'_b)$, and $\textsc{Tr}'(\mathcal{C}', p'_c)$ in constant time using the subcell mapper. The base case is handled in constant time as before: using the pointers, canonical labelings and order type lookup tables.

We now have a shallower decision tree of depth $\log_r \frac{n}{t} = O_\alpha(\frac{\log n}{\log\log n})$ and the work at each level takes constant time. This proves the query time constraints in Theorems 9 and 10.

Preprocessing Time We prove Theorem 18.

Proof. As before, the hierarchical cutting and all traces $\textsc{Tr}'(C, p')$ can be computed in $O(nr^\ell)$ time (with or without rank-select data structures). The lookup tables and leaf-table pointers can be computed in $O(n^2)$ time.

The intersection oracle, the disambiguation table, and the optional subcell mapper can be computed in subquadratic time. □

C.3 Higher-Dimensional Encodings

We generalize our point configuration encoding to any dimension d. The chirotope of a point set in $\mathbb{R}^d$ consists of all orientations of simplices defined by $d+1$ points of the set [139]. The orientation of the simplex with $d+1$ ordered vertices p_i with coordinates $(p_{i,1}, p_{i,2}, \ldots, p_{i,d})$ is given by the sign of the determinant

$$\begin{vmatrix} 1 & p_{1,1} & p_{1,2} & \cdots & p_{1,d} \\ 1 & p_{2,1} & p_{2,2} & \cdots & p_{2,d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & p_{d+1,1} & p_{d+1,2} & \cdots & p_{d+1,d} \end{vmatrix}.$$

We obtain the following generalized result:

Contribution 12. *All realizable chirotopes of rank $k \geq 4$ have an encoding using $O(\frac{n^{k-1}(\log\log n)^2}{\log n})$ bits of space and allowing for queries in $O(\frac{\log n}{\log\log n})$ time.*

Contribution 13. *In the real-RAM model and the constant-degree algebraic decision tree model, given n real-coordinate input points in $\mathbb{R}^d$ we can compute the encoding of their chirotope as in Theorem 12 in $O(n^d)$ time.*

Idea In the primal, the orientation of a simplex whose vertices are ordered can be interpreted as the location of its last vertex with respect to the (oriented) hyperplane spanned by its first d vertices. In the dual, this orientation corresponds to the location of the intersection of the first d dual hyperplanes with respect to the last (oriented) dual hyperplane. In the primal, degenerate simplices have orientation 0. In the dual, this corresponds to linearly dependent subsets of $d+1$ hyperplanes.

We gave an encoding for the two-dimensional case in §C.1. With this encoding, a query is answered by traversing the levels of some hierarchical cutting, branching on the location of the intersection of two of the three query lines. We generalize this idea to d dimensions. Now the cell considered at the next level of the hierarchy depends on the location of the intersection

of d of the $d+1$ query hyperplanes. We will also have to take care of degenerate cases.

Intersection Location In §C.1, we solved the following two-dimensional subproblem:

Problem 23. Given a triangle and n lines in the plane, build a data structure that, given two of those lines, allows to decide whether their intersection lies in the interior of the triangle.

In retrospective, we showed that there exists such a data structure using $O(n \log n)$ bits that allows for queries in $O(1)$ time. We generalize this result. Consider the following generalization of the problem in d dimensions:

Problem 24. Given a convex body and n hyperplanes in $\mathbb{R}^d$, build a data structure that, given d of those hyperplanes, allows to decide whether their intersection is a vertex that lies in the interior of the convex body.

Of course this problem can be solved using $O(n^d)$ space by explicitly storing the answers to all possible queries. If the input hyperplanes are given in an arbitrary order, this is best possible for $d = 1$. For $d \geq 2$, we show how to reduce the space to $O(n^{d-1} \log n)$ by recursing on the dimension, taking $d = 2$ as the base case.

We encode the function $\mathcal{I}_{C,H}$ that maps a d-tuple of indices of input dual hyperplanes H_i to 1 if their intersection is a vertex that lies in the interior of a fixed convex body C, and to 0 otherwise.

$$\mathcal{I}_{C,H} \colon [n]^d \to \{0,1\} \colon (i_1, i_2, \ldots, i_d) \mapsto (H_{i_1} \cap H_{i_2} \cap \cdots \cap H_{i_d}) \in C.$$

We call this function the *intersection function* of (C, H). We prove the following:

Lemma C.6. *All intersection functions have a $O(n^{d-1} \log n)$-bits $O(d)$-querytime encoding.*

Proof. Consider a convex body C and n hyperplanes H_i. At first, for simplicity, assume C is d-dimensional and assume that any d hyperplanes H_{i_j} meet in a single point. With those assumptions, we want a data structure that can answer any query of the type

$$(H_{i_1} \cap H_{i_2} \cap H_{i_3} \cap H_{i_4} \cap \cdots \cap H_{i_{d-1}} \cap H_{i_d}) \cap C \neq \emptyset.$$

Note that this is equivalent to deciding whether

$$(H_{i_1} \cap H_{i_2} \cap H_{i_3} \cap H_{i_4} \cap \cdots \cap H_{i_{d-1}}) \cap (H_{i_d} \cap C) \neq \emptyset,$$

where $H_{i_d} \cap C$ is a convex body of dimension $d-1$ (or empty), and the number of hyperplanes we want to intersect it with is $d-1$.

We unroll the recursion until the convex body is of dimension two (or empty), and only two hyperplanes are left to intersect. We then notice that the decision we are left with is equivalent to

$$(H_{i_1} \cap (H_{i_3} \cap H_{i_4} \cap \cdots \cap H_{i_d})) \cap (H_{i_2} \cap (H_{i_3} \cap H_{i_4} \cap \cdots \cap H_{i_d})) \\ \cap (C \cap (H_{i_3} \cap H_{i_4} \cap \cdots \cap H_{i_d})) \neq \emptyset,$$

which, if the three objects are non-empty, reads: "Given two lines and a convex body in some plane, do they intersect?". We can answer this query if we have the encoding for $d = 2$ which is obtained by replacing *triangle* by *convex body* in the two-dimensional original problem. The total space taken is multiplied by n for each time we unroll the recursion times the space taken in two dimensions, which is proportional to $n^{d-2} \cdot n \log n = O(n^{d-1} \log n)$. Queries can then be answered in $O(d)$ time.

Note that degenerate cases are likely to arise: empty convex bodies because of nonintersecting hyperplanes, convex bodies that are higher dimensional because of linearly dependent hyperplanes, and convex bodies that are lower dimensional because C was not full-dimensional to start with. However, all those cases can be dealt with appropriately: If the query suffix $H_{i_3} \cap H_{i_4} \cap \cdots \cap H_{i_d}$ leads to an empty convex body $C \cap (H_{i_3} \cap H_{i_4} \cap \cdots \cap H_{i_d})$ then the query point is not in C and we encode 0 for all the queries in C ending in this suffix. This information can be encoded in a table of size $O(n^{d-2})$. If the query suffix $H_{i_3} \cap H_{i_4} \cap \cdots \cap H_{i_d}$ leads to a convex body $C \cap (H_{i_3} \cap H_{i_4} \cap \cdots \cap H_{i_d})$ of dimension ≥ 3 then the intersection of all objects is not a 0-flat and we encode a 0 to follow the definition of $\mathcal{I}_{C,H}$. Again, this information can be encoded in a table of size $O(n^{d-2})$. If the query suffix $H_{i_3} \cap H_{i_4} \cap \cdots \cap H_{i_d}$ leads to a convex body $C \cap (H_{i_3} \cap H_{i_4} \cap \cdots \cap H_{i_d})$ of dimension zero, one, or two, then we use the encoding of size $O(n \log n)$ described in §C.1 as a base case. □

In the paragraphs that follow, we show how to plug this result in those

of the previous sections to obtain analogous results for the d-dimensional version of the problem (Theorem 12 and Theorem 13).

Encoding We build a hierarchical cutting as in §C.1 (this time in dimension d). Given n hyperplanes in $\mathbb{R}^d$ and some fixed parameters r and ℓ, compute a ℓ-levels hierarchical cutting of parameter r for those hyperplanes as in Lemma 8.5. This hierarchical cutting consists of ℓ levels labeled $0, 1, \ldots, \ell-1$. Level i has $O(r^{di})$ cells. Each of those cells is further partitioned into $O(r^d)$ subcells. The $O(r^{d(i+1)})$ subcells of level i are the cells of level $i+1$. Each cell of level i is intersected by at most $\frac{n}{r^i}$ hyperplanes, and hence each subcell is intersected by at most $\frac{n}{r^{i+1}}$ hyperplanes.

We compute and store a combinatorial representation of the hierarchical cutting as follows: For each level of the hierarchy, for each cell in that level, for each hyperplane intersecting that cell, for each subcell of that cell, we store two bits to indicate the location of the hyperplane with respect to that subcell, that is, whether the hyperplane lies above (`00`), lies below (`01`), intersects the interior of that subcell (`10`), or contains the subcell (`11`). When a hyperplane H_i intersects the interior of a subcell $\mathcal{C}'$, we also store the two $O(\log \frac{n}{r^{i+1}})$-bits two-dimensional intersections indices this hyperplane has in each of the $O\left(\left(\frac{n}{r^{i+1}}\right)^{d-2}\right)$ query suffixes $(H_{i_3} \cap H_{i_4} \cap \ldots \cap H_{i_d})$ with each of the H_{i_j} properly intersecting $\mathcal{C}'$.

This representation takes $O\left(\frac{n}{r^i} + \left(\frac{n}{r^{i+1}}\right)^{d-1} \log \frac{n}{r^{i+1}}\right)$ bits per subcell of level i by storing for each hyperplane its location and, when needed, the bits they hold in the encoding of the intersection function of the subcell. At the last level of the hierarchy, let $t = \frac{n}{r^\ell}$ denote an upper bound on the number of hyperplanes intersecting each subcell. For each of those $O(r^{d\ell}) = O(\frac{n^d}{t^d})$ subcells we store a pointer to a lookup table of size $O(t^{d+1})$ that allows to answer the query of the orientation of any triple of hyperplanes intersecting that subcell.

Using the Zone Theorem in higher dimensions (Theorem 6.3), we can have all bits belonging to a single cell-hyperplane pair in a contiguous block of memory with the same space bound.

Space Complexity As before, for the space taken by the lookup tables, their associated pointers and canonical labelings at the leaves, and the

parameters of the hierarchy n, r and t, the analysis is immediate. If not implicitly known, the dimension d can also trivially be added to the encoding.

For the space taken by the hierarchy, we generalize Lemma C.2 of §C.1. Let $H_r^\ell(n,d) \in \mathbb{N}$ be the maximum amount of space (bits), over all arrangements of n hyperplanes in $\mathbb{R}^d$, taken by the $\ell \in \mathbb{N}$ levels of a hierarchy with parameter $r \in (1, +\infty)$.

Lemma C.7. *For $r \geq 2$ we have*

$$H_r^\ell(n,d) = O\left(\frac{n^d}{t}\left(\log t + \frac{r}{t^{d-2}}\right)\right).$$

Proof. By definition we have

$$H_r^\ell(n,d) = O\left(\sum_{i=0}^{\ell-1}\left(r^{di}\cdot r^d\cdot\left(\frac{n}{r^i} + \left(\frac{n}{r^{i+1}}\right)^{d-1}\cdot\log\frac{n}{r^{i+1}}\right)\right)\right).$$

Using $t = \frac{n}{r^\ell}$, reversing the summation with $i \leftarrow \ell - i - 1$, and grouping the terms, we have

$$H_r^\ell(n,d) = O\left(\frac{n^d}{t}\left(\frac{r}{t^{d-2}}\sum_{i=0}^{\ell-1}\frac{1}{\left(r^{d-1}\right)^i} + \log t\sum_{i=0}^{\ell-1}\frac{1}{r^i} + \log r\sum_{i=0}^{\ell-1}\frac{i}{r^i}\right)\right).$$

Using the geometric inequalities (see the proof of Lemma C.2) the statement follows from $r \geq 2$. □

The final picture is almost the same as in Figure C.3. Summing all terms, we obtain

Lemma C.8. *The space taken by the encoding described in §C.3 is proportional to*

$$\underbrace{\log dntr}_{\textit{Parameters}} + \underbrace{\frac{n^d}{t}\left(\log t + \frac{r}{t^{d-2}}\right)}_{\textit{Traces}} + \underbrace{\frac{n^d}{t^d}(\log\nu_d(t) + t\log t)}_{\textit{Leaves}} + \underbrace{t^{d+1}\nu_d(t)}_{\textit{Lookup Tables}},$$

where $\nu_d(t) = 2^{\Theta(d^2 t\log t)}$ denotes the number of realizable rank-$(d+1)$ chirotopes of size t.

We pick r constant and choose $t = \delta\log n/\log\log n$ for small enough δ. The second term in Lemma C.8 dominates with $n^d(\log\log n)^2/\log n$. This proves the space constraint in Theorem 12.

Correctness and Query Complexity As before, a query is answered by traversing the hierarchy, which takes $O(\log n)$ time. The query time can be further improved using the method from §C.2 with $r = \Theta(t^{d-2} \log t)$. This proves the query time constraint in Theorem 12.

Preprocessing Time We prove Theorem 13.

Proof. The hierarchical cuttings can be computed in $O(n(r^\ell)^{d-1})$ time. The lookup table and leaf-table pointers can be computed in $O(n^d)$ time using the canonical labeling and representation for rank-$(d+1)$ chirotopes given in [17]. The intersection oracle, the disambiguation table, and the subcell mapper can be computed in $o(n^d)$ time. □

D

Encoding 3SUM

with Sergio Cabello, Jean Cardinal, John Iacono, Stefan Langerman, and Pat Morin

Given three sets of n real numbers $A = \{ a_1 < a_2 < \cdots < a_n \}$, $B = \{ b_1 < b_2 < \cdots < b_n \}$, and $C = \{ c_1 < c_2 < \cdots < c_n \}$, we wish to build a discrete data structure (using bits, words, and pointers) such that, given any triple $(i, j, k) \in [n]^3$ it is possible to compute the sign of $a_i + b_j + c_k$ by only inspecting the data structure (we cannot consult A, B, or C). We refer to the map $\chi : [n]^3 \to \{-, 0, +\}, (i, j, k) \mapsto \operatorname{sgn}(a_i + b_i + c_k)$ as the *3SUM type* of the instance $\langle A, B, C \rangle$.

Obviously, one can simply construct a lookup table of size $O(n^3)$, such that triple queries can be answered in $O(1)$ time. In §D.1 we show that a minimal integer representation of a 3SUM instance may require $\Theta(n)$ bits per value, yielding $O(n)$ query time and $O(n^2)$ space. In §D.2 we show how to use an optimal $O(n \log n)$ bits of space with a polynomial query time. Finally, in §D.3 we show how to use $\tilde{O}(n^{3/2})$ space to achieve $O(1)$-time queries.

D.1 Representation by Numbers

A first natural idea is to encode the real 3SUM instance by *rounding* its numbers to integers. We show a tight bound of $\Theta(n^2)$ bits for this representation.

Contribution 14. *Every 3SUM instance has an equivalent integer instance where all values have absolute value at most $2^{O(n)}$. Furthermore, there exists an instance of 3SUM where all equivalent integer instances require numbers*

at least as large as the nth Fibonacci number and where the standard binary representation of the instance requires $\Omega(n^2)$ bits.

Proof. Every 3SUM instance $A = \{a_1 < a_2 < \ldots < a_n\}$, $B = \{b_1 < b_2 < \cdots < b_n\}$, and $C = \{c_1 < c_2 < \cdots < c_n\}$ can be interpreted as the point $(a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_n)$ in $\mathbb{R}^{3n}$. Let us use the variables $x_1, \ldots, x_n$ to encode the first n dimensions of $\mathbb{R}^{3n}$, $y_1, \ldots, y_n$ to encode the next n dimensions, and $z_1, \ldots, z_n$ for the remaining dimensions. Consider the subset of $\mathbb{R}^{3n}$

$$\Delta = \{(x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n): \\ x_i < x_{i+1},\ y_j < y_{j+1},\ z_k < z_{k+1}\ \forall i, j, k \in [n-1]\}$$

and the set Π of n^3 hyperplanes $x_i + y_j + z_k = 0$, where $i, j, k \in [n]$. Let $\mathcal{A}$ be the arrangement defined by Π *inside* Δ. Instances of 3SUM correspond to points in Δ. Moreoever, two 3SUM instances have the same 3SUM type if and only if they are in the same cell of $\mathcal{A}$.

Consider an instance $\langle A, B, C\rangle$ and let $\sigma = \sigma(A, B, C)$ be the cell of $\mathcal{A}$ that contains it. Then σ is the cell defined by the inequalities

$$\forall i, j, k \in [n]: \quad \begin{cases} x_i + y_j + z_k > 0 & \text{if } \chi(i,j,k) = +1, \\ x_i + y_j + z_k = 0 & \text{if } \chi(i,j,k) = 0, \\ x_i + y_j + z_k < 0 & \text{if } \chi(i,j,k) = -1. \end{cases}$$

$$\forall i, j, k \in [n-1]: \quad \begin{cases} x_i - x_{i+1} < 0, \\ y_j - y_{j+1} < 0, \\ z_k - z_{k+1} < 0. \end{cases}$$

Let σ' be the subset of $\mathbb{R}^{3n}$ defined by the following inequalities:

$$\forall i, j, k \in [n]: \quad \begin{cases} x_i + y_j + z_k \geq 1 & \text{if } \chi(i,j,k) = +1, \\ x_i + y_j + z_k = 0 & \text{if } \chi(i,j,k) = 0, \\ x_i + y_j + z_k \leq -1 & \text{if } \chi(i,j,k) = -1. \end{cases}$$

$$\forall i, j, k \in [n-1]: \quad \begin{cases} x_i - x_{i+1} \leq 1, \\ y_j - y_{j+1} \leq 1, \\ z_k - z_{k+1} \leq 1. \end{cases}$$

Clearly σ' is contained in σ. Moreover, for a sufficiently large $\lambda > 0$ the scaled instance $\langle \lambda A, \lambda B, \lambda C \rangle$ belongs to σ'. Therefore, σ' is nonempty.

Since σ' is defined by a collection of linear inequalities defining closed halfspaces, there exists a point p in σ' defined by a subset of at most $3n$ inequalities, where the inequalities are actually equalities. Let us assume for simplicity that exactly $3n$ equalities define the point p. Then, $p = (x, y, z)$ is the solution to a linear system of equations $M[x\ y\ z]^T = \delta$ where M and δ have their entries in $\{-1, 0, 1\}$ and each row of M has at most three non-zero entries. The solution p to this system of equations is an instance equivalent to $\langle \lambda A, \lambda B, \lambda C \rangle$.

Because of Cramer's rule, the system of linear equations has solution with entries $\det(M_i)/\det(M)$, where M_i is the matrix obtained by replacing the ith column of M by δ. We use the following simple bound on the determinant. Since $\det(M) = \sum_\pi \operatorname{sgn}(\pi) \prod_i m_{i,\pi(i)}$, where π iterates over the permutations of $[3n]$, there are at most 3^{3n} summands where π gives non-zero product $\prod_i m_{i,\pi(i)}$ (we have to select one non-zero entry per row), and the product is always in $\{-1, 0, 1\}$. Therefore $|\det(M)| \leq 3^{3n}$. Similarly, $|\det(M_i)| \leq 4^{3n}$ because each row of M_i has at most 4 non-zero entries. We conclude that the solution to the system $M[x\ y\ z]^T = \delta$ are rationals that can be expressed with $O(n)$ bits. This solution gives a 3SUM instance with rationals that is equivalent to $\langle A, B, C \rangle$. Since all the rationals have the common denominator $(\det(M))$, we can scale the result by $\det(M)$ and we get an equivalent instance with integers, where each integer has $O(n)$ bits.

The proof of the second statement is by implementing the Fibonacci recurrence in each of the arrays A, B, C. This can be achieved by letting:

$$\begin{aligned} a_i + b_1 + c_{n-i+1} &= 0, \text{for } i \in [n] \\ a_1 + b_i + c_{n-i+1} &= 0, \text{for } i \in [n] \\ a_{i-1} + b_{i-2} + c_{n-i+1} &< 0, \text{for } i \in \{3, 4, \ldots, n\}, \end{aligned}$$

The first two sets of equations ensure that the two arrays A and B are identical, while the array C contains the corresponding negated numbers, in reverse order. From the inequalities in the third group, and depending on the choice of the initial values a_1, a_2, each array contains a sequence growing at least as fast as the Fibonacci sequence. □

Note that this is a much smaller lower bound than for order types of

points sets in the plane, the explicit representation of which can be shown to require exponentially many bits per coordinate [90].

D.2 Space-Optimal Representation

By considering the arrangement of hyperplanes defining the 3SUM problem, we get an information-theoretic lower bound on the number of bits in a 3SUM-type.

Lemma D.1. *There are* $2^{\Theta(n \log n)}$ *distinct 3SUM-types of size* n*.*

Proof. 3SUM-types of size n are in one-to-one correspondence with cells of the arrangement of n^3 hyperplanes in $\mathbb{R}^{3n}$. The number of such cells is $O(n^{9n})$ and at least $(n!)^2$. □

In order to reach this lower bound, we can simply encode the label of the cell of the arrangement in $\Theta(n \log n)$ bits. However, decoding the information requires to construct the whole arrangement which takes $n^{O(n)}$ time. An alternative solution is to store a vertex of the arrangement of hyperplanes $a_i + b_j + c_k \in \{-1, 0, 1\}$. There exists such a vertex that has the same 3SUM-type as the input point, as shown in the proof of Contribution 14. To answer any query, either recompute the vertex from the basis then answer the query using arithmetic, or use linear programming. Hence we can build a data structure of $O(n \log n)$ bits such that triple queries can be answered in polynomial time.

Note that we do not exploit much of the 3SUM structure here. In particular, the same essentially holds for k-SUM, and can also be generalized to a Subset Sum data structure of $O(n^2)$ bits, from which we can extract the sign of the sum of any subset of numbers.

D.3 Subquadratic Space and Constant Query Time

Our encoding is inspired by Grønlund and Pettie's $\tilde{O}(n^{3/2})$ non-uniform algorithm for 3SUM [91]. Our data structure stores three components, which we call the *differences*, the *staircase* and the *square neighbors*.

Differences. Partition A and B into *blocks* of $\sqrt{n}$ consecutive elements. Let D be the set of all differences of the form $a_{i_1} - a_{i_2}$ and $b_{j_1} - b_{j_2}$ where the

items come from the same block. There are $O(n^{3/2})$ such differences. Sort D and store a table indicating for each difference in D its rank among all differences in D. This takes $O(\log n)$ bits for each of the $O(n^{3/2})$ differences, for a total of $O(n^{3/2} \log n)$ bits.

Staircase. Look at the table G formed by all sums of the form $a_i + b_j$. It is monotonic in its rows and columns due to A and B being sorted. We view it as being partitioned into a grid G' of size $\sqrt{n} \times \sqrt{n}$ where each *square* of the grid is also of size $\sqrt{n} \times \sqrt{n}$. For each element $c_k \in C$, for each $i' \in [1, \sqrt{n}]$ we store the largest j' such that some elements of the square $G'[i', j']$ are $< c$, denote this as $L[k, i']$. We also store, for each $c_k \in C$, for each $i' \in [1, \sqrt{n}]$ the smallest j' such that some elements of the square $G'[i', j']$ are $\geq c$, denote this as $U[k, i']$. We thus store, in L and U, $2\sqrt{n}$ values of size $O(\log n)$ for each of the n elements of C, for a total space usage of $O(n^{3/2} \log n)$ bits. We call this the *staircase* as this implicitly classifies, for each $c \in C$, whether each square has elements larger than c, smaller than c, or some larger and some smaller; only $O(\sqrt{n})$ can be in the last case, which we refer to as the *staircase* of c.

Square neighbors. For each element $c_k \in C$, for each of the $O(\sqrt{n})$ squares on the staircase, we store the location of the predecessor and successor of c_k. Those squares are the $G'[i', j']$ such that $L[k, i'] \leq j' \leq U[k, i']$, for $i', j' \in [1, \sqrt{n}]$. This takes space $O(n^{3/2} \log n)$.

To execute a query (a_i, b_j, c_k), only a constant number of lookups in the tables stored are needed. Let us define $i' = \lceil i/\sqrt{n} \rceil$ and $j' = \lceil i/\sqrt{n} \rceil$ to be the indices of the cell of G' containing the point (a_i, b_j). If $j' < L[k, i']$, then we know $a_i + b_j < c_k$. If $j' > U[k, j']$, then we know $a_i + b_j > c_k$. If neither of these is true, then the square $G'[i', j']$ is on the staircase of c_k and thus using the square neighbors table we can determine the location of the predecessor and successor of c_k in this square; suppose they are at $G[s_i, s_j]$ and $G[p_i, p_j]$ and thus $G[s_i, s_j] \leq c_k \leq G[p_i, p_j]$. One need only determine how these two compare to $G[i, j] = a_i + b_j$ to answer the query. But this can be done using the differences as follows: to compare $G[s_i, s_j]$ to $G[i, j]$ this would be determining the sign of $(a_i + b_j) - (a_{s_i} + b_{s_j})$ which is equivalent to determining the result of comparing $a_i - a_{s_i}$ and $b_j - b_{s_j}$, which since both are in the same square, these differences are in D and the comparison can be

	1	2	10	14	17	22	32	33	40	91	92	97	98	110	120	127
1	2	3	11	15	18	23	33	34	41	92	93	98	99	111	121	128
11	12	13	21	25	28	33	43	44	51	102	103	108	109	121	131	138
13	14	15	23	27	30	35	45	46	53	104	105	110	111	123	133	140
19	20	21	29	33	36	41	51	52	59	110	111	116	117	129	139	146
24	25	26	34	38	41	46	56	57	64	115	116	121	122	134	144	151
34	35	36	44	48	51	56	66	67	74	125	126	131	132	144	154	161
51	52	53	61	65	68	73	83	84	91	142	143	148	149	161	171	178
57	58	59	67	71	74	79	89	90	97	148	149	154	155	167	177	184
59	60	61	69	73	76	81	91	92	99	150	151	156	157	169	179	186
114	115	116	124	128	131	136	146	147	154	205	206	211	212	224	234	241
119	120	121	129	133	136	141	151	152	159	210	211	216	217	229	239	246
127	128	129	137	141	144	149	159	160	167	218	219	224	225	237	247	254
128	129	130	138	142	145	150	160	161	168	219	220	225	226	238	248	255
133	134	135	143	147	150	155	165	166	173	224	225	230	231	243	253	260
138	139	140	148	152	155	160	170	171	178	229	230	235	236	248	258	265
142	143	144	152	156	159	164	174	175	182	233	234	239	240	252	262	269

Figure D.1. Illustration of the staircase and square neighbors of the constant query time encoding. Here the 16×16 table is partitioned into a 4×4 grid of squares of size 4×4. If $c_k = 100$, the grey illustrates the squares that form the staircase, containing values both larger and smaller than 100. Predecessors and successors within each staircase square are shown in red and blue.

4338 obtained by examining their stored ranks. By doing this for the predecessor
4339 and successor we will determine the relationship between $a_i + b_j$ and c_k.

Bibliography

[1] Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In *European Symposium on Algorithms (ESA 2014)*, pages 1–12. Springer, 2014.

[2] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443. IEEE Computer Society, 2014.

[3] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *ICALP (1)*, volume 8572 of *LNCS*, pages 39–51, 2014.

[4] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *STOC*, pages 41–50. ACM, 2015.

[5] Pankaj K. Agarwal and Jivr'i Matouvsek. On range searching with semialgebraic sets. *Discrete & Computational Geometry*, 11:393–418, 1994.

[6] Alfred V. Aho, Kenneth Steiglitz, and Jeffrey D. Ullman. Evaluating polynomials at fixed sets of points. *SIAM Journal on Computing*, 4(4):533–539, 1975.

[7] Oswin Aichholzer, Franz Aurenhammer, and Hannes Krasser. Enumerating order types for small point sets with applications. *Order*, 19(3):265–281, 2002.

[8] Oswin Aichholzer, Franz Aurenhammer, and Hannes Krasser. On the crossing number of complete graphs. In *SoCG*, pages 19–24. ACM, 2002.

[9] Oswin Aichholzer, Jean Cardinal, Vincent Kusters, Stefan Langerman, and Pavel Valtr. Reconstructing point set order types from

radial orderings. *International Journal of Computational Geometry & Applications*, 26(3-4):167–184, 2016.

[10] Oswin Aichholzer, Matias Korman, Alexander Pilz, and Birgit Vogtenhuber. Geodesic order types. *Algorithmica*, 70(1):112–128, 2014.

[11] Oswin Aichholzer and Hannes Krasser. The point set order type data base: A collection of applications and results. In *CCCG*, pages 17–20, 2001.

[12] Oswin Aichholzer and Hannes Krasser. Abstract order type extension and new results on the rectilinear crossing number. In *SoCG*, pages 91–98. ACM, 2005.

[13] Oswin Aichholzer, Vincent Kusters, Wolfgang Mulzer, Alexander Pilz, and Manuel Wettstein. An optimal algorithm for reconstructing point set order types from radial orderings. In *ISAAC*, pages 505–516. Springer, 2015.

[14] Oswin Aichholzer, Tillmann Miltzow, and Alexander Pilz. Extreme point and halving edge search in abstract order types. *Computational Geometry*, 46(8):970–978, 2013.

[15] Nir Ailon and Bernard Chazelle. Lower bounds for linear degeneracy testing. *J. ACM*, 52(2):157–171, 2005.

[16] Noga Alon. The number of polytopes configurations and real matroids. *Mathematika*, 33(1):62–71, 1986.

[17] Greg Aloupis, John Iacono, Stefan Langerman, Özgür Özkan, and Stefanie Wuhrer. The complexity of order type isomorphism. In *SODA*, pages 405–415. SIAM, 2014.

[18] Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *ICALP (1)*, volume 8572 of *LNCS*, pages 114–125, 2014.

[19] Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.

[20] Luis Barba, Jean Cardinal, John Iacono, Stefan Langerman, Aurélien Ooms, and Noam Solomon. Subquadratic algorithms for algebraic 3SUM. *Discrete & Computational Geometry*, 61(4):698–734, Jun 2019. URL: https://doi.org/10.1007/s00454-018-0040-y, doi:10.1007/s00454-018-0040-y.

[21] Gill Barequet and Sariel Har-Peled. Polygon-containment and translational min-Hausdorff-distance between segments sets are 3SUM-hard. *International Journal of Computational Geometry & Applications*, 11(04):465–474, 2001.

[22] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. Computing roadmaps of semi-algebraic sets (extended abstract). In *STOC*, pages 168–173. ACM, 1996.

[23] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer, 2006.

[24] Tim Baumann and Torben Hagerup. Rank-select indices without tears. In *WADS*, 2019.

[25] Michael Ben-Or. Lower bounds for algebraic computation trees. In *STOC*, pages 80–86. ACM, 1983.

[26] Jon Louis Bentley, Dorothea Haken, and James B Saxe. A general method for solving divide-and-conquer recurrences. *ACM SIGACT News*, 12(3):36–44, 1980.

[27] Marshall W. Bern, David Eppstein, Paul E. Plassmann, and F. Frances Yao. Horizon theorems for lines and polygons. In *Discrete and Computational Geometry*, volume 6 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 45–66. DIMACS/AMS, 1990.

[28] Anders Björner, Michel Las Vergnas, Bernd Sturmfels, Neil White, and Günter M Ziegler. Oriented matroids. In *Encyclopedia of Mathematics*, volume 46. Cambridge University Press, 1993.

[29] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.

[30] Jürgen Bokowski, Susanne Mock, and Ileana Streinu. On the Folkman-Lawrence topological representation theorem for oriented matroids of rank 3. *European Journal of Combinatorics*, 22(5):601–615, 2001.

[31] Jürgen Bokowski, Jürgen Richter-Gebert, and Werner Schindler. On the distribution of order types. *Computational Geometry*, 1(3):127–142, 1992.

[32] Peter Brass, William O. J. Moser, and János Pach. *Research problems in discrete geometry*. Springer, 2005.

[33] David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Pătraşcu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014.

[34] Hervé Brönnimann, Bernard Chazelle, and Jirí Matoušek. Product range spaces, sensitive sampling, and derandomization. *SIAM J. Comput.*, 28(5):1552–1575, 1999.

[35] Robert Creighton Buck. Partition of space. *The American Mathematical Monthly*, 50(9):541–544, 1943.

[36] Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1997.

[37] Sergio Cabello, Jean Cardinal, John Iacono, Stefan Langerman, Pat Morin, and Aurélien Ooms. Encoding 3SUM. *ArXiv e-prints*, 2019. arXiv:1903.02645 [cs.DS].

[38] Jean Cardinal, Timothy M. Chan, John Iacono, Stefan Langerman, and Aurélien Ooms. Subquadratic encodings for point configurations. In *Symposium on Computational Geometry*, volume 99 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[39] Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M Jungers, and J Ian Munro. An efficient algorithm for partial order production. *SIAM journal on computing*, 39(7):2927–2940, 2010.

[40] Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M Jungers, and J Ian Munro. Sorting under partial information (without the ellipsoid algorithm). *Combinatorica*, 33(6):655–697, 2013.

[41] Jean Cardinal, John Iacono, and Aurélien Ooms. Solving k-SUM using few linear queries. In *ESA*, volume 57 of *LIPIcs*, pages 25:1–25:17, 2016.

[42] Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *ITCS*, pages 261–270. ACM, 2016.

[43] Bob F Caviness and Jeremy R Johnson. *Quantifier elimination and cylindrical algebraic decomposition*. Springer, 2012.

[44] Timothy M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008.

[45] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010.

[46] Timothy M. Chan. More logarithmic-factor speedups for 3SUM, (median, +)-convolution, and some geometric 3SUM-hard problems. In *SODA*, pages 881–897. SIAM, 2018.

[47] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *STOC*, pages 31–40. ACM, 2015.

[48] Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158, 1993.

[49] Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. A singly exponential stratification scheme for real semi-algebraic varieties and its applications. *Theor. Comput. Sci.*, 84(1):77–105, 1991.

[50] Bernard Chazelle, Leonidas J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25(1):76–90, 1985.

[51] Bernard Chazelle and Jirí Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21(3):579–597, 1996.

[52] Otfried Cheong, Ketan Mulmuley, and Edgar Ramos. Randomization and derandomization. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 895–926. Chapman and Hall/CRC, 2004.

[53] Kenneth L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988.

[54] George E. Collins. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183. Springer, 1975.

[55] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms (3rd Ed.)*. MIT press, 2009.

[56] David Cox, John Little, and Donal O'shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer, 2007.

[57] James H. Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *J. Symb. Comput.*, 5(1/2):29–35, 1988.

[58] David P. Dobkin and Richard J. Lipton. On some generalizations of binary search. In *Symposium on Theory of Computing (STOC 1974)*, pages 310–316, 1974.

[59] David P. Dobkin and Richard J. Lipton. A lower bound of the $\frac{1}{2}n^2$ on linear search programs for the knapsack problem. *J. Comput. Syst. Sci.*, 16(3):413–417, 1978.

[60] Yevgeniy Dodis, Mihai Pătraşcu, and Mikkel Thorup. Changing base without losing space. In *Proceedings of the 42nd ACM Symposium on*

Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010, pages 593–602, 2010.

[61] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10. Springer Science & Business Media, 2012.

[62] Herbert Edelsbrunner, Leonidas J. Guibas, János Pach, Richard Pollack, Raimund Seidel, and Micha Sharir. Arrangements of curves in the plane - topology, combinatorics and algorithms. *Theor. Comput. Sci.*, 92(2):319–336, 1992.

[63] Herbert Edelsbrunner, Joseph O'Rourke, and Raimund Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.*, 15(2):341–363, 1986.

[64] Herbert Edelsbrunner, Raimund Seidel, and Micha Sharir. On the zone theorem for hyperplane arrangements. *SIAM J. Comput.*, 22(2):418–429, 1993.

[65] György Elekes and Lajos Rónyai. A combinatorial problem on polynomials and rational functions. *J. Comb. Theory, Ser. A*, 89(1):1–20, 2000.

[66] György Elekes and Endre Szabó. How to find groups? (and how to use them in Erdős geometry?). *Combinatorica*, 32(5):537–571, 2012.

[67] David Eppstein. *Forbidden Configurations in Discrete Geometry*. Cambridge University Press, 2018.

[68] Jeff Erickson. New lower bounds for Hopcroft's problem. *Discrete & Computational Geometry*, 16(4):389–418, 1996.

[69] Jeff Erickson. Lower bounds for linear satisfiability problems. *Chicago J. Theor. Comput. Sci.*, 1999.

[70] Jeff Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM J. Comput.*, 28(4):1198–1214, 1999.

[71] Hazel Everett, Ferran Hurtado, and Marc Noy. Stabbing information of a simple polygon. *Discrete Applied Mathematics*, 91(1-3):67–82, 1999.

[72] Esther Ezra and Micha Sharir. A nearly quadratic bound for the decision tree complexity of k-sum. In *Symposium on Computational Geometry*, volume 77 of *LIPIcs*, pages 41:1–41:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[73] Stefan Felsner. On the number of arrangements of pseudolines. In *SoCG*, pages 30–37. ACM, 1996.

[74] Stefan Felsner and Pavel Valtr. Coding and counting arrangements of pseudolines. *Discrete & Computational Geometry*, 46(3):405–416, 2011.

[75] Jon Folkman and Jim Lawrence. Oriented matroids. *Journal of Combinatorial Theory, Series B*, 25(2):199–236, 1978.

[76] Hervé Fournier. *Complexité et expressibilité sur les réels*. PhD thesis, École normale supérieure de Lyon, 2001.

[77] Michael L. Fredman. How good is the information theory bound in sorting? *Theor. Comput. Sci.*, 1(4):355–361, 1976.

[78] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with O(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

[79] Ari Freund. Improved subquadratic 3SUM. *Algorithmica*, pages 1–19, 2015.

[80] Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

[81] Omer Gold and Micha Sharir. Improved bounds for 3SUM, k-SUM, and linear degeneracy. In *ESA*, volume 87 of *LIPIcs*, pages 42:1–42:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[82] Jacob E. Goodman. Proof of a conjecture of Burr, Grünbaum, and Sloane. *Discrete Mathematics*, 32(1):27–35, 1980.

[83] Jacob E. Goodman. Pseudoline arrangements. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 97–128. Chapman and Hall/CRC, 2004.

[84] Jacob E. Goodman and Richard Pollack. Proof of Grünbaum's conjecture on the stretchability of certain arrangements of pseudolines. *Journal of Combinatorial Theory, Series A*, 29(3):385–390, 1980.

[85] Jacob E. Goodman and Richard Pollack. Multidimensional sorting. *SIAM Journal on Computing*, 12(3):484–507, 1983.

[86] Jacob E. Goodman and Richard Pollack. Semispaces of configurations, cell complexes of arrangements. *Journal of Combinatorial Theory, Series A*, 37(3):257–293, 1984.

[87] Jacob E. Goodman and Richard Pollack. Upper bounds for configurations and polytopes in $\mathbb{R}^d$. *Discrete & Computational Geometry*, 1:219–227, 1986.

[88] Jacob E. Goodman and Richard Pollack. The complexity of point configurations. *Discrete Applied Mathematics*, 31(2):167–180, 1991.

[89] Jacob E. Goodman and Richard Pollack. Allowable sequences and order types in discrete and computational geometry. In *New Trends in Discrete and Computational Geometry*, pages 103–134. Springer, 1993.

[90] Jacob E. Goodman, Richard Pollack, and Bernd Sturmfels. Coordinate representation of order types requires exponential storage. In *STOC*, pages 405–410. ACM, 1989.

[91] Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4):22:1–22:25, 2018.

[92] Branko Grünbaum. *Convex Polytopes*. Springer, 2005.

[93] Dan Halperin. Arrangements. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 529–562. Chapman and Hall/CRC, 2004.

[94] Heiko Harborth and Meinhard Möller. The Esther Klein problem in the projective plane. *J. Combin. Math. Combin. Comput.*, 15:171–179, 1994.

[95] Joe Harris. *Algebraic geometry: a first course*, volume 133. Springer, 2013.

[96] Robin Hartshorne. *Algebraic geometry*, volume 52. Springer, 1977.

[97] David Haussler and Emo Welzl. ε-nets and simplex range queries. *Discrete & Computational Geometry*, 2(1):127–151, 1987.

[98] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30. ACM, 2015.

[99] Max Hopkins, Daniel M. Kane, and Shachar Lovett. The power of comparisons for actively learning linear classifiers. *ArXiv e-prints*, 2019. arXiv:1907.03816 [cs.LG].

[100] Alfredo Hubard, Luis Montejano, Emiliano Mora, and Andrew Suk. Order types of convex bodies. *Order*, 28(1):121–130, 2011.

[101] Guy Joseph Jacobson. *Succinct static data structures*. PhD thesis, Carnegie Mellon University, 1988.

[102] Daniel M. Kane, Shachar Lovett, and Shay Moran. Generalized comparison trees for point-location problems. In *ICALP*, volume 107 of *LIPIcs*, pages 82:1–82:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[103] Daniel M. Kane, Shachar Lovett, and Shay Moran. Near-optimal linear decision trees for k-sum and related problems. In *STOC*, pages 554–563. ACM, 2018.

[104] Donald E. Knuth. *Axioms and Hulls*, volume 606 of *Lecture Notes in Computer Science*. Springer, 1992.

[105] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *SODA*, pages 1272–1287. SIAM, 2016.

[106] Hsiang-Tsung Kung. Fast evaluation and interpolation. Technical report, Carnegie Mellon University, 1973.

[107] Hsiang-Tsung Kung. A new upper bound on the complexity of derivative evaluation. Technical report, Carnegie Mellon University, 1973.

[108] Friedrich Levi. Die teilung der projektiven ebene durch gerade oder pseudogerade. *Ber. Math.-Phys. Kl. Sächs. Akad. Wiss*, 78:256–267, 1926.

[109] Jirí Matoušek. Range searching with efficient hierarchical cutting. *Discrete & Computational Geometry*, 10:157–182, 1993.

[110] Jirí Matoušek. Approximations and optimal geometric divide-and-conquer. *J. Comput. Syst. Sci.*, 50(2):203–208, 1995.

[111] Jirí Matoušek. Derandomization in computational geometry. *J. Algorithms*, 20(3):545–580, 1996.

[112] Yoshitake Matsumoto, Sonoko Moriyama, Hiroshi Imai, and David Bremner. Matroid enumeration for incidence geometry. *Discrete & Computational Geometry*, 47(1):17–43, 2012.

[113] Stefan Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993. doi:http://dx.doi.org/10.1006/inco.1993.1057.

[114] Friedhelm Meyer auf der Heide. A polynomial linear search algorithm for the n-dimensional knapsack problem. *J. ACM*, 31(3):668–676, 1984.

[115] John Milnor. On the Betti numbers of real varieties. *Proceedings of the American Mathematical Society*, 15(2):275–280, 1964.

[116] Bhubaneswar Mishra. Computational real algebraic geometry. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 743–764. Chapman and Hall/CRC, 2004.

[117] Joseph S. B. Mitchell and Joseph O'Rourke. Computational geometry column 42. *Int. J. Comput. Geometry Appl.*, 11(5):573–582, 2001.

[118] Nikolai E Mnëv. On manifolds of combinatorial types of projective configurations and convex polyhedra. In *Soviet Math. Doklady*, volume 32, pages 335–337, 1985.

[119] Nikolai E Mnëv. The universality theorems on the classification problem of configuration varieties and convex polytopes varieties. In *Topology and geometry—Rohlin seminar*, pages 527–543. Springer, 1988.

[120] H. Nassajian Mojarrad, T. Pham, C. Valculescu, and F. de Zeeuw. Schwartz-Zippel bounds for two-dimensional products. *ArXiv e-prints*, 2016. arXiv:1507.08181 [math.CO].

[121] Jaroslav Nešetřil and Pavel Valtr. A Ramsey property of order types. *Journal of Combinatorial Theory, Series A*, 81(1):88–107, 1998.

[122] János Pach and Micha Sharir. On the number of incidences between points and curves. *Combinatorics, Probability & Computing*, 7(1):121–127, 1998.

[123] János Pach and Micha Sharir. Combinatorial geometry with algorithmic applications – the Alcalá lectures. *AMS, Providence*, 2009.

[124] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

[125] Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Symposium on Theory of Computing (STOC 2010)*, pages 603–610. ACM, 2010.

[126] Mihai Pătraşcu and Ryan Williams. On the possibility of faster SAT algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075, 2010.

[127] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer, 1985.

[128] Michael O. Rabin. Proving simultaneous positivity of linear forms. *J. Comput. Syst. Sci.*, 6(6):639–650, 1972.

[129] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *J. Comput. Syst. Sci.*, 37(2):130–143, 1988.

[130] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct dynamic data structures. In *WADS*, volume 2125 of *Lecture Notes in Computer Science*, pages 426–437. Springer, 2001.

[131] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4):43, 2007.

[132] Orit E. Raz and Micha Sharir. The number of unit-area triangles in the plane: Theme and variations. In *SoCG*, volume 34 of *LIPIcs*, pages 569–583, 2015.

[133] Orit E. Raz, Micha Sharir, and Frank de Zeeuw. Polynomials vanishing on cartesian products: The Elekes-Szabó theorem revisited. In *SoCG*, volume 34 of *LIPIcs*, pages 522–536, 2015.

[134] Orit E. Raz, Micha Sharir, and Frank de Zeeuw. The Elekes-Szabó theorem in four dimensions. *ArXiv e-prints*, 2016. arXiv:1607.03600 [math.CO].

[135] Orit E. Raz, Micha Sharir, and Ilya D. Shkredov. On the number of unit-area triangles spanned by convex grids in the plane. *Comput. Geom.*, 62:25–33, 2017.

[136] Orit E. Raz, Micha Sharir, and József Solymosi. Polynomials vanishing on grids: The Elekes-Rónyai problem revisited. In *SoCG*, page 251. ACM, 2014.

[137] Orit E. Raz, Micha Sharir, and József Solymosi. On triple intersections of three families of unit circles. *Discrete & Computational Geometry*, 54(4):930–953, 2015.

[138] Jürgen Richter. Kombinatorische realisierbarkeitskriterien für orientierte matroide. *Mitt. Math. Sem. Univ. Giessen*, 194:1–112, 1989.

[139] Jürgen Richter-Gebert and Günter M. Ziegler. Oriented matroids. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 129–151. Chapman and Hall/CRC, 2004.

[140] Gerhard Ringel. Teilungen der ebene durch geraden oder topologische geraden. *Mathematische Zeitschrift*, 64(1):79–102, 1956.

[141] Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.

[142] Abraham Seidenberg. Constructions in algebra. *Transactions of the AMS*, 197:273–313, 1974.

[143] Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

[144] Jack Snoeyink. Point location. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 767–785. Chapman and Hall/CRC, 2004.

[145] Joel Spencer. *Ten lectures on the probabilistic method, 2nd Ed.*, volume 64. SIAM, 1994.

[146] J. Michael Steele and Andrew Yao. Lower bounds for algebraic decision trees. *J. Algorithms*, 3(1):1–8, 1982.

[147] William L. Steiger and Ileana Streinu. A pseudo-algorithmic separation of lines from pseudo-lines. *Information Processing Letters*, 53(5):295–299, 1995.

[148] Volker Strassen. Die berechnungskomplexität von elementarsymmetrischen funktionen und von interpolationskoeffizienten. *Numerische Mathematik*, 20(3):238–251, 1973.

[149] Ileana Streinu. Clusters of stars. In *SoCG*, pages 439–441. ACM, 1997.

[150] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*, pages 24–84. Springer Vienna, Vienna, 1998.

[151] René Thom. Sur l'homologie des variétés algébriques. In *Differential and Combinatorial Topology (A Symposium in Honor of Marston Morse)*, pages 255–265, 1965.

[152] VN Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.

[153] Sebastiano Vigna. Broadword implementation of rank/select queries. In *WEA*, volume 5038 of *Lecture Notes in Computer Science*, pages 154–168. Springer, 2008.

[154] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

[155] Andrew Yao. A lower bound to finding convex hulls. *J. ACM*, 28(4):780–787, 1981.

[156] Andrew Chi-Chih Yao. On parallel computation for the knapsack problem. *J. ACM*, 29(3):898–903, 1982.

[157] Andrew Chi-Chih Yao and F. Frances Yao. A general approach to d-dimensional geometric queries. In *STOC*, pages 163–168. ACM, 1985.

[158] Chee K. Yap. Robust geometric computation. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 927–952. Chapman and Hall/CRC, 2004.

[159] David Yun. On square-free decomposition algorithms. In *SYMSACC*, pages 26–35, 1976.

List of Contributions and Open Questions

1 Contribution 40
2 Contribution 41
3 Contribution 43
4 Contribution 43
5 Contribution 43
6 Contribution 43
7 Contribution 45
8 Contribution 45
9 Contribution 45
10 Contribution 45
11 Contribution (Contributions 17 and 18) 45
12 Contribution 46
13 Contribution 46
14 Contribution 47
15 Contribution 48
16 Contribution 48

1 Open Question 52
2 Open Question 52
3 Open Question 52
4 Open Question 52
5 Open Question 53
6 Open Question 53
7 Open Question 53
8 Open Question 53
9 Open Question 53
10 Open Question 54
11 Open Question 54

12 Open Question 54
13 Open Question 54
14 Open Question 54

1 Contribution 77
2 Contribution 85

3 Contribution 94
4 Contribution 101
5 Contribution 104
6 Contribution 111

7 Contribution 131
8 Contribution 131
17 Contribution 131
9 Contribution 142
10 Contribution 142
18 Contribution 142
12 Contribution 149
13 Contribution 149

14 Contribution 155

Solving k-SUM using few linear queries

Jean Cardinal[*1], **John Iacono**[†2], **and Aurélien Ooms**[‡1]

1 **Université libre de Bruxelles (ULB)**
Brussels, Belgium
{jcardin,aureooms}@ulb.ac.be
2 **New York University**
New York, United States of America
icalp2016submission@johniacono.com

Abstract

The k-SUM problem is given n input real numbers to determine whether any k of them sum to zero. The problem is of tremendous importance in the emerging field of complexity theory within P, and it is in particular open whether it admits an algorithm of complexity $O(n^c)$ with $c < \lceil \frac{k}{2} \rceil$. Inspired by an algorithm due to Meiser (1993), we show that there exist linear decision trees and algebraic computation trees of depth $O(n^3 \log^3 n)$ solving k-SUM. Furthermore, we show that there exists a randomized algorithm that runs in $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$ time, and performs $O(n^3 \log^3 n)$ linear queries on the input. Thus, we show that it is possible to have an algorithm with a runtime almost identical (up to the $+8$) to the best known algorithm but for the first time also with the number of queries on the input a polynomial that is independent of k. The $O(n^3 \log^3 n)$ bound on the number of linear queries is also a tighter bound than any known algorithm solving k-SUM, even allowing unlimited total time outside of the queries. By simultaneously achieving few queries to the input without significantly sacrificing runtime vis-à-vis known algorithms, we deepen the understanding of this canonical problem which is a cornerstone of complexity-within-P.

We also consider a range of tradeoffs between the number of terms involved in the queries and the depth of the decision tree. In particular, we prove that there exist $o(n)$-linear decision trees of depth $o(n^4)$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases k-SUM problem, linear decision trees, point location, ε-nets

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

The k-SUM problem is defined as follows: given a collection of real numbers, and a constant k, decide whether any k of them sum to zero. It is a fixed-parameter version of the subset-sum problem, a standard *NP*-complete problem. The k-SUM problem, and in particular the special case of 3SUM, has proved to be a cornerstone of the fine-grained complexity program aiming at the construction of a complexity theory for problems in P. In particular, there are deep connections between the complexity of k-SUM, the Strong

* Supported by the "Action de Recherche Concertée" (ARC) COPHYMA, convention number 4.110.H.000023.
† Research partially completed while on on sabbatical at the Algorithms Research Group of the Département d'Informatique at the Université Libre de Bruxelles with support from a Fulbright Research Fellowship, the Fonds de la Recherche Scientifique — FNRS, and NSF grants CNS-1229185, CCF-1319648, and CCF-1533564.
‡ Supported by the Fund for Research Training in Industry and Agriculture (FRIA).

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Exponential Time Hypothesis [28, 12], and the complexity of many other major problems in P [19, 8, 26, 27, 5, 2, 22, 24, 1, 3, 13].

It has been long known that the k-SUM problem can be solved in time $O(n^{\frac{k}{2}} \log n)$ for even k, and $O(n^{\frac{k+1}{2}})$ for odd k. Erickson [17] proved a near-matching lower bound in the k-linear decision tree model. In this model, the complexity is measured by the depth of a decision tree, every node of which corresponds to a query of the form $q_{i_1} + q_{i_2} + \cdots + q_{i_k} \leq^? 0$, where $q_1, q_2, \ldots, q_n$ are the input numbers. In a recent breakthrough paper, Grønlund and Pettie [22] showed that in the $(2k-2)$-linear decision tree model, where queries test the sign of weighted sums of up to $2k-2$ input numbers, only $O(n^{\frac{k}{2}}\sqrt{\log n})$ queries are required for odd values of k. In particular, there exists a 4-linear decision tree for 3SUM of depth $\tilde{O}(n^{\frac{3}{2}})$ (here the notation $\tilde{O}$ ignores polylogarithmic factors), while every 3-linear decision tree has depth $\Omega(n^2)$ [17]. This indicates that increasing the size of the queries, defined as the maximum number of input numbers involved in a query, can yield significant improvements on the depth of the minimal-height decision tree. Ailon and Chazelle [4] slightly extended the range of query sizes for which a nontrivial lower bound could be established, elaborating on Erickson's technique.

It has been well established that there exist nonuniform polynomial-time algorithms for the subset-sum problem. One of them was described by Meiser [25], and is derived from a data structure for point location in arrangements of hyperplanes using the bottom vertex decomposition. This algorithm can be cast as the construction of a linear decision tree in which the queries have non-constant size.

1.1 Our results.

Our first contribution, in Section 3, is through a careful implementation of Meiser's basic algorithm idea [25] to show the existence of an n-linear decision tree of depth $\tilde{O}(n^3)$ for k-SUM. Although the high-level algorithm itself is not new, we refine the implementation and analysis for the k-SUM problem. Meiser presented his algorithm as a general method of point location in $\mathcal{H}$ given n-dimensional hyperplanes that yielded a depth $\tilde{O}(n^4 \log |\mathcal{H}|)$ linear decision tree; when viewing the k-SUM problem as a point location problem, $\mathcal{H}$ is $O(n^k)$ and thus Meiser's algorithm can be viewed as giving a $\tilde{O}(n^4)$-height linear decision tree. We reduce this height to $\tilde{O}(n^3)$ for the particular hyperplanes that an instance k-SUM problem presents when viewed as a point location problem. While the original algorithm was cast as a nonuniform polynomial-time algorithm, we show that it can be implemented in the linear decision tree model. This first result also implies the existence of nonuniform RAM algorithms with the same complexity, as shown in Appendix B.

There are two subtleties to this result. The first is inherent to the chosen complexity model: even if the number of queries to the input is small (in particular, the degree of the polynomial complexity is invariant on k), the time required to *determine which queries should be performed* may be arbitrary. In a naïve analysis, we show it can be trivially bounded by $\tilde{O}(n^{k+2})$. In Section 4 we present an algorithm to choose which decisions to perform whereby the running time can be reduced to $\tilde{O}(n^{\frac{k}{2}+8})$. Hence, we obtain an $\tilde{O}(n^{\frac{k}{2}+8})$ time randomized algorithm in the RAM model expected to perform $\tilde{O}(n^3)$ linear queries on the input[1].

[1] Grønlund and Pettie [22] mention the algorithms of auf Der Heyde [6] and Meiser [25], and state "(...) it was known that all k-LDT problems can be solved by n-linear decision trees with depth $O(n^5 \log n)$ [25], or with depth $O(n^4 \log(nK))$ if the coefficients of the linear function are integers with absolute value at most K [6]. Unfortunately these decision trees are not efficiently constructible. The time required to

	# blocks	query size	# queries	time
Theorem 3	1	n	$\tilde{O}(n^3)$	$\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$
Theorem 11	b	$k\lceil \frac{n}{b} \rceil$	$\tilde{O}(b^{k-4}n^3)$	$\tilde{O}(b^{\lfloor \frac{k}{2} \rfloor - 9}n^{\lceil \frac{k}{2} \rceil + 8})$
Corollary 12	$b = \frac{k}{\varepsilon}$	εn	$\tilde{O}(n^3)$	$\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$
Corollary 13	$b = O(n^\varepsilon)$	$O(n^{1-\varepsilon})$	$\tilde{O}(n^{3+(k-4)\varepsilon})$	$\tilde{O}(n^{(1+\varepsilon)\frac{k}{2}+8.5})$

Table 1 Complexities of our new algorithms for the k-SUM problem. The query size is the maximum number of elements of the input that can be involved in a single linear query. The number of blocks is a parameter that allows us to change the query size (see Section 5). The origin of the constant in the exponent of the time complexity is due to Lemma 9. We conjecture it can be reduced, though substantial changes in the analysis will likely be needed to do so.

The second issue we address is that the linear queries in the above algorithm may have size n, that is, they may use all the components of the input. The lower bound of Erickson shows that if the queries are of minimal size, the number of queries cannot be a polynomial independent of k such as what we obtain, so non-minimal query size is clearly essential to a drastic reduction in the number of queries needed. This gives rise to the natural question as to what is the relation between query size and number of queries. In particular, one natural question is whether queries of size less than n would still allow the problem to be solved using a polynomial independent of k number of queries. We show that this is possible; in Section 5, we introduce a range of algorithms exhibiting an explicit tradeoff between the number of queries and their size. Using a blocking scheme, we show that we can restrict to εn-linear decision trees, for arbitrary ε. We also give a range of tradeoffs for $n^{1-\varepsilon}$-linear decision trees. Although the proposed algorithms still involve nonconstant-size queries, this is the first time such tradeoffs are explicitly tackled. Table 1 summarizes our results.

2 Definitions and previous work

2.1 Definitions

We consider the k-SUM problem for $k = O(1)$. In what follows, we use the notation $[n] = \{1, 2, \ldots, n\}$.

▶ **Problem** (k-SUM). *Given an input vector $q \in \mathbb{R}^n$, decide whether there exists a k-tuple $(i_1, i_2, \ldots, i_k) \in [n]^k$ such that $\sum_{j=1}^{k} q_{i_j} = 0$.*

The problem amounts to deciding in n-dimensional space, for each hyperplane H of equation $x_{i_1} + x_{i_2} + \cdots + x_{i_k} = 0$, whether q lies on, above, or below H. Hence this indeed amounts to locating the point q in the arrangement formed by those hyperplanes. We emphasize that the set of hyperplanes depends only on k and n and not on the actual input vector q.

Linear degeneracy testing (k-LDT) is a generalization of k-SUM where we have arbitrary rational coefficients[2] and an independent term in the equations of the hyperplanes.

▶ **Problem** (k-LDT). *Given an input vectors $q \in \mathbb{R}^n$ and $\alpha \in \mathbb{Q}^n$ and constant $c \in \mathbb{Q}$ decide whether there exists a k-tuple $(i_1, i_2, \ldots, i_k) \in [n]^k$ such that $c + \sum_{j=1}^{k} \alpha_j q_{i_j} = 0$.*

determine *which* comparisons to make is exponential." We prove that the trees can have depth $\tilde{O}(n^3)$ and that the whole algorithm can run in randomized polynomial-time.

[2] The usual definition of k-LDT allows arbitrary *real* coefficients. However, the algorithm we provide for Lemma 8 needs the vertices of the arrangement of hyperplanes to have rational coordinates.

Our algorithms apply to this more general problem with only minor changes.

The *s-linear decision tree model* is a standard model of computation in which several lower bounds for k-SUM have been proven. In the decision tree model, one may ask well-defined questions to an oracle that are answered "yes" or "no." For s-linear decision trees, a well-defined question consists of testing the sign of a linear function on at most s numbers $q_{i_1}, \ldots, q_{i_s}$ of the input $q_1, \ldots, q_n$ and can be written as

$$c + \alpha_1 q_{i_1} + \cdots + \alpha_s q_{i_s} \overset{?}{\leq} 0$$

Each question is defined to cost a single unit. All other operations can be carried out for free but may not examine the input vector q. We refer to n-linear decision trees simply as linear decision trees.

In this paper, we consider algorithms in the standard integer RAM model, but in which the input $q \in \mathbb{R}^n$ is accessible *only* via a linear query oracle. Hence we are not allowed to manipulate the input numbers directly. The complexity is measured in two ways: by counting the total number of queries, just as in the linear decision tree model, and by measuring the overall running time, taking into account the time required to determine the sequence of linear queries. This two-track computation model, in which the running time is distinguished from the query complexity, is commonly used in results on comparison-based sorting problems where analyses of both runtime and comparisons are of interest (see for instance [29, 10, 11]).

2.2 Previous Results.

The seminal paper by Gajentaan and Overmars [19] showed the crucial role of 3SUM in understanding the complexity of several problems in computational geometry. It is an open question whether an $O(n^{2-\varepsilon})$ algorithm exists for 3SUM. Such a result would have a tremendous impact on many other fundamental computational problems [19, 8, 26, 27, 5, 2, 22, 24, 1, 3, 13]. It has been known for long that k-SUM is $W[1]$-hard. Recently, it was shown to be $W[1]$-complete by Abboud et al. [1].

In Erickson [17], it is shown that we cannot solve 3SUM in subquadratic time in the 3-linear decision tree model:

▶ **Theorem 1** (Erickson [17]). *The optimal depth of a k-linear decision tree that solves the k-LDT problem is $\Theta(n^{\lceil \frac{k}{2} \rceil})$.*

The proof uses an adversary argument which can be explained geometrically. As we already observed, we can solve k-LDT problems by modeling them as point location problems in an arrangement of hyperplanes. Solving one such problem amounts to determining which cell of the arrangement contains the input point. The adversary argument of Erickson [17] is that there exists a cell having $\Omega(n^{\lceil \frac{k}{2} \rceil})$ boundary facets and in this model point location in such a cell requires testing each facet.

Ailon and Chazelle [4] study s-linear decision trees to solve the k-SUM problem when $s > k$. In particular, they give an additional proof for the $\Omega(n^{\lceil \frac{k}{2} \rceil})$ lower bound of Erickson [17] and generalize the lower bound for the s-linear decision tree model when $s > k$. Note that the exact lower bound given by Erickson [17] for $s = k$ is $\Omega((nk^{-k})^{\lceil \frac{k}{2} \rceil})$ while the one given by Ailon and Chazelle [4] is $\Omega((nk^{-3})^{\lceil \frac{k}{2} \rceil})$. Their result improves therefore the lower bound for $s = k$ when k is large. The lower bound they prove for $s > k$ is the following

▶ **Theorem 2** (Ailon and Chazelle [4]). *The depth of an s-linear decision tree solving the k-LDT problem is*

$$\Omega\left(\left(nk^{-3}\right)^{\frac{2k-s}{2\lceil \frac{s-k+1}{2} \rceil}(1-\varepsilon_k)} \right),$$

where $\varepsilon_k > 0$ *tends to* 0 *as* $k \to \infty$.

This lower bound breaks down when $k = \Omega(n^{1/3})$ or $s \geq 2k$ and the cases where $k < 6$ give trivial lower bounds. For example, in the case of 3SUM with $s = 4$ we only get an $\Omega(n)$ lower bound.

As for upper bounds, Baran et al. [7] gave subquadratic Las Vegas algorithms for 3SUM on integer and rational numbers in the circuit RAM, word RAM, external memory, and cache-oblivious models of computation. The idea of their approach is to exploit the parallelism of the models, using linear and universal hashing.

More recently, Grønlund and Pettie [22] proved the existence of a linear decision tree solving the 3SUM problem using a strongly subquadratic number of linear queries. The classical quadratic algorithm for 3SUM uses 3-linear queries while the decision tree of Grønlund and Pettie uses 4-linear queries and requires $O(n^{3/2}\sqrt{\log n})$ of them. Moreover, they show that their decision tree can be used to get better upper bounds for k-SUM when k is odd.

They also provide two subquadratic 3SUM algorithms. A deterministic one running in $O(n^2/(\log n/\log\log n)^{2/3})$ time and a randomized one running in $O(n^2(\log\log n)^2/\log n)$ time with high probability. These results refuted the long-lived conjecture that 3SUM cannot be solved in subquadratic time in the RAM model.

Freund [18] and Gold and Sharir [20] later gave improvements on the results of Grønlund and Pettie [22]. Freund [18] gave a deterministic algorithm for 3SUM running in $O(n^2 \log\log n/\log n)$ time. Gold and Sharir [20] gave another deterministic algorithm for 3SUM with the same running time and shaved off the $\sqrt{\log n}$ factor in the decision tree complexities of 3SUM and k-SUM given by Grønlund and Pettie.

Auf der Heide [6] gave the first point location algorithm to solve the knapsack problem in the linear decision tree model in polynomial time. He thereby answers a question raised by Dobkin and Lipton [15, 16], Yao [30] and others. However, if one uses this algorithm to locate a point in an arbitrary arrangement of hyperplanes the running time is increased by a factor linear in the greatest coefficient in the equations of all hyperplanes. On the other hand, the complexity of Meiser's point location algorithm is polynomial in the dimension, logarithmic in the number of hyperplanes and does not depend on the value of the coefficients in the equations of the hyperplanes. A useful complete description of the latter is also given by Bürgisser et al. [9] (Section 3.4).

3 Query complexity

In this section and the next, we prove the following first result.

▶ **Theorem 3.** *There exist linear decision trees of depth at most* $O(n^3 \log^3 n)$ *solving the* k*-SUM and the* k*-LDT problems. Furthermore, for the two problems there exists an* $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$ *Las Vegas algorithm in the RAM model expected to perform* $O(n^3 \log^3 n)$ *linear queries on the input.*

3.1 Algorithm outline

For a fixed set of hyperplanes $\mathcal{H}$ and given input vertex q in $\mathbb{R}^n$, Meiser's algorithm allows us to determine the cell of the arrangement $\mathcal{A}(\mathcal{H})$ that contains q in its interior (or that *is* q if q is a 0-cell of $\mathcal{A}(\mathcal{H})$), that is, the positions $\sigma(H, q) \in \{-, 0, +\}$ of q with respect to all hyperplanes $H \in \mathcal{H}$. In the k-SUM problem, the set $\mathcal{H}$ is the set of $\Theta(n^k)$ hyperplanes with

equations of the form $x_{i_1} + x_{i_2} + \cdots + x_{i_k} = 0$. These equations can be modified accordingly for k-LDT.

We use standard results on ε-nets. Using a theorem due to Haussler and Welzl [23], it is possible to construct an ε-net $\mathcal{N}$ for the range space defined by hyperplanes and simplices using a random uniform sampling on $\mathcal{H}$.

▶ **Theorem 4** (Haussler and Welzl [23]). *If we choose $O(\frac{n^2}{\varepsilon}\log^2\frac{n}{\varepsilon})$ hyperplanes of $\mathcal{H}$ uniformly at random and denote this selection $\mathcal{N}$ then for any simplex intersected by more than $\varepsilon|\mathcal{H}|$ hyperplanes of $\mathcal{H}$, with high probability, at least one of the intersecting hyperplanes is contained in $\mathcal{N}$.*

The contrapositive states that if no hyperplane in $\mathcal{N}$ intersects a given simplex, then with high probability the number of hyperplanes of $\mathcal{H}$ intersecting the simplex is at most $\varepsilon|\mathcal{H}|$.

We can use this to design a prune and search algorithm as follows: (1) construct an ε-net $\mathcal{N}$, (2) compute the cell C of $\mathcal{A}(\mathcal{N})$ containing the input point q in its interior, (3) construct a simplex S inscribed in C and containing q in its interior, (4) recurse on the hyperplanes of $\mathcal{H}$ intersecting the interior of S.

Proceeding this way with a constant ε guarantees that at most a constant fraction ε of the hyperplanes remains after the pruning step, and thus the cumulative number of queries made to determine the enclosing cell at each step is $O(n^2\log^2 n\log|\mathcal{H}|)$ when done in a brute-force way. However, we still need to explain how to find a simplex S inscribed in C and containing q in its interior. This procedure corresponds to the well-known *bottom vertex decomposition* (or *triangulation*) of a hyperplane arrangement [21, 14].

3.2 Finding a simplex

In order to simplify the exposition of the algorithm, we assume, without loss of generality, that the input numbers q_i all lie in the interval $[-1, 1]$. This assumption is justified by observing that we can normalize all the input numbers by the largest absolute value of a component of q. One can then see that every linear query on the normalized input can be implemented as a linear query on the original input. A similar transformation can be carried out for the k-LDT problem. This allows us to use bounding hyperplanes of equations $x_i = \pm 1, i \in [n]$. We denote by $\mathcal{B}$ this set of hyperplanes. Hence, if we choose a subset $\mathcal{N}$ of the hyperplanes, the input point is located in a bounded cell of the arrangement $\mathcal{A}(\mathcal{N} \cup \mathcal{B})$. Note that $|\mathcal{N} \cup \mathcal{B}| = O(|\mathcal{N}|)$ for all interesting values of ε.

We now explain how to construct S under this assumption. The algorithm can be sketched as follows. (Recall that $\sigma(H, p)$ denotes the relative position of p with respect to the hyperplane H.)

Algorithm 1 (Constructing S).

input A point q in $[-1, 1]^n$, a set $\mathcal{I}$ of hyperplanes not containing q, and a set $\mathcal{E}$ of hyperplanes in general position containing q, such that the cell

$$C = \{\, p\colon\ \sigma(H, p) = \sigma(H, q) \text{ or } \sigma(H, p) = 0 \text{ for all } H \in (\mathcal{I} \cup \mathcal{E})\,\}$$

is a bounded polytope. The value $\sigma(H, q)$ is known for all $H \in (\mathcal{I} \cup \mathcal{E})$.

output A simplex $S \in C$ that contains q in its interior (if it is not a point), and all vertices of which are vertices of C.

0. If $|\mathcal{E}| = n$, return q.

1. Determine a vertex ν of C.

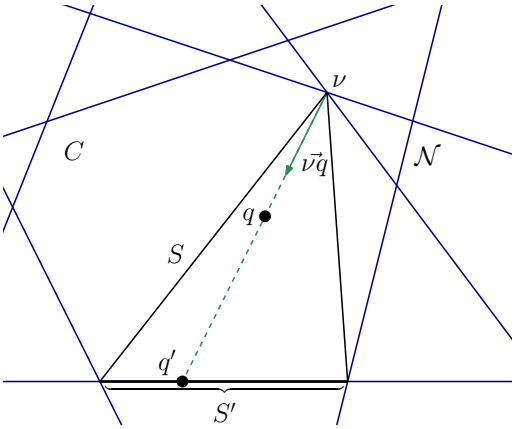


Figure 1 Illustration of a step of Algorithm 1.

2. Let q' be the projection of q along $\vec{\nu q}$ on the boundary of C. Compute $\mathcal{I}_\theta \subseteq \mathcal{I}$, the subset of hyperplanes in $\mathcal{I}$ containing q'. Compute $\mathcal{I}_\tau \subseteq \mathcal{I}_\theta$, a maximal subset of those hyperplanes such that $\mathcal{E}' = \mathcal{E} \cup \mathcal{I}_\tau$ is a set of hyperplanes in general position.
3. Recurse on q', $\mathcal{I}' = \mathcal{I} \setminus \mathcal{I}_\theta$, and $\mathcal{E}'$, and store the result in S'.
4. Return S, the convex hull of $S' \cup \{\nu\}$.

Step **0** is the base case of the recursion: when there is only one point left, just return that point. This step uses no query.

We can solve step **1** by using linear programming with the known values of $\sigma(H, q)$ as linear constraints. We arbitrarily choose an objective function with a gradient non-orthogonal to all hyperplanes in $\mathcal{I}$ and look for the optimal solution. The optimal solution being a vertex of the arrangement, its coordinates are independent of q, and thus this step involves no query at all.

Step **2** prepares the recursive step by finding the hyperplanes containing q'. This can be implemented as a ray-shooting algorithm that performs a number of comparisons between projections of q on different hyperplanes of $\mathcal{I}$ without explicitly computing them. In Appendix A, we prove that all such comparisons can be implemented using $O(|\mathcal{I}|)$ linear queries. Constructing $\mathcal{E}'$ can be done by solving systems of linear equations that do not involve q.

In step **3**, the input conditions are satisfied, that is, $q' \in [-1, 1]^n$, $\mathcal{I}'$ is a set of hyperplanes not containing q', $\mathcal{E}'$ is a set of hyperplanes in general position containing q', C' is a d-cell of C and is thus a bounded polytope. The value $\sigma(H, q')$ differs from $\sigma(H, q)$ only for hyperplanes that have been removed from $\mathcal{I}$, and for those $\sigma(H, q') = 0$, hence we know all necessary values $\sigma(H, q')$ in advance.

Since $|\mathcal{I}'| < |\mathcal{I}|$, $|\mathcal{E}'| > |\mathcal{E}|$, and $|\mathcal{I} \setminus \mathcal{I}'| - |\mathcal{E}' \setminus \mathcal{E}| \geq 0$, the complexity of the recursive call is no more than that of the parent call, and the maximal depth of the recursion is n. Thus, the total number of linear queries made to compute S is $O(n|\mathcal{I}|)$.

Hence given an input point $q \in [-1, 1]$, an arrangement of hyperplanes $\mathcal{A}(\mathcal{N})$, and the value of $\sigma(H, q)$ for all $H \in (\mathcal{N} \cup \mathcal{B})$, we can compute the desired simplex S by running Algorithm 1 on q, $\mathcal{I} = \{H \in (\mathcal{N} \cup \mathcal{B}) \colon \sigma(H, q) \neq 0\}$, and $\mathcal{E} \subseteq (\mathcal{N} \cup \mathcal{B}) \setminus \mathcal{I}$. This uses $O(n^3 \log^2 n)$ linear queries. Figure 1 illustrates a step of the algorithm.

3.3 Assembling the pieces

Let us summarize the algorithm

Algorithm 2.

input $q \in [-1,1]^n$

1. Pick $O(n^2 \log^2 n)$ hyperplanes of $\mathcal{H}$ at random and locate q in this arrangement. Call C the cell containing q.
2. Construct the simplex S containing q and inscribed in C, using Algorithm 1.
3. For every hyperplane of $\mathcal{H}$ containing S, output a solution.
4. Recurse on hyperplanes of $\mathcal{H}$ intersecting the interior of S.

The query complexity of step **1** is $O(n^2 \log^2 n)$, and that of step **2** is $O(n^3 \log^2 n)$. Steps **3** and **4** do not involve any query at all. The recursion depth is $O(\log |\mathcal{H}|)$, with $|\mathcal{H}| = O(n^k)$, hence the total query complexity of this algorithm is $O(n^3 \log^3 n)$. This proves the first part of Theorem 3.

We can also consider the overall complexity of the algorithm in the RAM model, that is, taking into account the steps that do not require any query, but for which we still have to process the set $\mathcal{H}$. Note that the complexity bottleneck of the algorithm are steps **3**-**4**, where we need to prune the list of hyperplanes according to their relative positions with respect to S. For this purpose, we simply maintain explicitly the list of all hyperplanes, starting with the initial set corresponding to all k-tuples. Then the pruning step can be performed by looking at the position of each vertex of S relative to each hyperplane of $\mathcal{H}$. Because in our case hyperplanes have only k nonzero coefficients, this uses a number of integer arithmetic operations on $\tilde{O}(n)$ bits integers that is proportional to the number of vertices times the number of hyperplanes. (For the justification of the bound on the number of bits needed to represent vertices of the arrangement see Appendix D.) Since we recurse on a fraction of the set, the overall complexity is $\tilde{O}(n^2|\mathcal{H}|) = \tilde{O}(n^{k+2})$. The next section is devoted to improving this running time.

4 Time complexity

Proving the second part of Theorem 3 involves efficient implementations of the two most time-consuming steps of Algorithm 2. In order to efficiently implement the pruning step, we define an intermediate problem, that we call the *double k-SUM* problem.

▶ **Problem** (double k-SUM)**.** *Given two vectors $\nu_1, \nu_2 \in [-1,1]^n$, where the coordinates of ν_i can be written down as fractions whose numerator and denominator lie in the interval $[-M, M]$, enumerate all $i \in [n]^k$ such that*

$$\left(\sum_{j=1}^{k} \nu_{1,i_j}\right)\left(\sum_{j=1}^{k} \nu_{2,i_j}\right) < 0.$$

In other words, we wish to list all hyperplanes of $\mathcal{H}$ intersecting the open line segment $\nu_1\nu_2$. We give an efficient output-sensitive algorithm for this problem.

▶ **Lemma 5.** *The double k-SUM problem can be solved in time $O(n^{\lceil \frac{k}{2} \rceil} \log n \log M + Z)$, where Z is the size of the solution.*

Proof. If k is even, we consider all possible $\frac{k}{2}$-tuples of numbers in ν_1 and ν_2 and sort their sums in increasing order. This takes time $O(n^{\frac{k}{2}} \log n)$ and yields two permutations π_1 and π_2 of $[n^{\frac{k}{2}}]$. If k is odd, then we sort both the $\lceil \frac{k}{2} \rceil$-tuples and the $\lfloor \frac{k}{2} \rfloor$-tuples. For simplicity, we will only consider the even case in what follows. The odd case carries through.

We let $N = n^{\frac{k}{2}}$. For $i \in [N]$ and $m \in \{1, 2\}$, let $\Sigma_{m,i}$ be the sum of the $\frac{k}{2}$ components of the ith $\frac{k}{2}$-tuple in ν_m, in the order prescribed by π_m.

We now consider the two $N \times N$ matrices M_1 and M_2 giving all possible sums of two $\frac{k}{2}$-tuples, for both ν_1 with the ordering π_1 and ν_2 with the ordering π_2.

We first solve the k-SUM problem on ν_1, by finding the sign of all pairs $\Sigma_{1,i} + \Sigma_{1,j}$, $i, j \in [N]$. This can be done in time $O(N)$ by parsing the matrix M_1, just as in the standard k-SUM algorithm. We do the same with M_2.

The set of all indices $i, j \in [N]$ such that $\Sigma_{1,i} + \Sigma_{1,j}$ is positive forms a staircase in M_1. We sweep M_1 column by column in order of increasing $j \in [N]$, in such a way that the number of indices i such that $\Sigma_{1,i} + \Sigma_{1,j} > 0$ is growing. For each new such value i that is encountered during the sweep, we insert the corresponding $i' = \pi_2(\pi_1^{-1}(i))$ in a balanced binary search tree.

After each sweep step in M_1 — that is, after incrementing j and adding the set of new indices i' in the tree — we search the tree to identify all the indices i' such that $\Sigma_{2,i'} + \Sigma_{2,j'} < 0$, where $j' = \pi_2(\pi_1^{-1}(j))$. Since those indices form an interval in the ordering π_2 when restricted to the indices in the tree, we can search for the largest i'_0 such that $\Sigma_{2,i'_0} < -\Sigma_{2,j'}$ and retain all indices $i' \leq i'_0$ that are in the tree. If we denote by z the number of such indices, this can be done in $O(\log N + z) = O(\log n + z)$ time. Now all the pairs i', j' found in this way are such that $\Sigma_{1,i} + \Sigma_{1,j}$ is positive and $\Sigma_{2,i'} + \Sigma_{2,j'}$ is negative, hence we can output the corresponding k-tuples. To get all the pairs i', j' such that $\Sigma_{1,i} + \Sigma_{1,j}$ is negative and $\Sigma_{2,i'} + \Sigma_{2,j'}$ positive, we repeat the sweeping algorithm after swapping the roles of ν_1 and ν_2.

Every matching k-tuple is output exactly once, and every $\frac{k}{2}$-tuple is inserted at most once in the binary search tree. Hence the algorithm runs in the claimed time.

Note that we only manipulate rational numbers that are the sum of at most k rational numbers of size $O(\log M)$. ◀

Now observe that a hyperplane intersects the interior of a simplex if and only if it intersects the interior of one of its edges. Hence given a simplex S we can find all hyperplanes of $\mathcal{H}$ intersecting its interior by running the above algorithm $\binom{n}{2}$ times, once for each pair of vertices (ν_1, ν_2) of S, and take the union of the solutions. The overall running time for this implementation will therefore be $\tilde{O}(n^2(n^{\lceil \frac{k}{2} \rceil} \log M + Z))$, where Z is at most the number of intersecting hyperplanes and M is to be determined later. This provides an implementation of the pruning step in Meiser's algorithm, that is, step **4** of Algorithm 2.

▶ **Corollary 6.** *Given a simplex S, we can compute all k-SUM hyperplanes intersecting its interior in $\tilde{O}(n^2(n^{\lceil \frac{k}{2} \rceil} \log M + Z))$ time, where M is proportional to the number of bits necessary to represent S.*

In order to detect solutions in step **3** of Algorithm 2, we also need to be able to quickly solve the following problem.

▶ **Problem** (multiple k-SUM). *Given d points $\nu_1, \nu_2, \ldots, \nu_d \in \mathbb{R}^n$, where the coordinates of ν_i can be written down as fractions whose numerator and denominator lie in the interval $[-M, M]$, decide whether there exists a hyperplane with equation of the form $x_{i_1} + x_{i_2} + \cdots + x_{i_k} = 0$ containing all of them.*

Here the standard k-SUM algorithm can be applied, taking advantage of the fact that the coordinates lie in a small discrete set.

▶ **Lemma 7.** *k-SUM on n integers $\in [-V, V]$ can be solved in time $\tilde{O}(n^{\lceil \frac{k}{2} \rceil} \log V)$.*

▶ **Lemma 8.** *Multiple k-SUM can be solved in time $\tilde{O}(dn^{\lceil \frac{k}{2} \rceil + 2} \log M)$.*

Proof. Let $\mu_{i,j}$ and $\delta_{i,j}$ be the numerator and denominator of $\nu_{i,j}$ when written as an irreducible fraction. We define

$$\zeta_{i,j} = \nu_{i,j} \prod_{(i,j) \in [d] \times [n]} \delta_{i,j} = \frac{\mu_{i,j} \prod_{(i',j') \in [d] \times [n]} \delta_{i',j'}}{\delta_{i,j}}.$$

By definition $\zeta_{i,j}$ is an integer and its absolute value is bounded by $U = M^{n^2}$, that is, it can be represented using $O(n^2 \log M)$ bits. Moreover, if one of the hyperplanes contains the point $(\zeta_{i,1}, \zeta_{i,2}, \ldots, \zeta_{i,n})$, then it contains ν_i. Construct n integers of $O(dn^2 \log M)$ bits that can be written $\zeta_{1,j} + U, \zeta_{2,j} + U, \ldots, \zeta_{d,j} + U$ in base $2Uk + 1$. The answer to our decision problem is "yes" if and only if there exists k of those numbers whose sum is $kU, kU, \ldots, kU$. We simply subtract the number $U, U, \ldots, U$ to all n input numbers to obtain a standard k-SUM instance on n integers of $O(dn^2 \log M)$ bits. ◀

We now have efficient implementations of steps **3** and **4** of Algorithm 2 and can proceed to the proof of the second part of Theorem 3.

Proof. The main idea consists of modifying the first iteration of Algorithm 2, by letting $\varepsilon = n^{-\frac{k}{2}}$. Hence we pick a random subset $\mathcal{N}$ of $O(n^{k/2+2} \log^2 n)$ hyperplanes in $\mathcal{H}$ and use this as an ε-net. This can be done efficiently, as shown in Appendix C.

Next, we need to locate the input q in the arrangement induced by $\mathcal{N}$. This can be done by running Algorithm 2 on the set $\mathcal{N}$. From the previous considerations on Algorithm 2, the running time of this step is

$$O(n|\mathcal{N}|) = \tilde{O}(n^{k/2+4}),$$

and the number of queries is $O(n^3 \log^3 n)$.

Then, in order to be able to prune the hyperplanes in $\mathcal{H}$, we have to compute a simplex S that does not intersect any hyperplane of $\mathcal{N}$. For this, we observe that the above call to Algorithm 2 involves computing a sequence of simplices for the successive pruning steps. We save the description of those simplices. Recall that there are $O(\log n)$ of them, all of them contain the input q and have vertices coinciding with vertices of the original arrangement $\mathcal{A}(\mathcal{H})$. In order to compute a simplex S avoiding all hyperplanes of $\mathcal{N}$, we can simply apply Algorithm 1 on the set of hyperplanes bounding the intersection of these simplices. The running time and number of queries for this step are bounded respectively by $n^{O(1)}$ and $O(n^2 \log n)$.

Note that the vertices of S are not vertices of $\mathcal{A}(\mathcal{H})$ anymore. However, their coordinates lie in a finite set (see Appendix D)

▶ **Lemma 9.** *Vertices of S have rational coordinates whose fraction representations have their numerators and denominators absolute value bounded by $O(C^{4n^5} n^{4n^5+2n^3+n})$, where C is a constant.*

We now are in position to perform the pruning of the hyperplanes in $\mathcal{H}$ with respect to S. The number of remaining hyperplanes after the pruning is at most $\varepsilon n^k = O(n^{k/2})$. Hence from Corollary 6, the pruning can be performed in time proportional $\tilde{O}(n^{\lceil k/2 \rceil + 7})$.

Similarly, we can detect any hyperplane of $\mathcal{H}$ containing S using the result of Lemma 8 in time $\tilde{O}(n^{\lceil k/2 \rceil + 8})$. Note that those last two steps do not require any query.

Finally, it remains to detect any solution that may lie in the remaining set of hyperplanes of size $O(n^{k/2})$. We can again fall back on Algorithm 2, restricted to those hyperplanes. The running time is $\tilde{O}(n^{k/2+2})$, and the number of queries is still $O(n^3 \log^3 n)$.

Overall, the maximum running time of a step is $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$, while the number of queries is always bounded by $O(n^3 \log^3 n)$.

◀

5 Query size

In this section, we consider a simple blocking scheme that allows us to explore a tradeoff between the number of queries and the size of the queries.

▶ **Lemma 10.** *For any integer $b > 0$, an instance of the k-SUM problem on $n > b$ numbers can be split into $O(b^{k-1})$ instances on at most $k\lceil \frac{n}{b} \rceil$ numbers, so that every k-tuple forming a solution is found in exactly one of the subproblems. The transformation can be carried out in time $O(n \log n + b^{k-1})$.*

Proof. Given an instance on n numbers, we can sort them in time $O(n \log n)$, then partition the sorted sequence into b consecutive blocks $B_1, B_2, \ldots, B_b$ of equal size. This partition can be associated with a partition of the real line into b intervals, say $I_1, I_2, \ldots, I_b$. Now consider the partition of $\mathbb{R}^k$ into grid cells defined by the kth power of the partition $I_1, I_2, \ldots, I_b$. The hyperplane of equation $x_1 + x_2 + \cdots + x_k = 0$ hits $O(b^{k-1})$ such grid cells. Each grid cell $I_{i_1} \times I_{i_2} \times \cdots \times I_{i_k}$ corresponds to a k-SUM problem on the numbers in the set $B_{i_1} \cup B_{i_2} \cup \ldots \cup B_{i_k}$ (note that the indices i_j need not be distinct). Hence each such instance has size at most $k\lceil \frac{n}{b} \rceil$. ◀

Combining Lemma 10 and Theorem 3 directly yields the following.

▶ **Theorem 11.** *For any integer $b > 0$, there exists a $k\lceil \frac{n}{b} \rceil$-linear decision tree of depth $\tilde{O}(b^{k-4}n^3)$ solving the k-SUM problem. Moreover, this decision tree can be implemented as an $\tilde{O}(b^{\lfloor \frac{k}{2} \rfloor - 9} n^{\lceil \frac{k}{2} \rceil + 8})$ Las Vegas algorithm.*

The following two corollaries are obtained by taking $b = \frac{k}{\varepsilon}$, and $b = O(n^{\varepsilon})$, respectively

▶ **Corollary 12.** *For any constant $\varepsilon > 0$ such that $b = \frac{k}{\varepsilon}$ and $\frac{n}{b}$ are positive integers, there exists an εn-linear decision tree of depth $\tilde{O}(n^3)$ solving the k-SUM problem. Moreover, this decision tree can be implemented as an $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$ Las Vegas algorithm.*

▶ **Corollary 13.** *For any ε such that $0 < \varepsilon < 1$, there exists an $O(n^{1-\varepsilon})$-linear decision tree of depth $\tilde{O}(n^{3+(k-4)\varepsilon})$ solving the k-SUM problem. Moreover, this decision tree can be implemented as an $\tilde{O}(n^{(1+\varepsilon)\frac{k}{2}+8.5})$ Las Vegas algorithm.*

Note that the latter query complexity improves on $\tilde{O}(n^{\frac{k}{2}})$ whenever $\varepsilon < \frac{1}{2}$ and $k \geq \frac{3-4\varepsilon}{1/2-\varepsilon}$. Hence for instance, we obtain an $O(n^{\frac{3}{4}})$-linear decision tree of depth $\tilde{O}(n^4)$ for the 8SUM problem, and $o(n)$-linear decision trees of depth $o(n^4)$ for any k.

References

1 Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. 2015. arXiv:1311.3054 [cs.CC].

2 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014.

3 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Symposium on Theory of Computing (STOC 2015)*, pages 41–50, 2015.

4 Nir Ailon and Bernard Chazelle. Lower bounds for linear degeneracy testing. *J. ACM*, 52(2):157–171, 2005.

5 Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, volume 8572 of *Lecture Notes in Computer Science*, pages 114–125. Springer, 2014.

6 Meyer Friedhelm auf der Heide. A polynomial linear search algorithm for the n-dimensional knapsack problem. *J. ACM*, 31(3):668–676, 1984.

7 Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.

8 Gill Barequet and Sariel Har-Peled. Polygon-containment and translational min-hausdorff-distance between segment sets are 3SUM-hard. In *Symposium on Discrete Algorithms (SODA 1999)*, pages 862–863, 1999.

9 Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1997.

10 Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M Jungers, and J Ian Munro. An efficient algorithm for partial order production. *SIAM journal on computing*, 39(7):2927–2940, 2010.

11 Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M Jungers, and J Ian Munro. Sorting under partial information (without the ellipsoid algorithm). *Combinatorica*, 33(6):655–697, 2013.

12 Marco Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mikhailin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. *Electronic Colloquium on Computational Complexity (ECCC 2015)*, 22:148, 2015.

13 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Symposium on Theory of Computing (STOC 2015)*, pages 31–40. ACM, 2015.

14 Kenneth L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988.

15 David P. Dobkin and Richard J. Lipton. On some generalizations of binary search. In *Symposium on Theory of Computing (STOC 1974)*, pages 310–316, 1974.

16 David P. Dobkin and Richard J. Lipton. A lower bound of the $\frac{1}{2}n^2$ on linear search programs for the knapsack problem. *J. Comput. Syst. Sci.*, 16(3):413–417, 1978.

17 Jeff Erickson. Lower bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science*, 1999.

18 Ari Freund. Improved subquadratic 3SUM. *Algorithmica*, 2015. To appear.

19 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

20 O. Gold and M. Sharir. Improved bounds for 3SUM, k-SUM, and linear degeneracy. *ArXiv e-prints*, 2015. arXiv:1512.05279 [cs.DS].

21 Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry, Second Edition*. Chapman and Hall/CRC, 2004.

22 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. In *Foundations of Computer Science (FOCS 2014)*, pages 621–630. IEEE, 2014.

23 David Haussler and Emo Welzl. ϵ-nets and simplex range queries. *Discrete & Computational Geometry*, 2(1):127–151, 1987.

24 T. Kopelowitz, S. Pettie, and E. Porat. Higher lower bounds from the 3SUM conjecture. *ArXiv e-prints*, 2014. arXiv:1407.6756 [cs.DS].

25 S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.

26 Joseph S. B. Mitchell and Joseph O'Rourke. Computational geometry column 42. *Int. J. Comput. Geometry Appl.*, 11(5):573–582, 2001.

27 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Symposium on Theory of Computing (STOC 2010)*, pages 603–610, 2010.

28 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *Symposium on Discrete Algorithms (SODA 2010)*, pages 1065–1075, 2010.

29 William L. Steiger and Ileana Streinu. A pseudo-algorithmic separation of lines from pseudo-lines. *Information Processing Letters*, 53(5):295–299, 1995.

30 Andrew Chi-Chih Yao. On parallel computation for the knapsack problem. *J. ACM*, 29(3):898–903, 1982.

A Keeping queries linear in Algorithm 1

In Algorithm 1, we want to ensure that the queries we make in step **2** are linear and that the queries we will make in the recursion step remain linear too.

▶ **Lemma 14.** *Algorithm 1 can be implemented so that it uses $O(n|I|)$ linear queries.*

Proof. Let us first analyze what the queries of step **2** look like. In addition to the input point q we are given a vertex ν and we want to find the projection q' of q in direction $\vec{\nu q}$ on the hyperplanes of $\mathcal{I}_\theta$. Let the equation of H_i be $\Pi_i(x) = c_i + d_i \cdot x = 0$ where c_i is a scalar and d_i is a vector. The projection of q along $\vec{\nu q}$ on a hyperplane H_i can thus be written[3] $\rho(q, \nu, H_i) = \nu + \lambda_i \vec{\nu q}$ such that $\Pi_i(\nu + \lambda_i \vec{\nu q}) = c_i + d_i \cdot \nu + \lambda_i d_i \cdot \vec{\nu q} = 0$. Computing the closest hyperplane amounts to finding $\lambda_\theta = \min_{\lambda_i > 0} \lambda_i$. Since $\lambda_i = -\frac{c_i + d_i \cdot \nu}{d_i \cdot \vec{\nu q}}$ we can test whether $\lambda_i > 0$ using the linear query[4] $-\frac{d_i \cdot \vec{\nu q}}{c_i + d_i \cdot \nu} >^? 0$. Moreover, if $\lambda_i > 0$ and $\lambda_j > 0$ we can test whether $\lambda_i < \lambda_j$ using the linear query $\frac{d_i \cdot \vec{\nu q}}{c_i + d_i \cdot \nu} <^? \frac{d_j \cdot \vec{\nu q}}{c_j + d_j \cdot \nu}$. Step **2** can thus be achieved using $O(1)$ $(2k)$-linear queries per hyperplane of $\mathcal{N}$.

In step **4**, the recursive step is carried out on $q' = \nu + \lambda_\theta \vec{\nu q} = \nu - \frac{c_\theta + d_\theta \cdot \nu}{d_\theta \cdot \vec{\nu q}} \vec{\nu q}$ hence comparing λ'_i to 0 amounts to performing the query $-\frac{d_i \cdot \vec{\nu q'}}{c_i + d_i \cdot \nu'} >^? 0$, which is not linear in q. The same goes for comparing λ'_i to λ'_j with the query $\frac{d_i \cdot \vec{\nu q'}}{c_i + d_i \cdot \nu'} <^? \frac{d_j \cdot \vec{\nu q'}}{c_j + d_j \cdot \nu'}$.

However, we can multiply both sides of the inequality test by $d_\theta \vec{\nu q}$ to keep the queries linear as shown below. We must be careful to take into account the sign of the expression $d_\theta \vec{\nu q}$, this costs us one additional linear query.

[3] Note that we project from ν instead of q. We are allowed to do this since $\nu + \lambda_i \vec{\nu q} = q + (\lambda_i - 1)\vec{\nu q}$ and there is no hyperplane separating q from ν.

[4] Note that if $c_i + d_i \cdot \nu = 0$ then $\lambda_i = 0$, we can check this beforehand for free.

This trick can be used at each step of the recursion. Let $q^{(0)} = q$, then we have

$$q^{(s+1)} = \nu^{(s)} - \frac{c_{\theta_s} + d_{\theta_s} \cdot \nu^{(s)}}{d_{\theta_s} \cdot \vec{\nu q}^{(s)}} \vec{\nu q}^{(s)}$$

and $(d_{\theta_s} \cdot \vec{\nu q}^{(s)})q^{(s+1)}$ yields a vector whose components are linear in $q^{(s)}$. Hence, $(\prod_{k=0}^{s} d_{\theta_k} \cdot \vec{\nu q}^{(k)})q^{(s+1)}$ yields a vector whose components are linear in q, and for all pairs of vectors d_i and $\nu^{(s+1)}$ we have that $(\prod_{k=0}^{s} d_{\theta_k} \cdot \vec{\nu q}^{(k)})(d_i \cdot \vec{\nu q}^{(s+1)})$ is linear in q.

Hence at the sth recursive step of the algorithm, we will perform at most $|\mathcal{N}|$ linear queries of the type

$$-\left(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)}\right) \frac{d_i \cdot \vec{\nu q}^{(s)}}{c_i + d_i \cdot \nu^{(s)}} \overset{?}{>} 0$$

$|\mathcal{N}| - 1$ linear queries of the type

$$\left(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)}\right) \frac{d_i \cdot \vec{\nu q}^{(s)}}{c_i + d_i \cdot \nu^{(s)}} \overset{?}{<} \left(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)}\right) \frac{d_j \cdot \vec{\nu q}^{(s)}}{c_j + d_j \cdot \nu^{(s)}}$$

and a single linear query of the type

$$d_{\theta_{s-1}} \cdot \vec{\nu q}^{(s-1)} \overset{?}{<} 0.$$

In order to detect all hyperplanes H_i such that $\lambda_i = \lambda_\theta$ we can afford to compute the query $f(q) > g(q)$ for all query $f(q) < g(q)$ that we issue, and vice versa.

Note that, without further analysis, the queries can become n-linear as soon as we enter the $\frac{n}{k}^{\text{th}}$ recursive step. ◀

B Algebraic computation trees

We consider *algebraic computation trees*, whose internal nodes are labeled with arithmetic $(r \leftarrow o_1 \operatorname{op} o_2, \operatorname{op} \in \{+, -, \times, \div\})$ and branching $(z : 0)$ operations. We say that an algebraic computation tree T *realizes* an algorithm A if the paths from the root to the leaves of T correspond to the execution paths of A on all possible inputs $q \in \mathbb{R}^n$, where n is fixed. A leaf is labeled with the output of the corresponding execution path of A. Such a tree is *well-defined* if any internal node labeled $r \leftarrow o_1 \operatorname{op} o_2$ has outdegree 1 and is such that either $o_k = q_i$ for some i or there exists an ancestor $o_k \leftarrow x \operatorname{op} y$ of this node, and any internal node labeled $z : 0$ has outdegree 3 and is such that either $z = q_i$ for some i or there exists an ancestor $z \leftarrow x \operatorname{op} y$ of this node. In the algebraic computation tree model, we define the complexity $f(n)$ of an algorithm A to be the minimum depth of a well-defined computation tree that realizes A for inputs of size n.

In the algebraic computation tree model, we only count the operations that involve the input, that is, members of the input or results of previous operations involving the input. The following theorem follows immediately from the analysis of the linearity of queries

▶ **Theorem 15.** *The algebraic computation tree complexity of k-LDT is $\tilde{O}(n^3)$.*

Proof. We go through each step of Algorithm 2. Indeed, each k-linear query of step **1** can be implemented as $O(k)$ arithmetic operations, so step **1** has complexity $O(|\mathcal{N}|)$. The construction of the simplex in step **2** must be handled carefully. What we need to show is that each n-linear query we use can be implemented using $O(k)$ arithmetic operations. It is not difficult to see from the expressions given in Appendix A that a constant number

of arithmetic operations and dot products suffice to compute the queries. A dot product in this case involves a constant number of arithmetic operations because the d_i are such that they each have exactly k non-zero components. The only expression that involves a non-constant number of operations is the product $\prod_{k=0}^{s} d_{\theta_k} \cdot \vec{\nu q}^{(k)}$, but this is equivalent to $(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)})(d_{\theta_s} \cdot \vec{\nu q}^{(s)})$ where the first factor has already been computed during a previous step and the second factor is of constant complexity. Since each query costs a constant number of arithmetic operations and branching operations, step **2** has complexity $O(n|\mathcal{N}|)$. Finally, steps **3** and **4** are free since they do not involve the input. The complexity of Algorithm 2 in this model is thus also $O(n^3 \log^2 n \log |\mathcal{H}|)$. ◀

C Uniform random sampling

Theorem 4 requires us to pick a sample of the hyperplanes uniformly at random. Actually the theorem is a little stronger; we can draw each element of $\mathcal{N}$ uniformly at random, only keeping distinct elements. This is not too difficult to achieve for k-LDT when the $\alpha_i, i \in [k]$ are all distinct: to pick a hyperplane of the form $\alpha_0 + \alpha_1 x_{i_1} + \alpha_2 x_{i_2} + \cdots + \alpha_k x_{i_k} = 0$ uniformly at random, we can draw each $i_j \in [n]$ independently and there are n^k possible outcomes. However, in the case of k-SUM, we only have $\binom{n}{k}$ distinct hyperplanes. A simple dynamic programming approach solves the problem for k-SUM. For k-LDT we can use the same approach, once for each class of equal α_i.

▶ **Lemma 16.** *Given $n \in \mathbb{N}$ and $(\alpha_0, \alpha_1, \ldots, \alpha_k) \in \mathbb{R}^{k+1}$, a uniform random sample $\mathcal{N}$ of hyperplanes in $\mathbb{R}^n$ with equations of the form $\alpha_0 + \alpha_1 x_{i_1} + \alpha_2 x_{i_2} + \cdots + \alpha_k x_{i_k} = 0$ can be computed in time $O(k|\mathcal{N}|)$ and preprocessing time $O(kn)$.*

Proof. We want to pick an assignment $a = \{ (\alpha_1, x_{i_1}), (\alpha_2, x_{i_2}), \ldots, (\alpha_k, x_{i_k}) \}$ uniformly at random. Note that all x_i are distinct while the α_j can be equal.

Without loss of generality, suppose $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_k$. There is a bijection between assignments and lexicographically sorted k-tuples $((\alpha_1, x_{i_1}), (\alpha_2, x_{i_2}), \ldots, (\alpha_k, x_{i_k}))$.

Observe that x_{i_j} can be drawn independently of $x_{i_{j'}}$ whenever $\alpha_j \neq \alpha_{j'}$. Hence, it suffices to generate a lexicographically sorted $|\chi|$-tuple of x_i for each class χ of equal α_i.

Let $\omega(m, l)$ denote the number of lexicographically sorted l-tuples, where each element comes from a set of m distinct x_i. We have

$$\omega(m,l) = \begin{cases} 1 & \text{if } l = 0 \\ \sum_{i=1}^{m} \omega(i, l-1) & \text{otherwise.} \end{cases}$$

To pick such a tuple $(x_{i_1}, x_{i_2}, \ldots, x_{i_l})$ uniformly at random we choose $x_{i_l} = x_o$ with probability

$$P(x_{i_l} = x_o) = \begin{cases} 0 & \text{if } o > m \\ \frac{\omega(o, l-1)}{\omega(m,l)} & \text{otherwise} \end{cases}$$

that we append to a prefix $(l-1)$-tuple (apply the procedure recursively), whose elements come from a set of o symbols. If $l = 0$ we just return the empty tuple.

Obviously, the probability for a given l-tuple to be picked is equal to $\frac{1}{\omega(m,l)}$.

Let X denote the partiton of the α_i into equivalence classes, then the number of assignments is equal to $\prod_{\chi \in X} \omega(n, |\chi|)$. (Note that for k-SUM this is simply $\omega(n, k)$ since there is only a single class of equivalence.) For each equivalence class χ we draw independently a lexicographically sorted $|\chi|$-tuple on n symbols using the procedure above. This yields

a given assignment with probability $\frac{1}{\prod_{\chi \in X} \omega(n,|\chi|)}$. Hence, this corresponds to a uniform random draw over the assignments.

For given n and k, there are at most nk values $\omega(m,l)$ to compute, and for a given k-LDT instance, it must be computed only once. Once those values have been computed, making a random draw takes time $O(k)$.

◀

D Proof of Lemma 9

▶ **Theorem 17** (Cramer's rule). *If a system of n linear equations for n unknowns, represented in matrix multiplication form $Ax = b$, has a unique solution $x = (x_1, x_2, \ldots, x_n)^T$ then, for all $i \in [n]$,*

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where A_i is A with the ith column replaced by the column vector b.

▶ **Lemma 18.** *The absolute value of the determinant of an $n \times n$ matrix M with integer entries is $O((Cn)^n)$ where C is the maximum absolute value in M.*

Proof. The determinant of such a matrix is the sum of $2n!$ integers with maximum absolute value C^n. ◀

▶ **Lemma 19.** *The determinant of an $n \times n$ matrix M with rational entries can be represented as a fraction whose numerators and denominators absolute values are bounded by $O((ND^{n-1}n)^n)$ and $O(D^{n^2})$ respectively, where N and D are respectively the maximum absolute value of a numerator and a denominator.*

Proof. Multiply each row M_i of M by $\prod_j d_{i,j}$. Apply Lemma 18. ◀

We can now proceed to the proof of Lemma 9.

Proof. Coefficients of the hyperplanes of the arrangement are constant rational numbers, those can be changed to constant integers (because each hyperplane has at most k nonzero coefficients). Let C denote the maximum absolute value of those coefficients.

Because of Theorem 17 and Lemma 18, vertices of the arrangement have rational coordinates whose numerators and denominators absolute values are bounded by $O(C^n n^n)$.

Given simplices whose vertices are vertices of the arrangement, hyperplanes that define the faces of those simplices have rational coefficients whose numerator and denominator are bounded by $O(C^{2n^3} n^{2n^3+n})$ by Theorem 17 and Lemma 19. (Note that some simplices might be not fully dimensional, but we can handle those by adding vertices with coordinates that are not much larger than that of already existing vertices).

By applying Theorem 17 and Lemma 19 again, we obtain that vertices of the arrangement of those new hyperplanes (and thus vertices of S) have rational coefficients whose numerator and denominator are bounded by $O(C^{4n^5} n^{4n^5+2n^3+n})$. ◀