Subquadratic Encodings for Point Configurations

Jean Cardinal[1]
Département d'Informatique, Université libre de Bruxelles (ULB)
Brussels, Belgium
jcardin@ulb.ac.be
https://orcid.org/0000-0002-2312-0967

Timothy M. Chan
Department of Computer Science, University of Illinois at Urbana-Champaign
Champaign, IL, USA
tmc@illinois.edu

John Iacono[2]
Département d'Informatique, Université libre de Bruxelles (ULB)
Brussels, Belgium
socg18@johniacono.com
https://orcid.org/0000-0001-8885-8172

Stefan Langerman[3]
Département d'Informatique, Université libre de Bruxelles (ULB)
Brussels, Belgium
slanger@ulb.ac.be
https://orcid.org/0000-0001-6999-3088

Aurélien Ooms[4]
Département d'Informatique, Université libre de Bruxelles (ULB)
Brussels, Belgium
aureooms@ulb.ac.be
https://orcid.org/0000-0002-5733-1383

Abstract

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 this contribution, we give a compact encoding for abstract order types that allows 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

[1] Supported by the "Action de Recherche Concertée" (ARC) COPHYMA, convention number 4.110.H.000023.
[2] Supported by NSF grants CCF-1319648, CCF-1533564, CCF-0430849 and MRI-1229185, a Fulbright Fellowship and by the Fonds de la Recherche Scientifique-FNRS under Grant n° MISU F 6001 1 and two Missions Scientifiques.
[3] Directeur de recherches du Fonds de la Recherche Scientifique-FNRS.
[4] Supported by the Fund for Research Training in Industry and Agriculture (FRIA).

encoding for those order types with fast orientation queries. 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. Finally, we generalize our results to the encoding of point configurations in higher dimension.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases point configuration, order type, chirotope, succinct data structure

Digital Object Identifier 10.4230/LIPIcs.SoCG.2018.20

Related Version A full version of this paper is available at `https://arxiv.org/abs/1801.01767`

1 Introduction

At SoCG'86, Chazelle asked [29]:

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

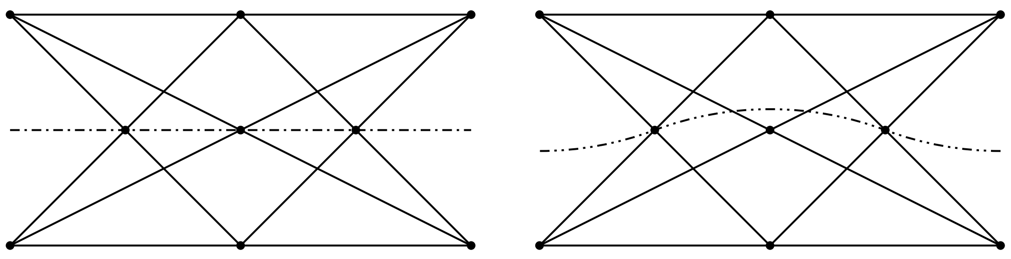
This question is of importance in Computational Geometry for the following two reasons: First, in many algorithms dealing with sets of points in the plane, the only relevant information carried by the input is the combinatorial configuration of the points given by the orientation of each triple of points in the set (clockwise, counterclockwise, or collinear) [18]. 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 [41].

The (counterclockwise) orientation $\nabla(p, q, r) \in \{-, 0, +\}$ of a triple of points p, q, and r with coordinates (x_p, y_p), (x_q, y_q), and (x_r, y_r) is the sign of the determinant

$$\begin{vmatrix} 1 & x_p & y_p \\ 1 & x_q & y_q \\ 1 & x_r & y_r \end{vmatrix}.$$

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. A great deal of the literature in computational geometry deals with this notion [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 19, 20, 21, 22, 25, 26, 27, 28, 29, 30, 32, 33, 34, 36, 37, 38, 39]. The order type of a point set has been further abstracted into combinatorial objects known as (rank-three) *oriented matroids* [21]. The *chirotope axioms* define consistent systems of signs of triples [12]. From the topological representation theorem [13], all such *abstract* order types correspond to pseudoline arrangements, while, from the standard projective duality, order types of point sets correspond to straight line arrangements. See Chapter 6 of The Handbook for more details [38].

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 [33] (see Figure 1). Ringel shows how to convert the so-called "non-Pappus" arrangement of Figure 1 (b)



(a) Realizable order type. **(b)** Abstract order type which is not realizable.

Figure 1 Pappus's configuration.

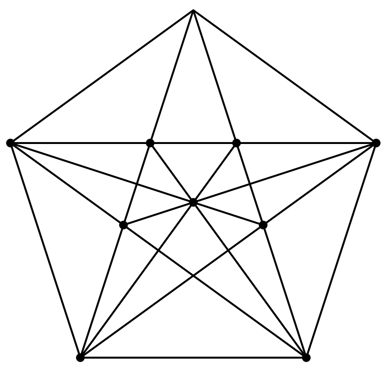


Figure 2 Perles's configuration.

to a simple arrangement while preserving nonstretchability [39]. All arrangements of eight or fewer pseudolines are stretchable [24], and the only nonstretchable simple arrangement of nine pseudolines is the one given by Ringel [37]. More information on pseudoline arrangements is available in Chapter 5 of The Handbook [23].

Figure 1 shows that not all pseudoline arrangements are stretchable. Indeed, most are not: there are $2^{\Theta(n^2)}$ abstract order types [19] and only $2^{\Theta(n \log n)}$ realizable order types [9, 27]. This discrepancy stems from the algebraic nature of realizable order types, as illustrated by the main tool used in the upper bound proofs (the Milnor-Thom Theorem [35, 40]).

Information theory implies that we need quadratic space for abstract order types whereas we only need linearithmic space for realizable order types. Hence, storing all $\binom{n}{3}$ orientations in a lookup table seems wasteful. Another obvious idea for storing 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 [31](see Figure 2). Order types of points in general position can always be represented by rational coordinates. It is well known, however, that some configurations require doubly exponential coordinates, hence coordinates with exponential bitsizes if represented in the normal way [30].

Goodman and Pollack defined λ-matrices which can encode abstract order types using $O(n^2 \log n)$ bits [25]. They asked if the space requirements could be moved closer to the information-theoretic lower bounds. 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 [19, 20] (as defined in [22]). 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 [10]. However, it is not known how to decode the orientation of one triple from any of those 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 results

In this contribution, 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. In Section 2, 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 $w \geq \log n$. Our encoding is far from being space-optimal for realizable order types. We show that its construction can be easily tuned to only require $O(n^2(\log\log n)^2/\log n)$ bits in this case. In Section 3, we further refine our encoding to reduce the query time to $O(\log n/\log\log n)$. In the realizable case, we give quadratic upper bounds on the preprocessing time required to compute an encoding in the real-RAM model. In the full version of the paper, we generalize our encodings for chirotopes of point sets in higher dimension [15].

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 constant-degree algebraic decision tree exists for the related problem of deciding whether a point set contains a collinear triple. Any such decision tree would yield another subquadratic encoding for realizable order types. 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.

2 Encoding order types via hierarchical cuttings

To make our statements clear, we use the following definition:

▶ **Definition 1.** 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 $t \in [n]^k$, we can compute $f(t)$ in $Q(n)$ query 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,[5] $k = 3$ 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:

▶ **Theorem 2.** *All abstract order types have an* $(O(n^2), O(\log n))$*-encoding.*

▶ **Theorem 3.** *All realizable order types have an* $(O(\frac{n^2(\log\log n)^2}{\log n}), O(\log n))$*-encoding.*

[5] 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 [10].

▶ **Theorem 4.** *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 2 and 3 in $O(n^2)$ time.*

For instance, Theorem 3 implies that for any set of points $\{ p_1, p_2, \dots, p_n \}$, 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 lines or pseudolines that realizes the 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 4, we specify the model of computation and how the input is given.

Hierarchical cuttings

We encode the order type of an arrangement via hierarchical cuttings as defined in [16]. A cutting in $\mathbb{R}^d$ is a set of (possibly unbounded and/or non-full dimensional) constant-complexity cells that together partition $\mathbb{R}^d$. A $\frac{1}{r}$-cutting of a set of n hyperplanes is a cutting with the constraint that each of its cells is intersected by at most $\frac{n}{r}$ hyperplanes. There exist various ways of constructing $\frac{1}{r}$-cuttings of size $O(r^d)$. Those cuttings allow for efficient divide-and-conquer solutions to many geometric problems. The hierarchical cuttings of Chazelle have the additional property that they can be composed without multiplying the hidden constant factors in the big-oh notation. In particular, they allow for $O(n^d)$-space $O(\log n)$-query d-dimensional point location data structures (for constant d). In the plane, hierarchical cuttings can be constructed for arrangement of pseudolines with the same properties.

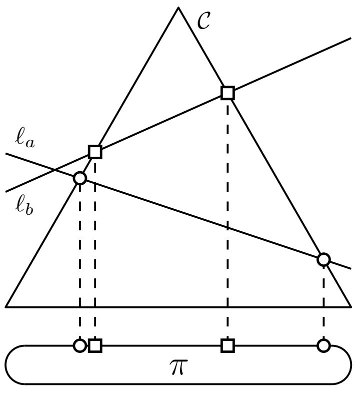
Idea

We want to preprocess n pseudolines $\{ \ell_1, \ell_2, \dots, \ell_n \}$ in the plane so that, given three indices a, b, and c, we can compute their orientation, that is, whether the intersection $\ell_a \cap \ell_b$ lies above, below or on ℓ_c. Our data structure builds on cuttings as follows: Given a cutting Ξ and the three indices, we can locate the intersection of ℓ_a and ℓ_b with respect to Ξ. The location of this intersection is a cell of Ξ. The next step is to decide whether ℓ_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 solve all subproblems efficiently.

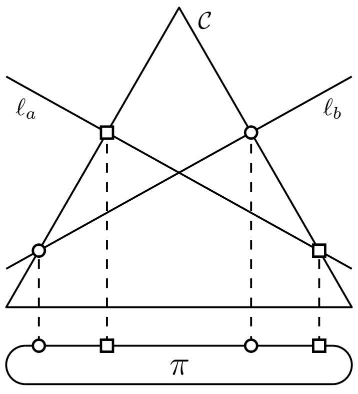
Intersection location

When the ℓ_a are straight lines, locating the intersection $\ell_a \cap \ell_b$ in Ξ is trivial if we know the real parameters of ℓ_a and ℓ_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 a simple observation illustrated by Figure 3.

▶ **Observation 5.** *Two pseudolines ℓ_a and ℓ_b intersect in the interior of a full-dimensional cell $\mathcal{C}$ if and only if each pseudoline properly intersects the boundary of $\mathcal{C}$ exactly twice and their intersections with its boundary alternate.*



(a) $\ell_a \cap \ell_b \cap \mathcal{C} = \emptyset$.



(b) $\ell_a \cap \ell_b \cap \mathcal{C} \neq \emptyset$.

Figure 3 Cyclic permutations (π).

This gives us a way to encode the location of the intersection of ℓ_a and ℓ_b in Ξ using only bits. For an arrangement of pseudolines, we use the standard vertical decomposition to construct a hierarchical cutting, which guarantees that a pseudoline intersects a cell boundary at most twice. For an arrangement of lines, we can use the standard bottom-vertex triangulation instead, which allows us to generalize our results to higher dimensions [15]. In the plane, the bottom-vertex triangulation partitions the space into triangular cells. We define the *cyclic permutation* of a full-dimensional cell $\mathcal{C}$ and a finite set of pseudolines $\mathcal{L}$ to be the finite sequence of properly intersecting pseudolines from $\mathcal{L}$ encountered when walking along the boundary of $\mathcal{C}$ in clockwise or counterclockwise order, up to rotation and reversal.

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 intersections with 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

Given n pseudolines in the plane and some fixed parameter r, compute a hierarchical $\frac{1}{r}$-cutting of those pseudolines. 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`) or intersects the interior of that subcell (`10`). When a subcell is non-full-dimensional, we use another value (`11`) when the pseudoline contains the subcell. When a pseudoline intersects the interior of a 2-dimensional subcell, we also store the two indices of the intersections of that pseudoline with the subcell in the cyclic permutation associated with 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 intersection in the acyclic permutation associated with 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 associated permutation.

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. For each of the $\lambda = O(\frac{n^2}{t^2})$ subcells of the last level of the hierarchy we store a pointer to a lookup table of size $\tau = O(t^3)$ that allows to answer the query of the orientation of any triple of pseudolines intersecting that subcell. The number $t = \frac{n}{r^\ell}$ denotes an upper bound on the number of pseudolines intersecting each of those subcells.

Storing the permutation at each subcell would suffice to answer all queries that do not reach the last level of the hierarchy. However, to get fast queries, we need to have access to all bits belonging to a given pseudoline without having to read the bits of the others. Using the Zone Theorem ([11, 17, 23]), and the fact that hierarchical cuttings are constructed by decompositions of subsets of the input pseudolines, we can bound the number of bits stored for a single pseudoline intersecting a given cell of level i by $\zeta_i = O(r^2 + r \log \frac{n}{r^{i+1}})$. This allows us to store all bits belonging to a given cell-pseudoline pair $(\mathcal{C}, \ell)$ in a contiguous block of memory $\sigma(\mathcal{C}, \ell)$ whose location in the encoding is easy to compute. We call $\sigma(\mathcal{C}, \ell)$ the *signature* of ℓ in $\mathcal{C}$. The overall number of bits stored stays the same up to a constant factor.

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 reuse space. Formally, let $\nu(n)$ denote the number of order types of size n, which is $\nu(n) = 2^{\Theta(n^2)}$ for abstract order types and $\nu(n) = 2^{\Theta(n \log n)}$ for realizable order types. At most $\nu(t)$ distinct lookup tables are needed for answering the queries on the subcells of the last level of the hierarchy. Hence the pointers have size $\Psi = O(\log \nu(t))$ and the total size needed for the lookup tables is $O(t^3 \nu(t))$. For each leaf, we store a canonical labeling of size $\kappa = O(t \log t)$ on the pseudolines that intersect it. We use that canonical labeling to order the queries in the associated lookup table.

The encoding is the concatenation of the parameters n, r, and t, all signatures $\sigma(\mathcal{C}, \ell)$ for all pairs $(\mathcal{C}, \ell)$ with $\ell \cap C$, the canonical labelings at the leaves, the leaf pointers and the lookup tables. We define a canonical order on the cells of level i so that they can be ordered as $\{\mathcal{C}_{i,1}, \mathcal{C}_{i,2}, \ldots, \mathcal{C}_{i,O(r^{2i})}\}$. We complement the encoding with appropriate padding so that the position $\rho(\mathcal{C}_{i,j}, \ell)$ of the signature $\sigma(\mathcal{C}_{i,j}, \ell)$ is

$$\rho(\mathcal{C}_{i,j}, \ell) = \rho(\mathcal{C}_{i-1,1}, \ell_0^{\mathcal{C}_{i-1,1}}) + cr^{2i-2} \left\lfloor \frac{n}{r^{i-1}} \right\rfloor \zeta_{i-1} + (j-1) \left\lfloor \frac{n}{r^i} \right\rfloor \zeta_i + \pi(\mathcal{C}_i, \ell)\zeta_i,$$

where c is some constant, $\pi(\mathcal{C}, \ell)$ is the first index of ℓ in the cyclic permutation of $\mathcal{C}$, $\ell_a^{\mathcal{C}}$ is the pseudoline such that $\pi(\mathcal{C}, \ell_a^{\mathcal{C}}) = a$, and, for the root cell of the hierarchy $\mathcal{C}_{0,1}$ representing the entire space and containing all the intersections of the arrangement, $\rho(\mathcal{C}_{0,1}, \ell_0)$ is the position after the encoding of the parameters n, r, and t, and $\rho(\mathcal{C}_{0,1}, \ell_a) = \rho(\mathcal{C}_{0,1}, \ell_0) + a\zeta_0$.

The canonical labeling of the first leaf is stored at position

$$\rho_{\Lambda_0} = c\sum_{i=0}^{\ell-1} r^{2i} \left\lfloor \frac{n}{r^i} \right\rfloor \zeta_i,$$

the lookup table pointer of the first leaf is stored at position $\rho_{\Lambda_0} + \kappa$, the canonical labeling of leaf Λ is stored at position $\rho_{\Lambda_0} + (\Lambda - 1)(\kappa + \Psi)$ and its table lookup pointer at κ from that position. The first lookup table is stored at position $\rho_{\Lambda_0} + \lambda(\kappa + \Psi)$ and lookup table Θ is stored at position $\rho_{\Lambda_0} + \lambda(\kappa + \Psi) + (\Theta - 1)\tau$.

Space complexity

We prove a general bound on the space taken by our construction when the hierarchy contains ℓ levels. 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 $r \in (1, +\infty)$. This excludes 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.

▶ **Lemma 6.** *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, we have

$$H_r^\ell(n) = O\left(\sum_{i=0}^{\ell-1}\left(r^{2i} \cdot r^2 \cdot \left(\frac{n}{r^i} + \frac{n}{r^{i+1}} \log \frac{n}{r^{i+1}}\right)\right)\right).$$

We multiply the previous equation by $\frac{n}{tr^\ell} = 1$

$$H_r^\ell(n) = O\left(\frac{n^2}{t}\sum_{i=0}^{\ell-1}\left(\frac{1}{r^{\ell-i-1}} \cdot \left(r + \log \frac{n}{r^{i+1}}\right)\right)\right).$$

We use the equivalence $\frac{n}{r^{i+1}} = tr^{\ell-i-1}$ to replace the last term in the previous equation

$$H_r^\ell(n) = O\left(\frac{n^2}{t}\sum_{i=0}^{\ell-1}\left(\frac{1}{r^{\ell-i-1}} \cdot (r + \log t + (\ell - i - 1)\log r)\right)\right).$$

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:

$$\sum_{i=0}^{k} x^i \leq \frac{1}{1-x} \qquad \text{and} \qquad \sum_{i=0}^{k} ix^i \leq \frac{x}{(1-x)^2}, \qquad \forall k \in \mathbb{N}, \forall x \in (0,1),$$

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 that for $r \geq 2$

$$H_r^\ell(n) = O\left(\frac{n^2}{t}(\log t + r)\right).$$ ◀

Taking into account the space taken by the other bits of the encoding we obtain

▶ **Lemma 7.** *The space taken by our encoding is*

$$S_r^\ell(n) = O\left(\log ntr + \frac{n^2}{t}(\log t + r) + t^3\nu(t) + \frac{n^2}{t^2}(\log \nu(t) + t\log t)\right).$$

We pick r constant for both abstract and realizable order type. 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 last term in Lemma 7 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 last term in Lemma 7 dominate with $n^2(\log\log n)^2/\log n$. This proves the space constraints in Theorems 2 and 3.

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.[6] Let $\mathcal{C} = \mathcal{C}_{0,1}$. First, find the subcell $\mathcal{C}'$ of $\mathcal{C}$ containing $\ell_a \cap \ell_b$ by testing for each subcell whether the intersections of ℓ_a and ℓ_b with the subcell alternate in the cyclic permutation. This can be done in $O(r^2)$ time by scanning $\sigma(\mathcal{C}, \ell_a)$ and $\sigma(\mathcal{C}, \ell_b)$ in parallel. Note that non-full dimensional subcells can be tested more easily. Next, if ℓ_c does not properly intersect $\mathcal{C}'$, answer the query accordingly. If on the other hand ℓ_c does properly intersect the subcell we recurse on $\mathcal{C}'$. This can be tested by scanning $\sigma(\mathcal{C}, \ell_c)$ in $O(r^2)$ time. Note that in case that 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 position of the signatures scanned during the first recursive step of the query can be computed in constant time and at each other recursive step of the query we can compute the positions of the signatures we need to scan from the position of the signatures scanned during the previous recursive step in constant time. When we reach the bottom of the recursion, the position of the lookup table pointer, the position of the canonical labeling, and the position of the lookup table can be computed in 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 2 and 3. 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 $\ell_a \cap \ell_b$. We show how to handle small but superconstant r properly in the next section.

[6] 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 except the smallest. 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.

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 4.

Proof. A hierarchical cutting can be computed in $O(nr^\ell)$ time in the dual plane. All signatures $\sigma(\mathcal{C}, \ell)$ 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 [10]. 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. ◀

3 Sublogarithmic query complexity

We further refine the data structure defined 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. The idea is to make each signature $\sigma(\mathcal{C}, \ell)$ fit in a single word of memory by only approximately encoding the cyclic permutation of the intersections around each subcell, relying on the fact that ambiguous situations rarely arise. Those ambiguous situations, if they happen, can be deterministically handled using additional lookup tables. This improvement is applicable for both abstract and realizable order types.

We improve our main theorems for the two-dimensional case:

▶ **Theorem 8.** *All abstract order types have an $(O(n^2), O(\frac{\log n}{\log\log n}))$-encoding.*

▶ **Theorem 9.** *All realizable order types have a $(O(\frac{n^2 \log^\varepsilon n}{\log n}), O(\frac{\log n}{\log\log n}))$-encoding.*

▶ **Theorem 10.** *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 8 and 9 in $O(n^2)$ time.*

Bit packing

Fix a large α, a small δ and define $r = \Theta(\log^\delta n)$. 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. Denote by $n_i = n/r^i$ an upper bound on the number of lines intersecting a cell of level i. For each subcell of level i, partition its cyclic permutation into $\log^\alpha n$ blocks of at most $n_{i+1}/\log^\alpha n$ intersections. For each pseudoline intersecting a cell we only store the block numbers that that pseudoline touches in its signature. Hence, each signature $\sigma(\mathcal{C}, \ell)$ only uses $\Theta(\log^{2\delta} n + \alpha \log^\delta n \log\log n) = \Theta(\log^{2\delta} n)$ bits, which fits in a word for small enough δ.

Intersection oracle

We construct an additional lookup table to compute the subcell in which $q_i \cap q_j$ lies in constant time. Computing it via scanning with so many subcells to check would waste any further savings. For that we need a general observation on the precomputation of functions on small universes.

▶ **Observation 11.** *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 $2^{s+1}w$ for all 2^s inputs $x \in [2^s]$ of bitsize $s \leq w$ in time $2^s T(s)$ where $T(s)$ is the complexity of computing $f(x)$, $x \in [2^s]$. The image of any input of bitsize s can then be retrieved in $O(1)$ time by a single lookup (since the input fits in a single word). In particular, we have $2^s T(s)$ and $2^{s+1}w$ sublinear as long as $T(s) = s^{O(1)}$ and $s \leq (1-\varepsilon)\log_2 n$.*

In other words, any polynomial time computable word-to-word function can be precomputed in sublinear time and space for all inputs of roughly logarithmic size.

Since our pseudoline identifiers now fit in $\Theta(\log^{2\delta} n)$ bits we can choose an appropriate δ so as to satisfy the requirements given above. We can thus precompute the function that sends two pseudoline identifiers to either the subcell containing their intersection or to some special value in case of an ambiguous input.

Disambiguation

Note that ambiguous inputs rarely occur. An input is ambiguous if and only if at least one boundary intersection of each pseudoline appears in the same cyclic permutation block of the cell that contains their intersection. This happens less than $\log^\alpha n \cdot (n_{i+1}/\log^\alpha n)^2 = \frac{n^2}{r^{2(i+1)}}/\log^\alpha n$ times per subcell of level i of the hierarchy. Summing over all levels, we get a subquadratic size lookup table for ambiguous cases which can be implemented using standard tools.

Space complexity

The total space used for the signatures $\sigma(\mathcal{C}, \ell)$ is proportional to

$$\sum_{i=0}^{\ell-1} r^{2i} \cdot r^2 \cdot \left(\frac{n}{r^i} + \frac{n}{r^{i+1}}\alpha \log\log n\right) = O\left(\frac{n^2}{t}(\log^\delta n + \alpha \log\log n)\right).$$

Intersection oracles and disambiguation tables fit in subquadratic space and the space analysis for the rest of the data structure still holds. For small enough δ the space remains quadratic for abstract order types and subquadratic for realizable order types. This proves the space constraints in Theorems 8 and 9. Unfortunately, we must incur a nonabsorbable extra $\log^\delta n$ factor in the realizable case. Note that a $\log\log\log n$ factor can be squeezed without increasing the space usage by choosing $r = \Theta(\log\log n)$ instead.

Correctness and query complexity

The previous analysis still holds modulo additional disambiguation lookups and oracle-based intersection location. We now have a shallower decision tree of depth $\log_r n = O_\delta(\frac{\log n}{\log\log n})$. This proves the query time constraints in Theorems 8 and 9.

Preprocessing time

We prove Theorem 10.

Proof. As before, the hierarchical cutting and all signatures $\sigma(C, \ell)$ can be computed in $O(nr^\ell)$ time. The lookup table and leaf-table pointers can be computed in $O(n^2)$ time. All intersection oracles and disambiguation tables can be computed in subquadratic time. ◀

4 Conclusion

Observe the following. Assume we are given an instance of some real-input decision problem. Given a decision tree of depth D for this problem for which each input query and answer can be encoded using at most Q bits, we can encode the instance using at most DQ bits by encoding the path traversed when executing the decision tree on this instance. The general position testing problem (GPT) asks if an input set of n points in the plane contains a collinear triple. It is an example of a real-input decision problem for which no subquadratic real-RAM algorithm is known even though the best known lower bound is only linearithmic. Since shallow decision trees yield short encodings, we see the design of a subquadratic encoding for realizable order types as a stepping stone towards nonuniform and uniform subquadratic algorithms for GPT.

Unfortunately, even though our encodings achieve subquadratic space for realizable order types, they cannot be used to test for isomorphism in subquadratic time. This is partly because the preprocessing time to construct the encoding is already quadratic. However, observe that the preprocessing time we achieve in this contribution matches the best known upper bound for GPT in the algebraic decision tree model:

▶ **Theorem 12.** *If there is an encoding with construction cost $C(N)$ for realizable order types in the algebraic decision tree model, then there is a nonuniform algorithm for general position testing that runs in time $C(N)$ in the algebraic decision tree model.*

Proof. Construct the encoding. Then at zero cost in the nonuniform model, run all $O(n^3)$ queries on the encoding. ◀

Goodman and Pollack saw GPT as a multidimensional generalization of sorting [25]. We again stress the need for a better understanding of this fundamental problem.

References

1 Oswin Aichholzer, Franz Aurenhammer, and Hannes Krasser. Enumerating order types for small point sets with applications. *Order*, 19(3):265–281, 2002.

2 Oswin Aichholzer, Franz Aurenhammer, and Hannes Krasser. On the crossing number of complete graphs. In *SOCG*, pages 19–24. ACM, 2002.

3 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.

4 Oswin Aichholzer, Matias Korman, Alexander Pilz, and Birgit Vogtenhuber. Geodesic order types. *Algorithmica*, 70(1):112–128, 2014.

5 Oswin Aichholzer and Hannes Krasser. The point set order type data base: A collection of applications and results. In *CCCG*, pages 17–20, 2001.

6 Oswin Aichholzer and Hannes Krasser. Abstract order type extension and new results on the rectilinear crossing number. In *SOCG*, pages 91–98. ACM, 2005.

7 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.

8 Oswin Aichholzer, Tillmann Miltzow, and Alexander Pilz. Extreme point and halving edge search in abstract order types. *Computational Geometry*, 46(8):970–978, 2013.

9 Noga Alon. The number of polytopes configurations and real matroids. *Mathematika*, 33(1):62–71, 1986.

10 Greg Aloupis, John Iacono, Stefan Langerman, Özgür Özkan, and Stefanie Wuhrer. The complexity of order type isomorphism. In *SODA*, pages 405–415. SIAM, 2014.

11 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.

12 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.

13 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.

14 Jürgen Bokowski, Jürgen Richter-Gebert, and Werner Schindler. On the distribution of order types. *Computational Geometry*, 1(3):127–142, 1992.

15 Jean Cardinal, Timothy M. Chan, John Iacono, Stefan Langerman, and Aurélien Ooms. Subquadratic encodings for point configurations. *ArXiv e-prints*, 2018. arXiv:1801.01767 [cs.CG].

16 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158, 1993.

17 Bernard Chazelle, Leonidas J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25(1):76–90, 1985.

18 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10. Springer Science & Business Media, 2012.

19 Stefan Felsner. On the number of arrangements of pseudolines. In *SOCG*, pages 30–37. ACM, 1996.

20 Stefan Felsner and Pavel Valtr. Coding and counting arrangements of pseudolines. *Discrete & Computational Geometry*, 46(3):405–416, 2011.

21 Jon Folkman and Jim Lawrence. Oriented matroids. *Journal of Combinatorial Theory, Series B*, 25(2):199–236, 1978.

22 Jacob E. Goodman. Proof of a conjecture of Burr, Grünbaum, and Sloane. *Discrete Mathematics*, 32(1):27–35, 1980.

23 Jacob E. Goodman. Pseudoline arrangements. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 97–128. Chapman and Hall/CRC, 2004.

24 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.

25 Jacob E. Goodman and Richard Pollack. Multidimensional sorting. *SIAM Journal on Computing*, 12(3):484–507, 1983.

26 Jacob E. Goodman and Richard Pollack. Semispaces of configurations, cell complexes of arrangements. *Journal of Combinatorial Theory, Series A*, 37(3):257–293, 1984.

27 Jacob E. Goodman and Richard Pollack. Upper bounds for configurations and polytopes in $\mathbb{R}^d$. *Discrete & Computational Geometry*, 1:219–227, 1986.

28 Jacob E. Goodman and Richard Pollack. The complexity of point configurations. *Discrete Applied Mathematics*, 31(2):167–180, 1991.

29 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.

30 Jacob E. Goodman, Richard Pollack, and Bernd Sturmfels. Coordinate representation of order types requires exponential storage. In *STOC*, pages 405–410. ACM, 1989.

31 Branko Grünbaum. *Convex Polytopes.* Springer, 2005.

32 Alfredo Hubard, Luis Montejano, Emiliano Mora, and Andrew Suk. Order types of convex bodies. *Order*, 28(1):121–130, 2011.

33 Friedrich Levi. Die teilung der projektiven ebene durch gerade oder pseudogerade. *Ber. Math.-Phys. Kl. Sächs. Akad. Wiss*, 78:256–267, 1926.

34 Yoshitake Matsumoto, Sonoko Moriyama, Hiroshi Imai, and David Bremner. Matroid enumeration for incidence geometry. *Discrete & Computational Geometry*, 47(1):17–43, 2012.

35 John Milnor. On the Betti numbers of real varieties. *Proceedings of the American Mathematical Society*, 15(2):275–280, 1964.

36 Jaroslav Nešetřil and Pavel Valtr. A Ramsey property of order types. *Journal of Combinatorial Theory, Series A*, 81(1):88–107, 1998.

37 Jürgen Richter. Kombinatorische realisierbarkeitskriterien für orientierte matroide. *Mitt. Math. Sem. Univ. Giessen*, 194:1–112, 1989.

38 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.

39 Gerhard Ringel. Teilungen der ebene durch geraden oder topologische geraden. *Mathematische Zeitschrift*, 64(1):79–102, 1956.

40 René Thom. Sur l'homologie des variétés algébriques. In *Differential and Combinatorial Topology (A Symposium in Honor of Marston Morse)*, pages 255–265, 1965.

41 Chee K. Yap. Robust geometric computation. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 927–952. Chapman and Hall/CRC, 2004.