NumPSLA — An experimental research tool for pseudoline arrangements and order types

Günter Rote
Freie Universität Berlin, Institut für Informatik, Takustraße 9, 14195 Berlin, Germany

Abstract

We present a program for enumerating all pseudoline arrangements with a small number of pseudolines and abstract order types of small point sets. This program supports computer experiments with these structures, and it complements the order-type database of Aichholzer, Aurenhammer, and Krasser. This system makes it practical to explore the abstract order types for 12 points, and the pseudoline arrangements of 11 pseudolines. We also report some statistics about abstract order types of 13 points that were obtained on a compute-cluster.

Keywords and phrases Mathematical software, enumeration and counting, literate programming

Digital Object Identifier 10.57717/cgt.v5i3.88

Related Version A preliminary version of this work has been presented at the 41st European Workshop on Computational Geometry (EuroCG'25) in Liblice, Czech Republic, April 9–11, 2025 [29].

Supplementary Material *Software and results*: `https://github.com/guenterrote/NumPSLA` [31]

Acknowledgements I thank the High-Performance-Computing Service of FUB-IT, Freie Universität Berlin [8] for computing time. I thank Akiva Weinberger (Brandeis University) for pointing out erroneous entries in the last three lines of Table 6 in a previous version, which uncovered a mistake in the exclude-file for 11 points (Section 6).

1 Introduction

Questions about finite configurations of points or lines are at the core of discrete geometry. As one example of an outstanding open question, we mention the rectilinear crossing number problem for the complete graph K_n: For a given set S of n points in the plane, draw all straight segments between pairs of points in S, and count the pairs of segments that cross. What is the smallest number that can be obtained?

The order type of a point set. The above question and many other questions and algorithms in discrete and computational geometry depend only on the "combinatorial structure", which is typically captured by an orientation predicate: Consider a finite point set $S = \{p_1, \ldots, p_n\}$. For each triplet $p_i, p_j, p_k \in S$, we need to know whether they lie in clockwise or counterclockwise order, or whether they are collinear. This information is enough to determine, say, the number of convex hull vertices, or the number of crossings.

The order-type database. It is useful if one can let the computer exhaustively check small examples. This may provide a sanity check for conjectures, or it may form the basis for quantitative results that hold in general. We will mention one example below. The prime tool that facilitates this approach is the order-type database of Aichholzer, Aurenhammer, and Krasser [1, 3] at Graz University of Technology from the early 2000's. Originally, it contained a point set (given by coordinates) for each of the 14,309,547 order types of 10 points (and also for the smaller sets). These point sets are optimized to avoid degeneracies

© Günter Rote
licensed under Creative Commons License CC-BY 4.0

as much as possible. Later, the database was extended [5] to include the 2.3 billion order types of 11 points (see the second column of Table 1).

Over the years, the database has been enriched with all sorts of useful information about each order type, ranging from such basic data as the size of the convex hull to advanced characteristics that are hard to compute, such as the number of triangulations or the number of crossing-free Hamilton cycles. The database of order types with up to 10 points can be obtained from the website of the project [2], and it can be queried via an e-mail interface. The database for 11 points needs 102.7 GBytes (44 bytes per order type for two 16-bit coordinates per point). Obviously, the approach of storing a representative of every order type has currently reached its limits with 11 points. We take an alternative approach: *generating* order types from scratch.

Big results from small sets. We mention just one example of a result that rests on the order-type database. Aichholzer et al. [4, Theorem 1] proved that every set S of n points in general position contains $\Omega(n \log^{4/5} n)$ convex 5-holes, i.e., 5-tuples of points in convex position with no points of S in the interior. Harborth [20] showed in 1978 that every set of 10 points contains a convex 5-hole. From this, one gets an immediate lower bound of $\lfloor n/10 \rfloor$ 5-holes by partitioning S into groups of size 10 by vertical lines. Various improvements of the constant factor of this linear bound were obtained over the years. The superlinear bound $\Omega(n \log^{4/5} n)$ goes beyond what can be reached by this simple technique. Nevertheless, at the basis of its proof, there are some structural lemmas about sets of 11 points. These lemmas were checked with the help of a computer by exhaustive enumeration of order types.

1.1 Line arrangements and pseudoline arrangements

The well-known duality

$$\text{point } (a, b) \longleftrightarrow \text{ line } y = ax - b \tag{1}$$

is a bijection between points and non-vertical lines. It swaps the role of points and lines, and it preserves incidences and above-below relationships. Thus, problems about points can be translated into problems about lines and vice versa.

Pseudoline arrangements and abstract order types of points. Pseudoline arrangements are a generalization of line arrangements. A pseudoline arrangement (PSLA) is a collection of unbounded curves, with the condition that any two curves intersect exactly once, and they cross at this intersection point. We refer to these curves as *pseudolines* or simply as *lines*. See Figure 1 for an example with 5 pseudolines. The middle and the right picture show a standard representation as a *wiring diagram*, in two different styles. Both drawings are produced by our program. In a wiring diagram, the pseudolines run on n horizontal tracks, and they cross by swapping between adjacent tracks.

By duality, there is an analogous notion for point configurations, an *abstract order type* (AOT). We will elucidate this relation in Section 3. There is a variety of equivalent notions for these objects, such as rhombus tilings, oriented matroids of rank 3, or signotopes; see for example [13, Chapter 6] or the surveys [14, 28]. We briefly discuss the connections with oriented matroids in Section 3.1.

Our program focuses on pseudoline arrangements as the primary objects. The main reason is that they are easy to generate in an incremental way. Another advantage is that they are easy to draw and to visualize.

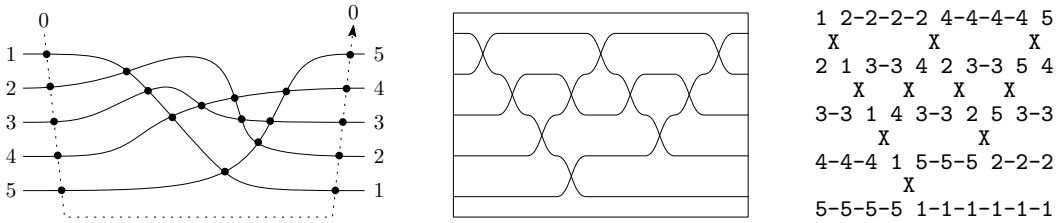


Figure 1 Left: A pseudoline arrangement of 5 lines, extended by a line 0 "at infinity". Middle and right: wiring diagrams

Throughout this paper, we will assume general position. In other words, we restrict our attention to *simple* pseudoline arrangements, where no three lines go through a common point. In the setting of point sets, this corresponds to forbidding collinear point triples.

n	[A006247] #AOT	[A063666] #OT	$\Delta =$ #nonr. AOT	$\frac{\Delta}{\#\text{AOT}}$	[A006245] #PSLA
3	1	1	0	0	2
4	2	2	0	0	8
5	3	3	0	0	62
6	16	16	0	0	908
7	135	135	0	0	24,698
8	3,315	3,315	0	0	1,232,944
9	158,830	158,817	13	0.01 %	112,018,190
10	14,320,182	14,309,547	10,635	0.07 %	18,410,581,880
11	2,343,203,071	2,334,512,907	8,690,164	0.37 %	5,449,192,389,984
12	691,470,685,682				2,894,710,651,370,536
13	366,477,801,792,538				2,752,596,959,306,389,652

Table 1 #AOT = number of abstract order types for n points. #OT = number of order types. #PSLA = number of (x-monotone) pseudoline arrangements with n pseudolines. These are the objects that the program actually enumerates one by one (almost, because we try to apply shortcuts). The column headings link to the corresponding entries of the On-Line Encyclopedia of Integer Sequences [33].

1.2 Overview

We will describe our algorithm for enumerating pseudoline arrangements, and we will apply it to enumerate abstract order types. None of the techniques that we use are novel, but we have tried to streamline and simplify the algorithms. In terms of speed, we can compete with the order type database (see Appendix A for a direct comparison). The main distinction is, of course, that the order type database contains only *realizable* order types, and that they come with coordinates. For many applications, the restriction to realizable order types is not important, and coordinates are not needed. In those applications, our approach shows its strength. Mustering the 14 million 10-point abstract order types takes 10–20 seconds. The 11-point sets can be handled in half an hour, and the 12-point sets take about 200 CPU hours. To this, one must of course add the time for whatever one wants to do with those order types. The program is trivially parallelizable, and with a powerful compute-cluster, it is feasible to go even for 13 points, see Section 8.

The program is available in a GitHub repository [31]. It is written in the programming language C, using the CWEB system of structured documentation of Donald E. Knuth and Silvio Levy[1].

We have occasionally used the enumeration for our research questions [9], and we hope it finds other users.

2 Enumeration of pseudoline arrangements

We concentrate on x-monotone pseudoline arrangements, in which the curves are x-monotone. Every pseudoline arrangement can be drawn in an x-monotone way, but this incurs a choice: One of the unbounded faces must be selected as the *top face* T, and the opposite unbounded face will become the *bottom face* B. Then the lines run from left to right, and we number them from 1 to n as they appear from top to bottom on the left side (see Figure 1). If they were straight lines, they would be numbered by increasing slope.

2.1 Representing a pseudoline arrangement

The vertices and edges of a pseudoline arrangement form a planar graph. The storage and manipulation of this graph is greatly simplified by the fact that we have a precise control over the vertices: There is a vertex for each pair of lines, and every vertex has degree 4. We thus store the edges in two 2-dimensional arrays *succ* and *pred* of successor and predecessor pointers, see Figure 2 for an illustration. The entries $succ[j,k]$ and $pred[j,k]$ refer to the crossing between line k and the line j. We think of the lines as oriented from left to right. Then $succ[j,k]$ and $pred[j,k]$ point to the next and previous crossing on line j. For the reversed index pair $[k,j]$, we get the corresponding information for line k. Thus, in Figure 2, $succ[2,1] = 4$, and accordingly, $pred[2,4] = 1$.

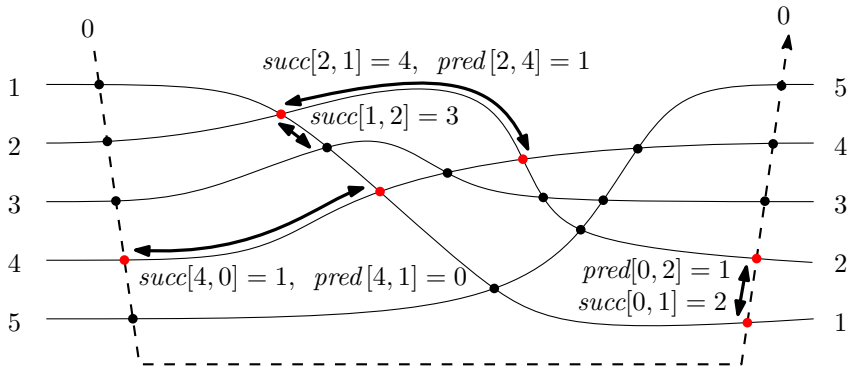


Figure 2 Some successor and predecessor pointers for the PSLA of Figure 1

We can easily determine which of j and k enters the intersection (k,j) from the top and bottom: By our numbering convention, the line with the smaller index always enters above the other line, and to the right of the crossing, it lies below the other line.

The infinite rays on line j are represented by the additional line 0: $succ[j,0]$ is the first (leftmost) crossing on line j, and $pred[j,0]$ is the last crossing. The intersections on line 0 are cyclically ordered $1,\dots,n$. Thus, $succ[0,i] = i+1$ and $succ[0,n] = 1$, and consequently, $pred[0,i] = i-1$ and $pred[0,1] = n$.

[1] `http://tug.ctan.org/info/knuth/cwebman.pdf`

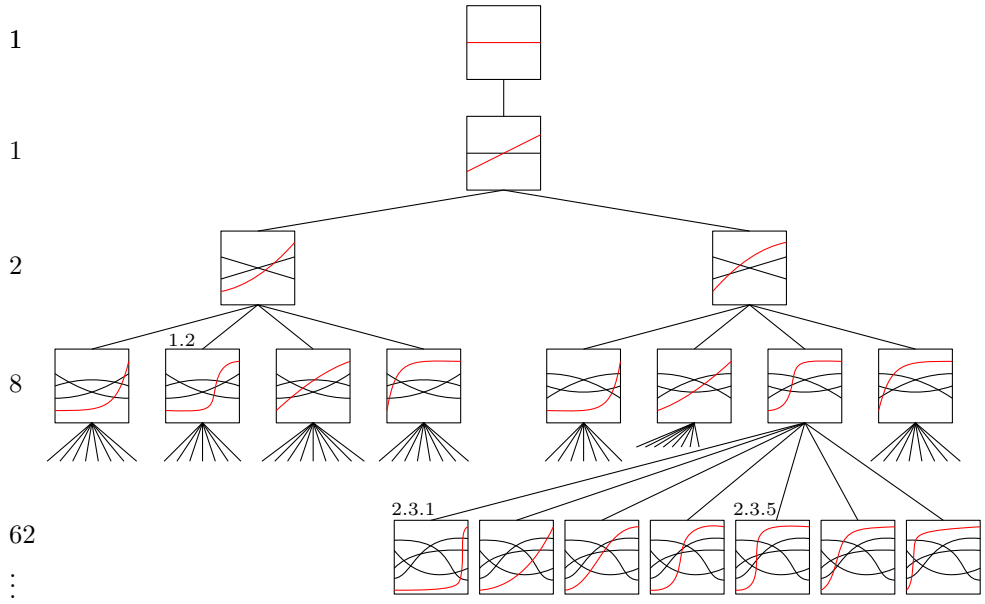


Figure 3 The first three levels of the enumeration tree and a few nodes of the fourth level. The last inserted pseudoline is highlighted in color. For selected nodes, the Dewey decimal notation is given.

2.2 Incremental generation of pseudoline arrangements

We generate a PSLA with n lines by inserting line n into a PSLA with $n-1$ lines, in all possible ways. Then each PSLA has a unique predecessor PSLA, and this predecessor relation imposes a tree structure on the PSLAs, as shown in Figure 3. Our program explores this tree in depth-first order. If we number the children of each node in the order in which they are visited, this leads to a unique identifier for every node, and thus for every PSLA, analogous to the Dewey decimal classification that is used to classify books in libraries.

Inserting the n-th pseudoline into a PSLA of $n-1$ lines corresponds to threading a curve from the bottom face B to the top face T, see Figure 4. (We temporarily relax the requirement that the extra pseudoline has to be x-monotone.) Following Knuth [23, Section 9, p. 38], such a curve is called a *cutpath* [15, 12]. This corresponds to a source-to-target path in the dual graph of the PSLA. Orienting the dual edges in the direction in which line n can cross them, namely, from below to above, leads to a directed acyclic graph (a DAG). We can enumerate all such paths in a backtracking manner. Since the DAG has no sinks other than the target vertex T, a path cannot get stuck, and thus the enumeration of the paths is simple and fast.

The whole algorithm is thus a double recursion. The outer recursion extends a PSLA by adding a pseudoline n. The inner recursion extends a partially drawn pseudoline n to the next crossing, see Figure 5. It is implemented by walking along the boundary of the face that has been entered through the last crossing. All upper edges of the face are candidate edges for the next crossing of line n, and we try them in succession. We have decided to walk in counterclockwise order around the face. This means that the paths for line n are generated in "lexicographic" order from right to left, as can be checked in Figure 3.

Appendix B shows a self-contained PYTHON program that implements this enumeration algorithm.

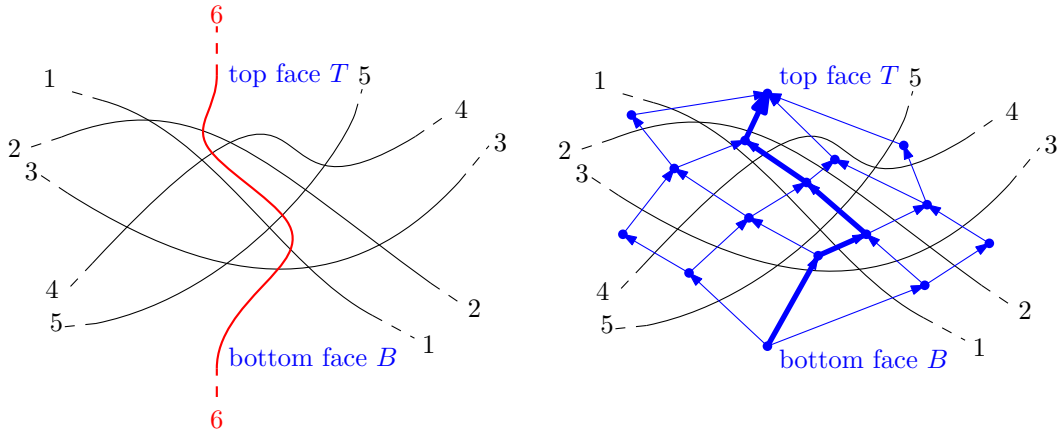


Figure 4 Left: Threading line 6 through a PSLA of 5 lines. Right: The dual DAG of this PSLA

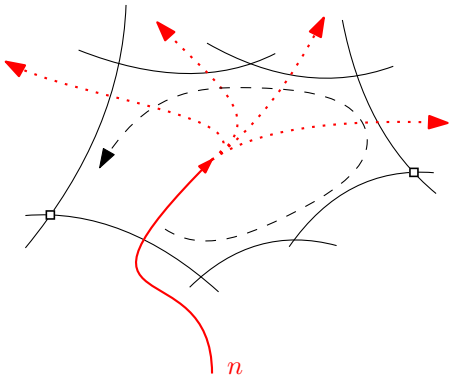


Figure 5 Continuing line n after entering a face.

3 Duality between pseudoline arrangements and abstract order types

The duality between pseudoline arrangements and abstract order types is not as straightforward as one would hope for. Figure 6 shows the intricate network of relationships. The top right box refers to *oriented* abstract order types (OAOTs), where a point set is still distinguished from its reflection. (AOTs don't make this distinction.) At the lower left corner, we find our favorite objects, the (x-monotone) PSLAs.

There is an offset of 1 in the number of objects when transferring between the "point set world" (shaded) and the "pseudoline world": The (x-monotone) PSLAs with n pseudolines correspond to OAOTs or AOTs with $n + 1$ points. However, the correspondence is not one-to-one. Different PSLAs may give rise to the same OAOT and AOT, and the algorithm has to take care of this ambiguity when trying to enumerate OAOTs or AOTs without duplication. The details are given in Section 4.

Let us discuss some of the boxes in more detail, starting at the lower left corner of Figure 6: The pseudolines of an x-monotone PSLA are numbered from 1 to n in order of increasing slope. If we start with the analogy of a line arrangement and apply the duality (1), we get a set of points that are sorted by x-coordinate, as shown in Figure 7. Now, the notion of being sorted by x-coordinate is foreign to order types, but we can incorporate it by imagining a point 0 at vertical positive infinity, around which the points are radially sorted.

We can also move this extra point to a finite distance, sufficiently far above, without

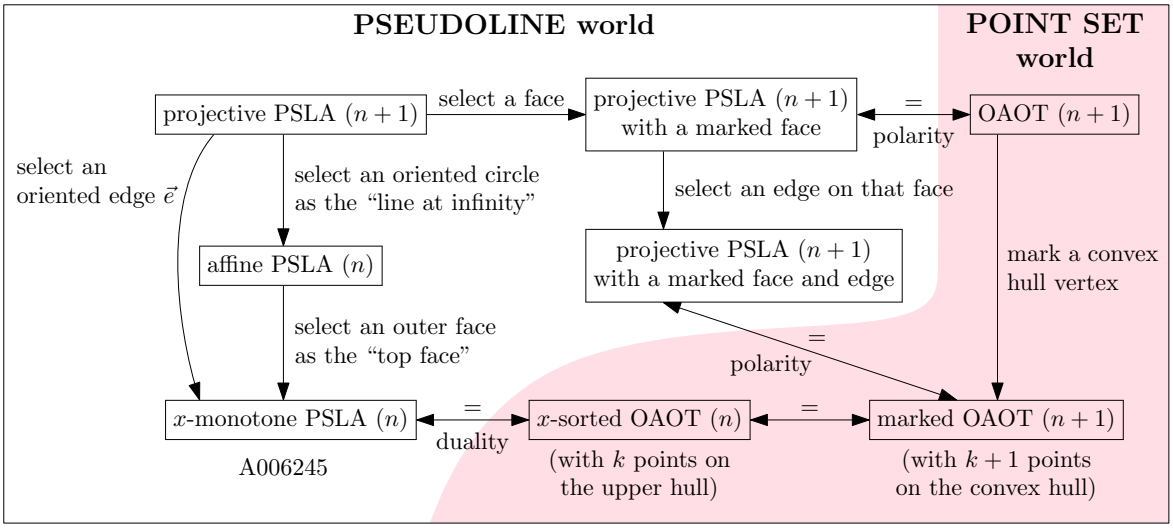


Figure 6 Relation between different concepts. An arrow in one direction indicates a specialization.

changing the order type. Moreover, if we move the point 0 to the left of all points, as in Figure 7, we see that we can let this point correspond to the line 0 in the PSLA, or more precisely, to the part of line 0 that lies at the left of all crossings. This line has a smaller slope than all other lines, and it intersects the other lines in the order $1, \ldots, n$. This is perfectly in accordance with the situation in the dual setting: The corresponding point 0 has a smaller x-coordinate than all other points, and the counterclockwise cyclic order of the other points is $1, \ldots, n$.

This extended point set has $n+1$ points, and it has a special point 0 on the boundary. We see that this is equivalent to an arbitrary set of $n+1$ points where some *pivot point* on the convex hull is marked. By a projective transformation, the pivot point can be moved far up without changing the order type.

Thus we have explained the three boxes in the bottom row of Figure 6. The boxes refer to *oriented* abstract order types (OAOTs), because at this point, we still distinguish a point set from its reflection.

We are, however, interested in point sets without a marked pivot point. Therefore we must study what it means to select another hull point as the pivot point. This is best understood by looking at pseudoline arrangements in the projective plane. As the model of the projective plane, we use the sphere in which opposite points are identified. Figure 8 shows a picture, where the image of the PSLA to which we are used appears on the "front half" of the sphere in the left part of the picture. The "back half" of the sphere, which carries the centrally reflected PSLA, is unfolded into the right part of the picture, so that we look at both parts from the outside. On the sphere, each pseudoline becomes a closed cycle. The line 0 "at infinity" is the cycle that separates the front part from the back part. The dashed lines connect points that are identical, as the front part and the back part are stitched together.

Now, on this spherical model, we have $n+1$ lines. They are all equal; line 0 does not play a distinguished role. In fact, the *succ* and *pred* pointers allow navigation on the sphere just fine. If we follow the *succ* pointers along some line j without caring to stop when we cross line 0, we will simply traverse the whole cycle again and again.

We have obtained our x-monotone pseudoline arrangement because we know which circle is line 0, and moreover, we have marked two opposite faces within this circle as the bottom

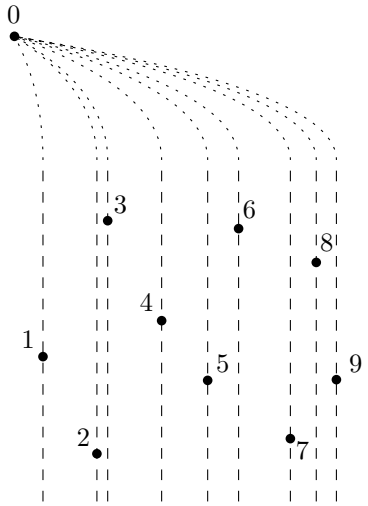


Figure 7 Modeling the x-sorted order of a point set by an extra point 0.

face B and the top face T.

We can obtain another x-monotone pseudoline arrangement from the same projective class by declaring a different line to be line 0, and marking the faces that should become the bottom and top faces. One such choice is indicated by the labels $0', B', T'$ in Figure 8. A different way to express this is to say that we pick a directed edge as a *starting edge*, namely the edge of cycle 0 that has the face T on its left.

This discussion covers the boxes on the left side of Figure 6. As an intermediate notion, we have *affine* (or *Euclidean*) PSLAs, where the line at infinity is fixed, but it has not been decided which unbounded faces are the bottom and the top faces. Figure 6 includes some intermediate boxes, in which some data are partially fixed, and it shows a few more translations between the pseudoline world and the point-set world, but we don't discuss them here.

If a different starting edge has been chosen, it is not hard to realize this in the data structure. The graph is the same as before; one just has to relabel the lines. The line through the starting edge becomes line 0, and the other lines get the labels $1, 2, \ldots, n$ in the order in which they are crossed by line 0, starting from the starting edge. We simply need to carry out this relabeling for j, k, and i or i' in all relations $succ[j,k] = i$ and $pred[j,k] = i'$.

3.1 Relation to oriented matroids

For the benefit of the reader who is familiar with oriented matroids, we point out the (somewhat subtle) correspondences between some of the concepts shown in Figure 6 and oriented matroids. As mentioned, pseudoline arrangements correspond to oriented matroids of *rank 3*. We deal exclusively with *simple* PSLAs (general position, with no three lines going through a common point). These correspond to *uniform* oriented matroids of rank 3. Throughout, we deal with *labeled* objects, i.e., oriented matroids on a fixed ground set $\{0, 1, \ldots, n\}$.

In the oriented matroid world, every pseudoline has a positive and a negative side. In the pseudoline world, these distinctions are not made, and hence, PSLAs correspond to *reorientation classes* of oriented matroids.

The box in the upper-left corner, projective PSLAs with $n + 1$ lines, corresponds exactly

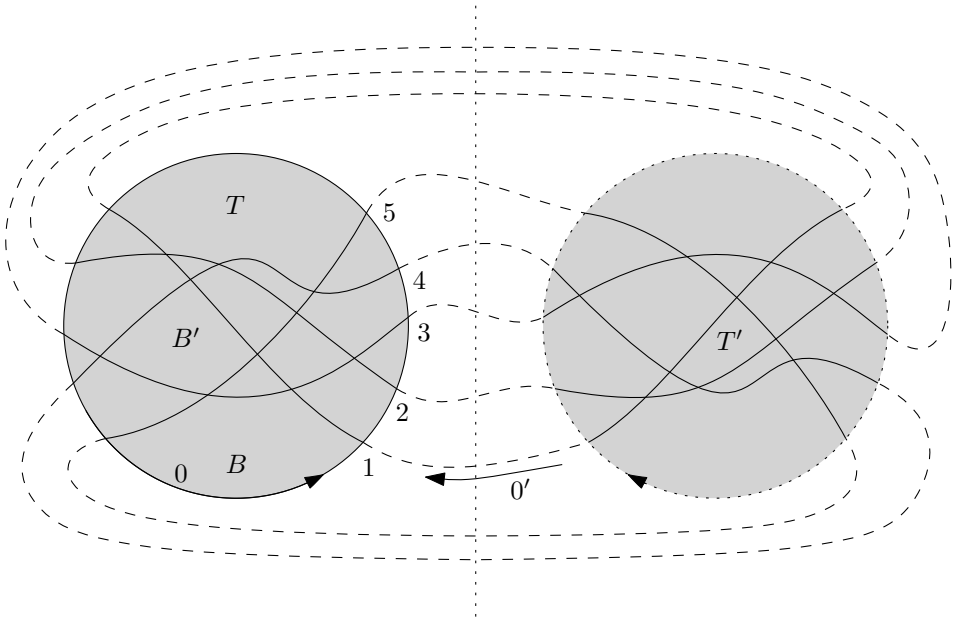


Figure 8 The spherical model of a projective PSLA, for the PSLA of Figure 4

to reorientation classes of uniform rank-3 oriented matroids.

The objects of the top middle box, projective PSLAs with a marked face, *almost* correspond to *acyclic* uniform rank-3 oriented matroids, those having a face (covector) with sign sequence $(+,+,\ldots,+)$. The exception is that a projective PSLA with a marked face is distinguished from its mirror image, while an oriented matroid (despite its name) is insensitive to a global reorientation. Thus, if we identify each projective PSLA with a marked face with its mirror image, we obtain exactly the acyclic uniform rank-3 oriented matroids. By polarity, these correspond exactly to AOTs, i.e., (unoriented) abstract order types.

3.2 The convex hull in the pseudoline world

As is well-known, the convex hull of a point set consists of those points whose dual line is incident to the top face or the bottom face. However, when applying this criterion, we must add line 0 as the line with the most negative slope, as illustrated in Figure 9. Then there are only two lines incident to the bottom face: lines 0 and n. But these two lines are anyway also incident to the top face. Thus, in our setting, the convex hull vertices correspond to the lines of the top face (including line 0).

3.3 The orientation predicate

The *succ* and *pred* arrays are useful for navigating in the arrangement, but to get the full power of working with an abstract order type, one needs the orientation predicate. In terms of pseudolines, the orientation is defined as shown in Figure 10. For three lines $i < j < k$ the orientation is determined by looking at the triangle formed by these lines. The orientation $\mathit{orient}(i,j,k)$ is positive if the triangle lies below j and negative otherwise. The orientation is unchanged under an even permutation of the parameters (i,j,k), and it is flipped by an odd permutation of the parameters (i,j,k). This orientation agrees with the orientation of the corresponding point set, in case we apply the duality (1) to a proper line arrangement, in

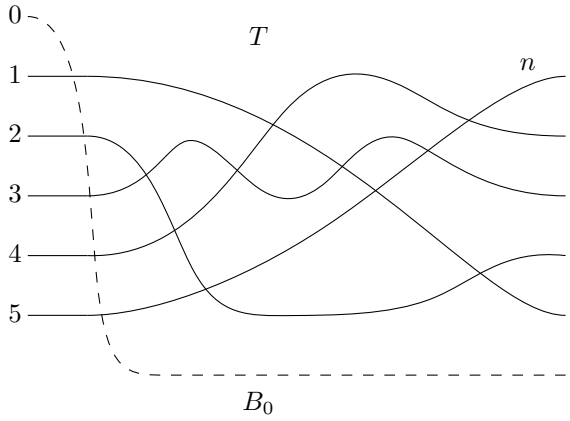


Figure 9 The convex hull in a pseudoline arrangement

the following sense: For three lines $i < j < k$ with dual points p_i, p_j, p_k, the crossing of i and k lies below j if and only if p_j lies below the segment $p_i p_k$, i.e., if and only if p_i, p_j, p_k are in counterclockwise order. (For triples involving line 0, this requires that the dual point 0 lies far above and to the left of the other points, see below.) Extending the definition to pseudoline arrangements is in fact one way to define abstract order types.

Now, for $i < j < k$, as shown in the picture, the orientation can be figured out if one knows the order of the crossings along line j, for example: Is the crossing (j, i) to the left or to the right of (j, k)? This information is not available, but it can easily be provided by preprocessing.

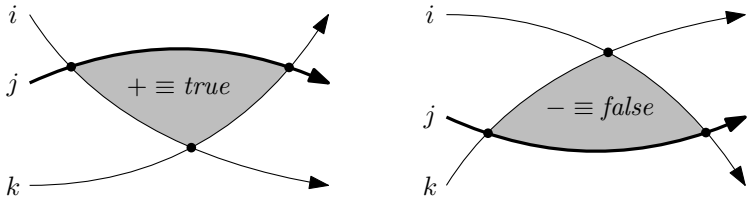


Figure 10 The orientation of three lines

Thus, when we want to work with a PSLA, we prepare additional data structures, the *local sequences array* P and the *inverse local sequences array* $\bar{P}$.

3.4 The local sequences matrix and its inverse

Here is a representation as a two-dimensional array. For each pseudoline i, the sequence P_i indicates the sequence of crossings with the other lines, starting at 0 and moving to the right. For the example in Figure 1, the local sequences are as follows:

$$
\begin{array}{ll}
P_0 = [1,2,3,4,5] & \bar{P}_0 = [\text{-},0,1,2,3,4] \\
P_1 = [0,2,3,4,5] & \bar{P}_1 = [0,\text{-},1,2,3,4] \\
P_2 = [0,1,4,3,5] & \bar{P}_2 = [0,1,\text{-},3,2,4] \\
P_3 = [0,1,4,2,5] & \bar{P}_3 = [0,1,3,\text{-},2,4] \\
P_4 = [0,1,3,2,5] & \bar{P}_4 = [0,1,3,2,\text{-},4] \\
P_5 = [0,1,2,3,4] & \bar{P}_5 = [0,1,2,3,4,\text{-}]
\end{array}
$$

The first row P_0 and the first column are fixed. Each row P_i consists of n different elements, excluding the element i itself. The inverse local sequence $\bar{P}_i$ is essentially the inverse permutation of P_i: The j-th element of $\bar{P}_i$ gives the position in P_i where the crossing with j occurs, where positions are counted from 0. The diagonal entries are irrelevant. From the *succ* links, it is straightforward to build the local sequences and the inverse local sequences, in $O(n^2)$ time. With the help of $\bar{P}$, the orientation predicate can be evaluated in constant time as the exclusive-or of three simple tests:

$$\mathit{orient}(i,j,k) \equiv (i<j) \oplus (j<k) \oplus (\bar{P}[j,i] < \bar{P}[j,k])$$

Here we use a Boolean value instead of a sign $\pm$. It is clear that this formula is correct for the standard case $i<j<k$, and it is easy (but tedious) to check that it works for all other orderings of i,j,k.

4 Elimination of duplicate AOTs

An abstract order type with h hull vertices corresponds to $2h$ PSLAs: Each of the h hull vertices can become vertex 0, and there are two choices of orientation. (If there are symmetries, some of these $2h$ PSLAs will coincide.) The standard approach to tackle this problem is to compute some sort of canonical representation. Since a pseudoline arrangement is a plane graph, one may apply isomorphism testing algorithms for plane graphs.

Indeed, such methods are available, due to Hopcroft and Wong [21] from 1974, for (3-connected) planar graphs, and a simplified method of Akutsu, de la Higuera, and Tamura [6] for plane graphs with a fixed outer face. Both methods run in linear time in the size of the graph, which is $O(n^2)$ in our case, and they produce a canonical labeling as well as the automorphism group of the graph. We have to adapt the input, because pseudoline arrangements have infinite rays. A simple way to treat these rays is to enclose the arrangement by a ring and clip the rays at the ring, generating a cycle of $2n$ degree-3 vertices.

(We mention another special algorithm that tests (abstract) *order-type* isomorphism of *point sets* and determines the automorphism group in $O(n^2)$ time [7]; this algorithm attacks the problem directly and does not rely on planar graph isomorphism via duality.)

However, instead of applying these general techniques for planar graphs, we apply a simpler, tailored method, which runs very fast in practice. To eliminate duplicates, we compare the local sequences matrices P lexicographically. The algorithm reports an AOT A only if the P-matrix of the PSLA at hand is the smallest in the class of PSLAs that represent A. Conceptually, we proceed as follows: Let $A^1 = A, A^2, A^3, \ldots, A^{2h}$ be the $2h$ ways of labeling the point set A, preserving the set of convex hull vertices. $A^1, A^2, A^3, \ldots, A^h$ are the *forward candidates* that are generated by cyclic rotation, and $A^{h+1}, A^{h+2}, \ldots, A^{2h}$ are the *reversed candidates* that involve a reversal. We then look at the corresponding local sequences matrices $P^1, P^2, \ldots, P^{2h}$. P^1 represents the pseudoline arrangement currently at hand, and $P^2, \ldots, P^{2h}$ are its competitors. If the current matrix is not the smallest, we discard the current PSLA. On this occasion, we will also find out when some of the other P-matrices are equal to P^1. This indicates the presence of a symmetry. The symmetry may be a rotational symmetry, rotating the convex h-gon by some number of vertices. (The first P^j that is equal to P^1 determines the period $j-1$, which must be a divisor of h.) A mirror symmetry exists when P^1 is equal to one of the reversed candidates $P^{h+1}, P^{h+2}, \ldots, P^{2h}$. It can occur alone or in combination with a rotational symmetry.

There are several considerations that play a role in practice:

1. The average number h of sides of the convex hull is a little bit less than 4, see Table 2. This confirms theoretical predictions of Goaoc and Welzl [19]. They showed that for

labeled order types (where symmetries don't matter), the average size of the convex hull is

$$4 - \frac{8}{n^2 - n + 2}. \tag{2}$$

This statement holds both for AOTs [19, Theorem 10.2] and for (realizable) *order types* [19, Theorem 1.2]. In the latter setting of order types, convergence to 4 carries over to the unlabeled case [19, Theorem 1.3]. In our setting of unlabeled abstract order types, no such convergence result has been proved. Nevertheless, Table 2 shows that formula (2) seems to give a very precise estimate even in this setting.

2. The vast majority of AOTs have no symmetries. (See Table 5 in Section 8.3 for exact counts of symmetric versus asymmetric AOTs.) Thus we can assume that roughly only one out of 8 PSLAs is the lex-min PSLA, and 7 out of 8 are generated in vain. One can check that the numbers of Table 1 are in accordance with this. The 112,018,190 PSLAs with 9 lines give rise to only 14,320,182 AOTs with 10 points. The ratio is 0.127838, just barely larger than 1/8.
3. Most of the total runtime for enumeration is spent in the lex-min test at the leaves of the tree.

In practice, it would be wasteful to compute the complete matrices $P^1, \ldots, P^{2h}$ in advance, which would take $\Theta(hn^2)$ time. We compute the first entry of each matrix and compare these entries. It may turn out at this point that P^1 has already lost, and we can quickly abandon the comparison. Some other matrices might also be out of the game, and they are discarded. For the matrices that remain, we compute the second entry, and so on (see also [5, p. 4]). The comparison will only go to the very end if some matrices are equal, and this can only happen in case of symmetry. As mentioned, symmetric solutions are a tiny fraction.

4.1 Early screening of pseudoline arrangements

The way we compare the local sequences matrices in the lexicographic order is row-wise from right to left. That is, we start with the right-most entry P_{1n} in the first row P_1. (Row P_0 is always the same. In contrast to the convention for $\bar{P}$, we now count the columns j of the entries P_{1j} from 1 to n.) The reason for this unusual choice is that, in some preliminary tests, it seemed to be more effective in connection with the screening approach that is described below.

We have mentioned that the effect of choosing a different starting edge consists of relabeling all lines. Thus, in order to compute the matrices $P^2, \ldots, P^{2h}$, we compute a renaming table for each matrix. This takes $O(n)$ time per matrix, by simply following the pointers along one pseudoline. This task has to be completed before the first matrix entry is even looked at.

To speed things up, we sidestep the renaming table and compute the entry P_{1n} (and only this entry) directly. The meaning of P_{1n} is the (label of the) last line ℓ intersected by line 1. This label is defined by how far away the intersection of ℓ is from the start, when walking along line 0. This interpretation can be used to determine the value of P_{1n} even with incorrect labels, by simply walking along line 0 (in a *pedestrian* way, so to speak).

If, for example, one of the other matrices P^i has a smaller value P^i_{1n} than P^1_{1n} in the matrix P^1, we immediately conclude that P^1 is not lex-min, and we have saved a lot of work. A matrix P^i with $P^i_{1n} > P^1_{1n}$ can be excluded from further consideration, and hence its renaming table need not be computed. The details are a bit tricky, and they are explained in the documentation of the program. This screening test is quite effective. For example, there

are 18,410,581,880 PSLAs with $n = 10$ lines. Of these, only 5,910,452,118 pass the screening test. Eventually, only 2,343,203,071 PSLAs are really lex-min, and this is the number of AOTs that we really want.

For those cases that pass the screening test, it turns out that the lex-min testing procedure is quite fast: When enumerating AOTs with $n \geq 10$ points, on average, a lex-min test had to look at fewer than 6 entries in total before it could make a decision. This total is over all matrices $P^1, \ldots, P^{2h}$ taken together. (This does not include the $2h$ entries P^i_{1n} that were compared in the screening test. The screening test eliminates some of the $2h$ candidates, but for the surviving candidates, the lex-min test looks at the entries P^i_{1n} again, for uniformity. By adapting the code, the 6 entries that are looked at on average could be further reduced.)

4.1.1 More aggressive pre-screening at the next-to-last level

In some cases, it can be determined already at level $n-1$ that there is no way that the insertion of line n into the current PSLA can lead to a lex-min PSLA. In this case, we can abandon the procedure right away, instead of generating all children in the tree and subjecting them to the lex-min test. The details are described in the documentation of the program.

5 Parallelization

We have implemented a trivial way to parallelize the enumeration. The user can choose a *split level*, usually 8. The program will then work normally up to level 8 of the tree, that is, it will enumerate all 1,232,944 PSLAs with 8 lines, but it will only expand a selection of these PSLAs. The selection is determined as follows. As the PSLAs with 8 lines are enumerated, a running counter is incremented, thus assigning a number between 1 and 1,232,944 to each PSLA. We specify a modulus m and a value k. Then the program will expand only those nodes whose number is congruent to k modulo m. By running the program for $k = 1, \ldots, m$, the work is split into m roughly equal parts. The same parallelization technique is used in the `gtools` graph generation programs that are part of the `nauty` system[2], see [27, Section 8].

6 Enumerating only the realizable AOTs

We have implemented a provision to enumerate only the (realizable) order types of point sets, for up to 11 points, to make the results comparable with those of the order-type database: There is an option to specify an *exclude-file* for the program. The exclude-file is a sorted list of decimal codes for tree nodes that should be skipped.[3] The exclude-files were prepared with the help of the order-type database. Essentially, we are storing the AOTs that are *not* realizable, which is a tiny minority compared to the realizable ones, see Table 1. Still, the exclude-file for up to 11 points has 7,710,079 entries and needs 161.9 MBytes. (With some technical effort, like eliminating common prefixes or a compressed binary format, one could reduce this space requirement significantly.)

[2] `https://users.cecs.anu.edu.au/~bdm/nauty/` or `https://pallini.di.uniroma1.it/`

[3] Currently the exclude-file feature does not work together with the parallelization feature. (For 11 points, the program should anyway be fast enough without parallelization.)

7 Symmetries of x-monotone PSLAs

For *x-monotone* PSLAs, there are three natural symmetry operations.

a) A horizontal reflection (with a vertical mirror axis) keeps the bottom and the top faces fixed and exchanges the *pred* and *succ* pointers. The lines are traversed in the opposite direction, and they are renumbered from $1, 2, \ldots, n$ to $n, n-1, \ldots, 1$. (Such a symmetry corresponds to a mirror symmetry of the associated AOT, with the mirror going through the pivot point.)

b) A vertical reflection (with a horizontal symmetry axis) swaps the bottom face with the top face. As above, the lines are renumbered from $1, 2, \ldots, n$ to $n, n-1, \ldots, 1$.

c) A 180° rotation is obtained as a combination of the above operations. The lines are not renumbered, but they are traversed in the opposite direction.

These symmetries preserve many properties of PSLAs, for example, the number of cutpaths. By taking advantage of these symmetries, one can typically reduce the amount of computation for experiments by a factor of up to four.

A PSLA with an odd number $n \geq 3$ of lines cannot have a horizontal symmetry axis: The middle line $(n+1)/2$ must pass either above or below the crossing of 1 and n, and this property is inverted by a reflection at a horizontal axis.

A PSLA with $n \geq 3$ lines cannot have a 180° rotation. For $n = 3$ (and general odd n), the same argument as above can be applied. If $n \geq 4$, then lines 1 and n partition the plane into four sectors. The crossing of lines 2 and $n-1$ is in one of these four sectors, and the rotation moves it to the opposite sector.

8 Computational results

As mentioned, going through all 12-point AOTs takes around 200 CPU hours. We also ran the program for 13 points on a parallel compute-cluster [8], which took about 3200 CPU days of computing time. The number h of hull vertices and the symmetry are already computed as part of the lex-min test; thus we might as well record these data.

For the purpose of illustration, we decided to collect some more statistics: the number of halving lines and the crossing number.

These data can be computed from the wiring diagram: By the duality (1), the number of crossings at level k in the wiring diagram is the number of lines through pairs of points that have k points *above* them, and hence it is clear that these numbers are related to k-edges and k-sets. In particular, by counting the crossings at the different levels, one immediately obtains the number of halving lines. By a remarkably simple formula of Lovász, Vesztergombi, Wagner, and Welzl [26], the number of crossings can be calculated directly from the number of k-edges for all k.

Since we did not know what interesting phenomena might emerge from the data, we decided not to do any aggregation during the enumeration. We maintain the number of AOTs for each combination of the characteristics (n, h, symmetry, halving lines, crossings), and in the end, we write the nonzero numbers to a log-file, thus relieving the enumeration of the task of making a statistical analysis. The result file with these raw data is available in the repository.[4] All statistics reported in Sections 8.1–8.3 were extracted from these files by PYTHON scripts.

[4] [31], file `results/crossing+halving-results-13.txt`; the data for the (realizable) OTs in Table 6 are in the file `results/realizable11-10.txt`.

8.1 Number of convex hull points

Table 2 counts the AOTs by size of the convex hull h. This can be compared to Table 2 in [5], where the corresponding data are given for (realizable) order types up to $n = 11$.

	$n=7$	$n=8$	$n=9$	$n=10$	$n=11$	$n=12$	$n=13$
$h=3$	49	1,178	55,239	4,879,546	786,103,220	229,258,881,954	120,410,822,315,097
$h=4$	59	1,468	70,482	6,324,559	1,031,019,051	303,315,298,426	160,356,153,417,352
$h=5$	22	570	28,234	2,630,639	440,348,013	132,120,240,798	70,900,318,730,166
$h=6$	4	90	4,552	450,300	79,039,502	24,562,198,935	13,533,084,234,118
$h=7$	1	8	311	33,969	6,447,723	2,124,883,478	1,222,365,995,348
$h=8$		1	11	1,146	241,522	87,484,087	53,890,715,843
$h=9$			1	22	4,006	1,683,531	1,154,715,041
$h=10$				1	33	14,410	11,618,261
$h=11$					1	62	51,210
$h=12$						1	101
$h=13$							1
sum	135	3,315	158,830	14,320,182	2,343,203,071	691,470,685,682	366,477,801,792,538
average h	3.8815	3.8793	3.8935	3.913 29	3.928 582	3.940 299 5	3.949 367 11
(2)	3.8182	3.8621	3.8919	3.913 04	3.928 571	3.940 298 5	3.949 367 09

Table 2 Number of abstract order types with n points in total and h points on the convex hull. The last row is the value of formula (2), which has been discussed in Section 4 on p. 12.

	$h=3$	$h=4$	$h=5$	$h=6$	$h=7$	$h=8$
$n=10$	0.340746	0.441654	0.183702	0.031445	0.002372	0.000080
$n=11$	0.335482	0.440004	0.187926	0.033731	0.002752	0.000103
$n=12$	0.331553	0.438652	0.191071	0.035522	0.003073	0.000127
$n=13$	0.328562	0.437560	0.193464	0.036927	0.003335	0.000147

Table 3 Relative frequencies of convex hull sizes h

Table 3 shows the relative frequencies of the various convex hull sizes h, for the larger values $n = 10, 11, 12, 13$. They seem to converge to some limiting distribution. We are not aware of any theoretical results that would predict the limiting frequency of, say, triangular convex hulls. This should be related to the expected number of triangular faces in a "random" PSLA.

8.2 Crossing numbers and halving lines

Table 4 deals with the number of crossings in a straight-line drawing of the complete graph. We report results only for 12 points. The smallest number of crossings is 153 (which has been known for a long time), and is achieved by a unique AOT. The largest number of crossings is $495 = \binom{12}{4}$, and is achieved by a unique AOT, namely by points in convex position. The next-largest number of crossings is 486, and it is again achieved by a unique AOT. There are a few more gaps, as visible in the table. For every number X in the range 153–459, there is at least one AOT with that number of crossings. The most frequent number of crossings is $X = 252$; we can see that the frequencies do not vary monotonically but fluctuate up and down in the vicinity of this value.

Figure 11 shows the joint distribution of both parameters, the crossing number and the number of halving lines.

X	#AOT		X	#AOT		X	#AOT	X	#AOT	X	#AOT
153	1		250	9,599,727,792		451	41	459	11	470	11
154	15		251	9,774,280,765		452	76	461	41	471	1
155	215		252	10,813,519,833		453	68	462	12	472	1
156	1,354		253	10,549,648,258		454	119	463	21	474	1
157	4,066	⋮	254	9,551,226,473	⋮	455	33	464	1	477	5
158	6,966		255	9,720,622,387		456	46	465	10	479	1
159	13,904		256	10,543,935,293		457	1	467	2	486	1
160	42,950		257	10,332,151,661		458	38	468	7	495	1

Table 4 The number of AOTs of 12 points with X crossings, for selected values of X

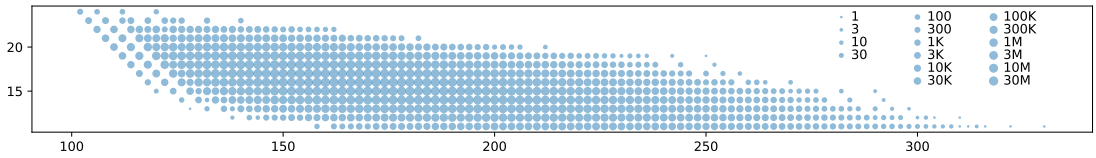


Figure 11 Scatter-plot of crossing number (horizontal axis) versus number of halving lines (vertical axis) for AOTs with $n = 11$ points. The area of each dot represents the frequency, on a logarithmic scale. Thus the tails are really not as heavy as they appear. One can see that the crossing number and the number of halving lines are negatively correlated. The crossing number ranges between 102 and 330, and it is always an even number. The number of halving lines ranges between 11 and 24.

8.3 Symmetries

As mentioned in Section 4, our algorithm delivers the symmetries of an AOT for free, as part of the lex-min test that is necessary to pick a single PSLA among the several PSLAs representing the AOT. Table 5 classifies the AOTs (first column) according to the types of symmetries that they have. The second column gives the AOTs that have no symmetry at all, and these are the vast majority. The third column counts AOTs that have a mirror symmetry (possibly including a rotational symmetry as well). The fourth column gives the AOTs that have a non-trivial symmetry that is purely rotational (without mirror symmetry). The first column is the sum of columns 2–4.

A mirror symmetry will *reverse* all orientations, and thus, there can be different opinions whether it should be regarded as a symmetry operation. The last column is the number of *oriented AOTs*, where an AOT and its mirror are counted as distinct objects (unless the AOT is mirror-symmetric). It is obtained by taking columns 2 and 4 twice and adding column 3 once.

Table 6 gives a more refined account of columns 3 and 4 of Table 5, classifying AOTs by their symmetry groups. We use the same notations C_k and D_k as for the symmetry groups of finite objects in the plane (the cyclic and dihedral groups), although our groups act in a purely combinatorial way on AOTs, by permuting the points. Since such a symmetry must preserve the convex hull vertices and the adjacency between them, it must be isomorphic to a subgroup of the symmetry group of a regular h-gon, if there are h hull vertices.

D_k is the symmetry group of a regular k-gon: the *dihedral group* of order $2k$. For each n, we have one AOT with symmetry group D_n, namely convex position, which corresponds to the regular n-gon in the geometric setting. In addition, if n is even, we can have an

n	[A006247] (unoriented) AOTs	asymmetric AOTs	mirror-sym. AOTs	rot. sym. AOTs	[A006246] oriented AOTs
3	1	0	1	0	1
4	2	0	2	0	2
5	3	0	3	0	3
6	16	4	12	0	20
7	135	105	28	2	242
8	3,315	3,085	225	5	6,405
9	158,830	157,981	825	24	316,835
10	14,320,182	14,306,748	13,103	331	28,627,261
11	2,343,203,071	2,343,126,871	76,188	12	4,686,329,954
12	691,470,685,682	691,468,293,616	2,358,635	33,431	1,382,939,012,729
13	366,477,801,792,538	366,477,779,812,782	21,954,947	24,809	732,955,581,630,129

Table 5 AOTs with various symmetries. The column headings link to the corresponding entries of the On-Line Encyclopedia of Integer Sequences [33].

$(n-1)$-gon with a point in the center, having symmetry group D_{n-1}. The most frequent group is the group D_1, which has a single mirror symmetry as the only nontrivial element.

C_k is the rotational symmetry group of a regular k-gon, i.e., a k-fold rotation. C_2 corresponds to a rotation by 180°, or equivalently, a reflection in a central point.

We see that many fields in the table are empty. There are systematic reasons for this. For example, if a set of n points has a rotational symmetry of order 3 (C_3, or any of its supergroups D_3, C_6, D_6, D_9, or D_{12}), then n must be a multiple of 3 or a multiple of 3 plus 1 (with a fixpoint in the "center"), cf. [19, Theorem 1.5]. A C_2 symmetry cannot exist for odd n, because it would have a fixpoint in the center, and "opposite" points would have to be aligned with the center, which is excluded in a simple AOT.

Counting of AOTs by enumerating symmetric AOTs. There is a relation between the number of AOTs with prescribed symmetries and the number of PSLAs: Each AOT corresponds to a certain number of PSLAs, depending on the symmetry group. Thus if we know the entries in Table 6, together with the asymmetric AOTs in the second column of Table 5, we can work out the number of PSLAs. (This is actually how the program computes the correct number of PSLAs even though it prunes branches of the enumeration tree and does not visit each PSLA individually.)

We could use this relation in the other direction. We might think about counting the various symmetric AOTs for $n > 13$ by enumerating them directly, since their number should still be manageable. Together with the number of PSLAs, which is known up to 16 lines, we can then calculate the number of asymmetric AOTs, and hence the total number of AOTs.

8.4 The number of cutpaths of a PSLA

An extra pseudoline that runs from the bottom face to the top face in a given PSLA is called a cutpath (see Figure 4). The number of cutpaths is equal to the number of children of the corresponding node in the enumeration tree. Figure 12 shows the distribution of the number

n	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	C_2	C_3	C_4	C_5	C_6
3			1															
4			1	1														
5	2				1													
6	7	1	2		1	1												
7	26		1				1								2			
8	218	4		1			1	1						4		1		
9	818		6						1						24			
10	13,059	27	11		4				1	1				234	93		4	
11	76,186				1						1						12	
12	2,358,210	303	111	7		2					1	1		29,573	3,765	86		7
13	21,954,912		34										1		24,809			
(realizable) OTs																		
9	818		6						1						23*			
10	13,021*	27	10*		4				1	1				234	84*		3*	
11	75,802*				1						1						7*	

Table 6 The symmetric AOTs according to their symmetry group. The last three rows concern OTs, for those cases where the set of OTs is known ($n \leq 11$) and differs from the set of AOTs ($n \geq 9$). The differences to AOTs are marked. The majority of the difference set (column Δ in Table 1) belongs to the class C_1 with no symmetries at all, which is not shown in this table.

of cutpaths for the PSLAs with 8 pseudolines.[5,6]

9 Estimating the number of PSLAs with larger n

Knuth [22] observed that there is an easy method to obtain an unbiased estimate for the number of leaves of a tree, by making a random "dive" into the depth of the tree: Iteratively proceed to a random child and multiply the encountered vertex degrees, see also [24, Section 7.2.2, pp. 46–51, Corollary E]. It is straightforward to prove that the expected value of this estimate is indeed the number of leaves of the tree.

In this way we can estimate the size of the deeper levels of the enumeration tree, beyond the range that can be explicitly enumerated one by one. The procedure can be easily adapted to estimate the number of nodes on *each level* of the tree, up to a maximum depth, or to give an unbiased estimate for the number of OAOTs and AOTs.

For each node we need to compute the number of cutpaths (= the number of children), and, except at the lowest level, pick a random child. The number c of cutpaths equals the number of source-to-sink paths in the dual DAG of the PSLA (see the right part of Figure 4). It can be computed by a traversal of the graph in topological order in linear time in the size of the graph, i.e., in $O(n^2)$ time. In the same time, we can also, for a random number r

[5] Frequencies of the number of children have been computed for PSLAs with up to 10 pseudolines; the data are available in the repository [31, `results/cutpaths+grandchildren-results-10.txt`].

[6] We remark that Knuth [23] has computed some statistics about the number of cutpaths, in particular about the minimum, the maximum and the average [23, Table at the bottom of p. 39]. However, he counts a different class of cutpaths, namely the paths from the top face T to the bottom face B in the spherical arrangement, see Figure 8 and [23, Example (9.9) on p. 38]. (Note that the conjecture about the maximum number $n2^{n-2}$ of cutpaths that was expressed in this context has been disproved by Ondřej Bílka; see [15, 12] for details.)

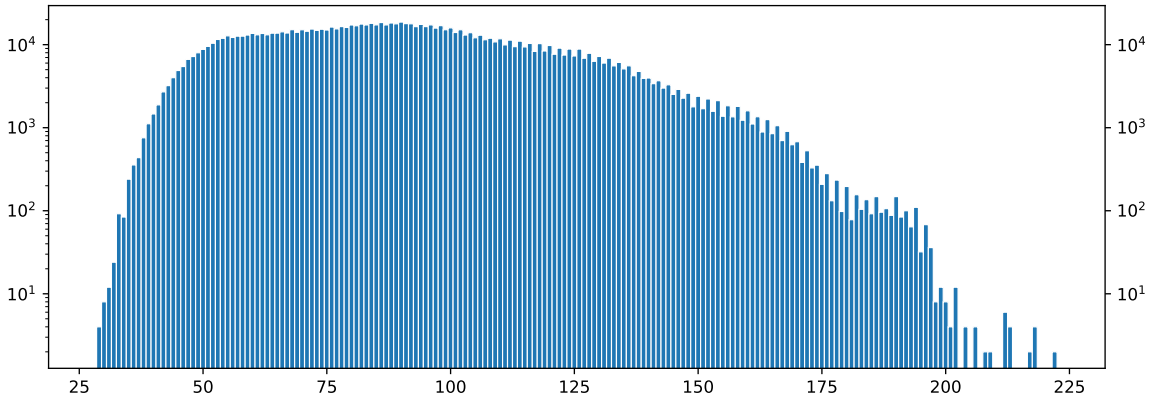


Figure 12 The distribution of the number of cutpaths of the 1,232,944 PSLAs with 8 pseudolines. The frequencies are on a logarithmic scale. For symmetry reasons, every number of cutpaths occurs with an even frequency (see Section 7). The number of cutpaths ranges between 29 and 222, and the average is 90.85, which equals the quotient of the number of PSLAs with 9 and with 8 pseudolines, see Table 1.

between 1 and c, determine the r-th cutpath in lexicographic order. Thus, we can determine a random cutpath in $O(n^2)$ time.

Figure 13 shows the results of two experiments to estimate #PSLA in this way. It is known that the number of pseudoline arrangements grows asymptotically like $2^{\Theta(n^2)}$; more precisely,

$$0.2721 \le \liminf_{n\to\infty} \frac{\log_2 \#\mathrm{PSLA}(n)}{n^2} \le \limsup_{n\to\infty} \frac{\log_2 \#\mathrm{PSLA}(n)}{n^2} \le 0.6496,$$

and it is believed that $(\log_2 \#\mathrm{PSLA}(n))/n^2$ converges to a constant. The best known upper bound (due to Dallant [12]) and lower bound [11] are quite recent and are still being improved.

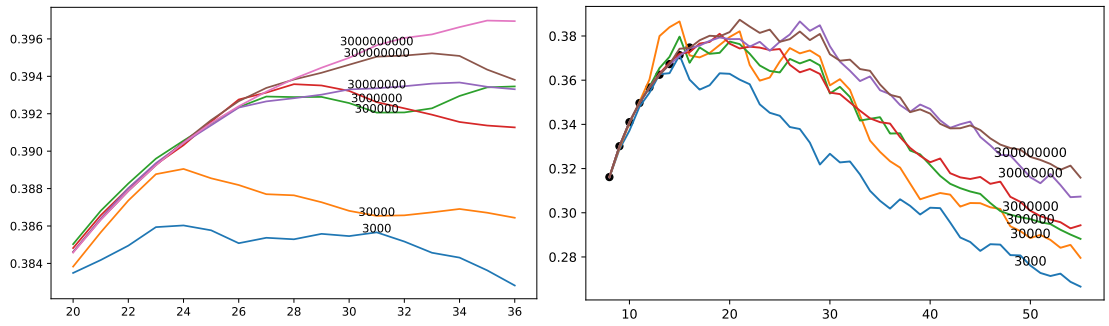


Figure 13 $(\log_2 \bar{B}_n)/n^2$, where $\bar{B}_n$ is the estimate for $\#\mathrm{PSLA}(n)$ obtained as the average of m samples, for various values of m between 3000 and 3 billion. The horizontal axis is the number n of pseudolines. The right graph uses a simpler and faster method.

The left part of Figure 13 shows the estimate of the number of PSLAs with n lines, taking the average $\bar{B}_n$ of the results from 3×10^9 random paths to depth 36, together with some partial estimates obtained along the way from smaller subsamples. In accordance with the exponential growth of #PSLA, we have plotted the quantities $(\log_2 \bar{B}_n)/n^2$.

This method, as currently implemented, reaches its limits somewhere near $n = 70$. When the number of children exceeds 2^{64}, calculating the number of children requires multi-precision

arithmetic, and a more elaborate random number generator would be needed to select a random child. We set the cutoff, beyond which no reasonable random choice can be expected, at a maximum of 2^{55} children. This limit was often exceeded for large samples already for $n = 69$.

We therefore considered a simplified sampling procedure, whose results are shown in the right part of Figure 13. When incrementally inserting the n-th pseudoline, we always pick a random edge on the upper boundary of the current face where the pseudoline should leave the face (see Figure 5). This procedure generates a different enumeration tree for the PSLAs, which is very much refined in comparison to the tree of Figure 3, and it is simpler and much faster: The n-th pseudoline can be inserted with $n - 1$ consecutive random decisions in $O(n)$ time. (By the zone theorem, the total number of edges of the faces that a cutpath visits is $O(n)$.) This allowed us to go up to $n = 200$ pseudolines, taking 300 million samples. Figure 13 shows the estimates up to $n = 55$. The black dots represent the true values of $\#\mathrm{PSLA}(n)$, which are known up to $n = 16$. Comparing to the original experiment in the left graph, we see that the fluctuations are much larger and occur earlier.

Files with the complete results, including also estimates of AOTs and OAOTs, are available in the repository.[7]

In both experiments, we can notice a systematic underestimation, with a few exceptional overestimations. The reason is that the estimate, while having the correct expectation, has an extremely large variation. Most of the subtrees at a given level are smaller than average, and very few subtrees are huge. With a uniform choice of a child, it is unlikely that the rare huge subtrees are hit, and as long as none of these few subtrees is hit, the estimate is too low. The same behavior has been observed in other contexts, for example, when estimating the cost of backtracking trees [22] and branch-and-bound trees.

One can try to counteract this phenomenon by deviating from the uniform selection among the children, favoring the children that have themselves many children. The estimation formula can accommodate any nonuniform selection: Instead of the product of the degrees, one takes the inverse of the product of the probabilities of the chosen children.

In fact, it seems that the children of high-degree vertices tend to have high degree themselves. Figure 14 plots cutpaths of 7-line PSLAs in relation to the average number of cutpaths of their 8-line children. The correlation between the degrees of vertices and the degrees of their children is clearly visible. For PSLAs with 8 or 9 pseudolines, the diagram looks qualitatively the same.[8] This phenomenon exacerbates the poor behavior of a uniform choice of a child. One should deviate from this uniform selection among the children in a stronger way than just by weighting them by *their* number of children. Different strategies in this direction might be evaluated experimentally.

Another experiment that might be worth trying would be to start a single random path at each of the 18 billion nodes at level 10, as opposed to starting, say, 18 billion random paths from the root. It would be interesting to see whether this improves the estimates. Other methods of improving the estimate for the tree size have been proposed, for example *heuristic sampling*, due to Pang C. Chen [10] (see also [25, Section 7.2.2.9]), or *stochastic enumeration*, due to Reuven Rubinstein [32], see also [34]. There is a general framework that encompasses these variations [30].

[7] [31, `results/random-estimate-results-36.txt`, `results/random-face-estimate-results-200.txt`]
[8] The raw data are available in the repository [31, `results/cutpaths+grandchildren-results-10.txt`].

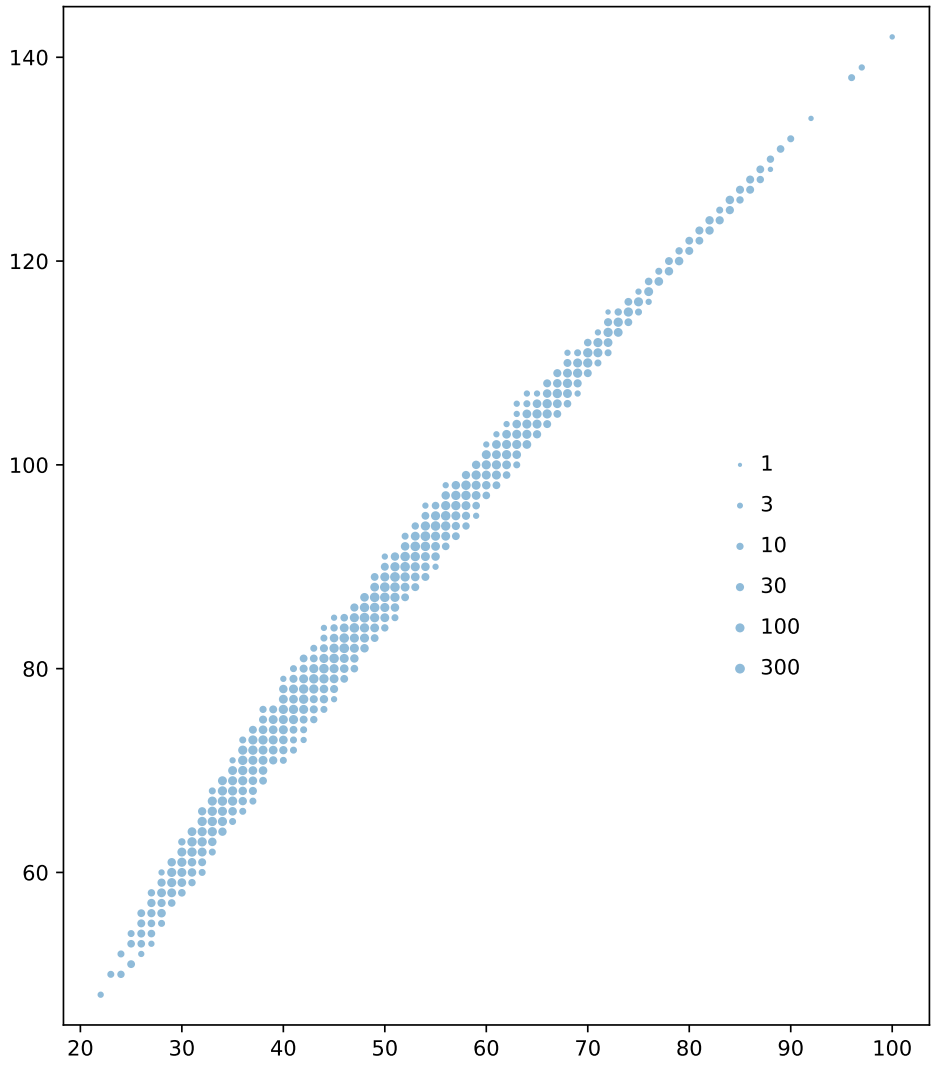


Figure 14 Scatter-plot of children versus grandchildren, for PSLAs with 7 pseudolines. The horizontal axis gives the number of cutpaths of each PSLA, or in other words, the degree of each node (the number of children) in the enumeration tree. The vertical axis gives the average degree of those children, or in other words, the number of grandchildren divided by the number of children, rounded to the nearest integer. The area of each dot represents the frequency, on a logarithmic scale. The number of cutpaths ranges between 22 and 100.

10 Extensions

There are many ways in which one could think of extending the program.

- In this article, we have concentrated on AOTs. We have used PSLAs mainly as a tool to enumerate AOTs, but PSLAs can also be considered in their own right. They might be counted or classified with respect to different criteria, like projective equivalence classes or affine equivalence classes (cf. Figure 6). Each of these classes could be further classified by their symmetries, as in Section 8.3.
- It should not be hard to generate "partial" pseudoline arrangements, in which lines are not forced to cross; see Figure 15 for an example.

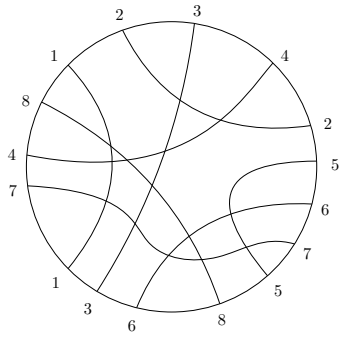


Figure 15 Example of a partial PSLA

- Random generation. It is easy to generate a random PSLA by following a random path into the tree. This will, however, not be a uniform selection.
  To generate uniformly random pseudoline arrangements with a particular number n of pseudolines, say $n = 11$, one could, in a preprocessing stage, compute and store the size of all 112 million subtrees at level 9. This allows one to choose a random subtree according to the correct distribution quickly. A random leaf in that subtree then can be computed by exploring the subtree, which contains on average about 50,000 leaves. In going through these leaves, one does not need to go through every leaf on the last level (level 11); it mostly suffices to determine the number of children at the previous level (level 10). This method should therefore be reasonably fast. A few particularly large subtrees could be further preprocessed to speed up the process. The method can be adapted to generate a random AOT.
- Non-simple pseudoline arrangements, in which more than two pseudolines are allowed to cross in a point, are challenging. In the language of oriented matroids, they correspond to nonuniform oriented matroids, and they have been enumerated by a method of Finschi and Fukuda [17], also in higher dimensions, see [16] for a catalog. These are much more numerous, see also Table 1 in [18], where also the realizability is considered. Handling them by our approach would involve a redesign of the data structures from scratch.
- A side issue are nice drawings of pseudoline arrangements. The wiring diagram is simple to obtain but it is very jagged. (Another representation of pseudoline arrangements via rhombus tilings is even more jagged.) Stretchability of pseudolines to straight lines, on the other hand, is a hard problem, despite the relative scarcity of non-stretchable PSLAs of moderate size. As the number of pseudolines increases, non-stretchable arrangements become more and more dominant. Moreover, when drawing pseudoline arrangements by straight lines such that all vertices of the arrangement are visible, one often cannot avoid having some vertices crammed close together in a tiny area. An algorithm to construct

a drawing in which the pseudolines don't bend too much while keeping good distances between the features would be nice to have.

References

1 O. Aichholzer, F. Aurenhammer, and H. Krasser. Enumerating order types for small point sets with applications. In *Proc. 17th Ann. ACM Symp. Computational Geometry*, pages 11–18, 2001. doi:10.1145/378583.378596.

2 Oswin Aichholzer. Enumerating order types for small point sets with applications. Website of the order-type database. Last accessed July 2025. URL: http://www.ist.tugraz.at/aichholzer/research/rp/triangulations/ordertypes/.

3 Oswin Aichholzer, Franz Aurenhammer, and Hannes Krasser. Enumerating order types for small point sets with applications. *Order*, 19(3):265–281, 2002. doi:10.1023/A:1021231927255.

4 Oswin Aichholzer, Martin Balko, Thomas Hackl, Jan Kynčl, Irene Parada, Manfred Scheucher, Pavel Valtr, and Birgit Vogtenhuber. A superlinear lower bound on the number of 5-holes. *Journal of Combinatorial Theory, Series A*, 173:105236, 2020. doi:10.1016/j.jcta.2020.105236.

5 Oswin Aichholzer and Hannes Krasser. Abstract order type extension and new results on the rectilinear crossing number. *Computational Geometry*, 36(1):2–15, 2007. doi:10.1016/j.comgeo.2005.07.005.

6 Tatsuya Akutsu, Colin de la Higuera, and Takeyuki Tamura. A simple linear-time algorithm for computing the centroid and canonical form of a plane graph and its applications. In *29th Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*, volume 105 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:12, 2018. doi:10.4230/LIPIcs.CPM.2018.10.

7 Greg Aloupis, John Iacono, Stefan Langerman, Özgür Özkan, and Stefanie Wuhrer. The complexity of order type isomorphism. *Discrete Comput. Geom.*, 72(2):483–502, 2024. arXiv:1311.0928, doi:10.1007/s00454-024-00687-1.

8 Loris Bennett, Bernd Melchers, and Boris Proppe. Curta: A general-purpose high-performance computer at ZEDAT, Freie Universität Berlin, 2020. doi:10.17169/refubium-26754.

9 Therese Biedl, Erin Chambers, Irina Kostitsyna, and Günter Rote. Sweeping x-monotone pseudolines. In Shahin Kamali, editor, *Proceedings of the 37th Canadian Conference on Computational Geometry, (CCCG'2025), Toronto*, pages 12–21, 2025. arXiv:2507.21322.

10 Pang C. Chen. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21(2):295–315, 1992. doi:10.1137/0221022.

11 Fernando Cortés Kühnast, Justin Dallant, Stefan Felsner, and Manfred Scheucher. An improved lower bound on the number of pseudoline arrangements. In *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:18, 2024. doi:10.4230/LIPIcs.SoCG.2024.43.

12 Justin Dallant. Improved bound on the number of pseudoline arrangements via the zone theorem. In *Abstracts of the 41st European Workshop on Computational Geometry (EuroCG 2025)*, pages 76:1–76:6, April 2025. URL: https://kam.mff.cuni.cz/conferences/eurocg2025/booklet2025.pdf, arXiv:2502.20909.

13 Stefan Felsner. *Geometric Graphs and Arrangements*. Vieweg, 2004. URL: http://page.math.tu-berlin.de/~felsner/Buch/gga-book.pdf, doi:10.1007/978-3-322-80303-0.

14 Stefan Felsner and Jacob E. Goodman. Pseudoline arrangements. In Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, chapter 5, pages 125–157. Chapman and Hall/CRC, 3rd edition, 2017. doi:10.1201/9781315119601.

15 Stefan Felsner and Pavel Valtr. Coding and counting arrangements of pseudolines. *Discrete Comput. Geom.*, 46:405–416, 2011. doi:10.1007/s00454-011-9366-4.

16 Lukas Finschi. Homepage of oriented matroids. https://finschi.com/math/om/. Accessed 2025-02-25.

17 Lukas Finschi and Komei Fukuda. Generation of oriented matroids—a graph theoretical approach. *Discrete Comput. Geom.*, 27(1):117–136, 2002. doi:10.1007/S00454-001-0056-5.

18 Komei Fukuda, Hiroyuki Miyata, and Sonoko Moriyama. Complete enumeration of small realizable oriented matroids. *Discrete Comput. Geom.*, 49(2):359–381, 2013. doi:10.1007/S00454-012-9470-0.

19 Xavier Goaoc and Emo Welzl. Convex hulls of random order types. *Journal of the ACM*, 70(1):Article No. 8, 47 pp., 2023. doi:10.1145/3570636.

20 Heiko Harborth. Konvexe Fünfecke in ebenen Punktmengen. *Elemente der Mathematik*, 33:116–118, 1978. URL: http://eudml.org/doc/141217.

21 J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 172–184, 1974. doi:10.1145/800119.803896.

22 Donald E. Knuth. Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29:122–136, 1975. URL: https://www.ams.org/journals/mcom/1975-29-129/S0025-5718-1975-0373371-6/.

23 Donald E. Knuth. *Axioms and Hulls*, volume 606 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, 1992. doi:10.1007/3-540-55611-7.

24 Donald E. Knuth. *Combinatorial Algorithms, Part 2*, volume 4B of *The Art of Computer Programming*. Addison-Wesley, 2023.

25 Donald E. Knuth. *Combinatorial Algorithms, Part 3*, volume 4C of *The Art of Computer Programming*. Addison-Wesley, 2027+. In preparation. Draft of Section 7.2.2.9, "Estimating Backtrack Costs" at https://cs.stanford.edu/~knuth/fasc9c.ps.gz, version November 2, 2026.

26 László Lovász, Katalin Vesztergombi, Uli Wagner, and Emo Welzl. Convex quadrilaterals and k-sets. In János Pach, editor, *Towards a theory of geometric graphs*, volume 342 of *Contemp. Math.*, pages 139–148. Amer. Math. Soc., 2004. doi:10.1090/conm/342/06138.

27 Brendan D. McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26(2):306–324, 1998. doi:10.1006/jagm.1997.0898.

28 Jürgen Richter-Gebert and Günter M. Ziegler. Oriented matroids. In Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, chapter 6, pages 159–184. Chapman and Hall/CRC, 3rd edition, 2017. doi:10.1201/9781315119601.

29 Günter Rote. NumPSLA — an experimental research tool for pseudoline arrangements and order types. In *41st European Workshop on Computational Geometry (EuroCG 2025)*, pages 18:1–18:8, April 2025. URL: https://kam.mff.cuni.cz/conferences/eurocg2025/booklet2025.pdf.

30 Günter Rote. Estimating the number of nodes and paths in a DAG by following a random path, October 2026. arXiv:2610.38586.

31 Günter Rote. NumPSLA — an experimental research tool for pseudoline arrangements and order types. GitHub repository, 2024. URL: https://github.com/guenterrote/NumPSLA.

32 Reuven Rubinstein. Stochastic enumeration method for counting NP-hard problems. *Methodology and Computing in Applied Probability*, 15:249–291, 2013. doi:10.1007/s11009-011-9242-y.

33 The On-Line Encyclopedia of Integer Sequences. URL: https://oeis.org/.

34 Radislav Vaisman and Dirk P. Kroese. Stochastic enumeration method for counting trees. *Methodology and Computing in Applied Probability*, 19:31–73, 2017. doi:10.1007/s11009-015-9457-4.

A Benchmark comparison with the order-type database

We compared the usage of the order-type database against our enumeration approach, and we found that generation from scratch can actually compete in terms of runtime. We compared the following two tasks:

A. Read the 14,309,547 order types of 10 points from the database and compute the size of the convex hull. The convex hull can be computed in linear time, since the first point is always a convex hull vertex and the other points are ordered clockwise around this point. The coordinates are 16-bit unsigned integers, and the orientation test is performed by determinant computation in the usual way, with two multiplications, using 64-bit integer arithmetic.

B. Generate the 14,320,182 abstract order types of 10 points by `NumPSLA` and report the size of the convex hull. The size of the convex hull is computed anyway as part of the lexicographic normalization procedure; thus it does not cost any extra runtime.
(With the `-exclude` option, we could also generate the 14,309,547 *realizable* abstract order types, but this did not make a noticeable difference in runtime.)

Both programs took about 10–20 seconds with a slight advantage for one or the other program, depending on the computer.

For task A, typically about 60 % of the total time was "system time", for reading the file, and 40 % was "user time", for the actual computation.

The usual goal is to perform some more time-consuming checks or calculations on each order type. Thus, the time for either reading the point set from the file or for generating it is a minor issue.

B A Python version of the basic enumeration algorithm

The following program will carry out the basic enumeration of PSLAs. The function `recursive_generate_PSLA_start` is the outer recursion, inserting the next pseudoline. The function `recursive_generate_PSLA` is the inner recursion, extending pseudoline n into the next face by crossing an edge. Figure 16 is a refined version of Figure 5, illustrating the meaning of the variables in the inner loop.

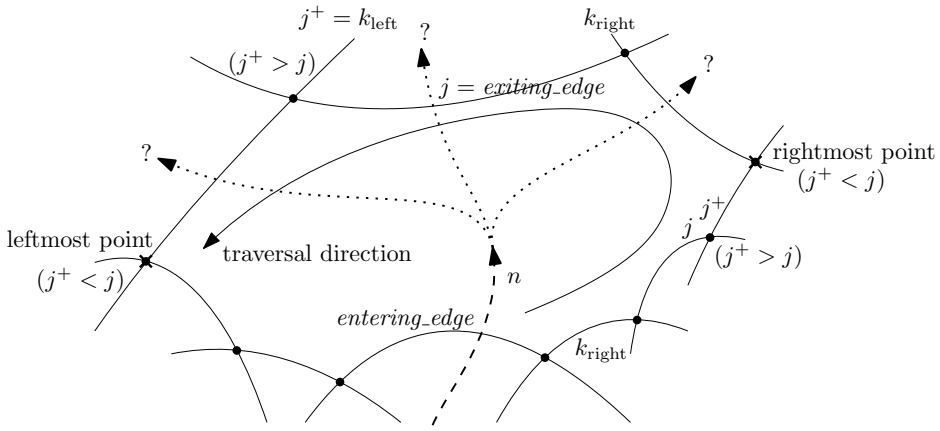


Figure 16 Threading line n through a face

The program is available in the repository under the filename `python/NumPSLA-basic.py`. Starting the program with

```
python python/NumPSLA-basic.py 7
```

will count all x-monotone pseudoline arrangements with at most 7 lines by running through each of them individually. By importing the module `python/wiring_diagram.py`, one can for example modify the program to print wiring diagrams of all arrangements.

```
"The basic framework of NumPSLA, python version"
import sys
# from wiring_diagram import print_wiring_diagram #, IPE_end

def LINK(j, k1,k2): # make crossings with k1 and k2 adjacent on line j
    SUCC[j,k1] = k2
    PRED[j,k2] = k1

def Process_PSLA(n): # insert your code for processing the PSLA here:
    countPSLA[n] += 1
    # print(n,  countPSLA[n], ".".join(str(x) for x in localCountPSLA[3:n+1]))
    # print_wiring_diagram(n, SUCC, ipe=False)

def recursive_generate_PSLA(entering_edge, k_right, n):
    j,jplus = entering_edge,k_right
    while jplus>j: # find right vertex of the current cell F
        j,jplus = jplus,SUCC[jplus,j]
    # the right vertex is at the crossing of j and jplus
    if jplus==0: # F is unbounded
        if j==n-1: # F is the top face.
            LINK(n, entering_edge,0) # complete the insertion of line n
            localCountPSLA[n]+=1
            Process_PSLA(n)
            if n<n_max:
                localCountPSLA[n+1]=0 # reset child counter
                recursive_generate_PSLA_start(n+1) # thread the next pseudoline
            return;
        else: # jump to the upper bounding ray of F
            jplus=j+1; j = 0;
    while True:
        # scan the upper edges of F from right to left and try them out.
        k_right = j;
        j = exiting_edge = jplus;
        k_left = jplus = PRED[j,k_right];
        LINK(exiting_edge, k_left,n); # prepare for the recursive call
        LINK(exiting_edge, n,k_right);
        LINK(n, entering_edge,exiting_edge);

        recursive_generate_PSLA(exiting_edge, k_right, n) # enter the recursion

        LINK(exiting_edge, k_left,k_right); # undo the changes
        if jplus <= j: return
        #terminate at left endpoint of the face F or at unbounded ray (jplus=0)

def recursive_generate_PSLA_start(n):
    LINK(0, n-1,n);
    LINK(0, n,1); # insert crossing with line n on line 0
    recursive_generate_PSLA(0, 0, n);
    LINK(0, n-1,1); # undo the insertion of line n

n_max = int(sys.argv[1])
```

```
52
53  # Start the generation proper:
54  PRED = {}; SUCC = {}
55  LINK(1, 0,0);
56  LINK(0, 1,1);
57
58  countPSLA = [0]*(n_max+1)
59  localCountPSLA = [0]*(n_max+1)
60  recursive_generate_PSLA_start(2)
61  # IPE_end() # finish and close ipe-file, in case it was used.
62  print ("Number of PSLAs:", *countPSLA[2:])
```