Lines: 556

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

Günter Rote
Freie Universität Berlin

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.

2012 ACM Subject Classification Mathematics of computing → Mathematical software performance; Mathematics of computing → Enumeration

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

1 Introduction

Questions about finite configurations of points or lines are a 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. Many 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 crossing number.

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] 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 as much as possible. Later, the database was extended [3] 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[1], 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

[1] http://www.ist.tugraz.at/aichholzer/research/rp/triangulations/ordertypes/

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. [2, 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 [8] 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.

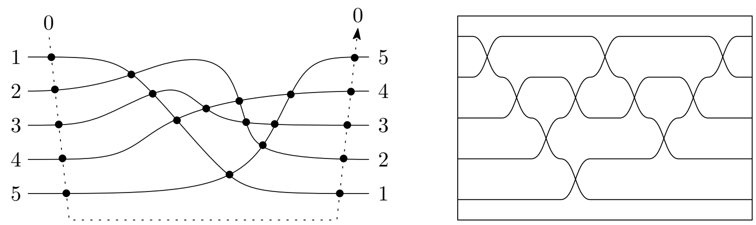
1.1 Line arrangements and pseudoline arrangements

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, as produced by our program. In a wiring diagram, the pseudolines run on n horizontal tracks, and they cross by swapping between adjacent tracks.




```
1 2-2-2-2 4-4-4-4 5
 X       X       X
2 1 3-3 4 2 3-3 5 4
   X   X   X   X
3-3 1 4 3-3 2 5 3-3
     X       X
4-4-4 1 5-5-5 2-2-2
       X
5-5-5-5 1-1-1-1-1-1
```

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

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 [5, Chapter 6].

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 reason is that they are easy to draw and to visualize.

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 Online Encyclopedia of Integer Sequences [13].

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. 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 via anonymous github at `https://anonymous.4open.science/r/NumPSLA-50B7`. It is written in C, using the CWEB system of structured documentation of Donald E. Knuth and Silvio Levy[2]. We have occasionally used the enumeration for research questions (details are withheld for anonymity reasons), and we hope that it finds other users. We encourage everybody to try out whether they can use it for their own purposes.

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

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. 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. 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 the example of Figure 1, $succ(2,3) = 5$, and accordingly, $pred(2,5) = 3$.

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 $1,\ldots,n$. Thus, $succ(0,i) = i+1$ and $succ(0,n) = 1$.

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 imposes a tree structure on the PSLAs, see Figure 2. 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 3. (We temporarily relax the requirement that the extra pseudoline has to be x-monotone. Following Knuth [10, Section 9, p. 38], such a line is also called a *cutpath* [6].) This corresponds to a source-to-target path in the dual graph of the PSLA. Orienting the dual edges in the way how 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 4. 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

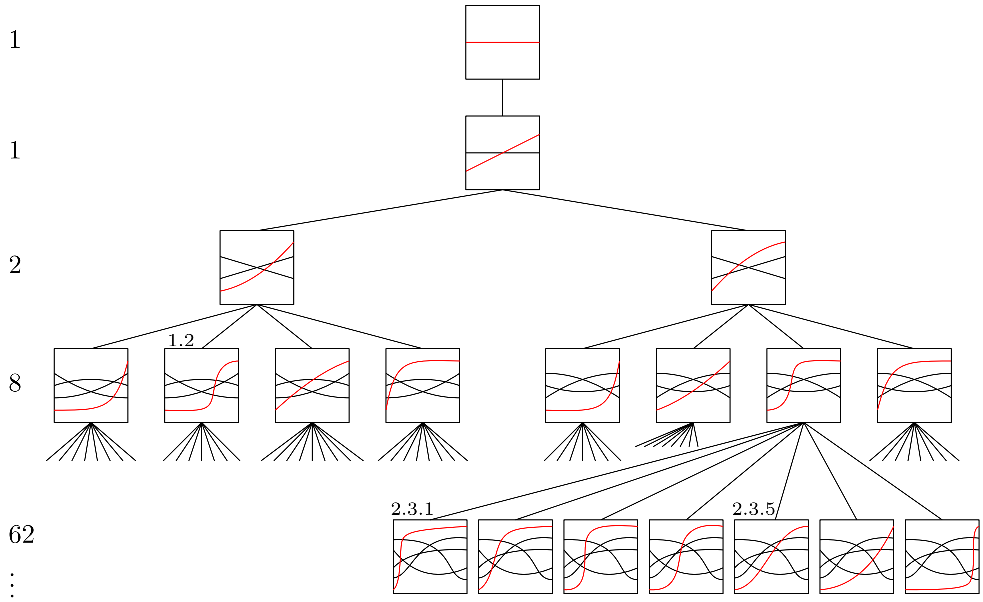


Figure 2 The first three three levels of the enumeration tree and a few nodes of the fourth level. The last inserted pseudoline is highlighted in color. For some nodes, the Dewey decimal notation is indicated.

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 2.

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

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 5 shows the confusing network of relationships. At the lower left corner, we find our favorite objects, the (x-monotone) PSLAs. The pseudolines are

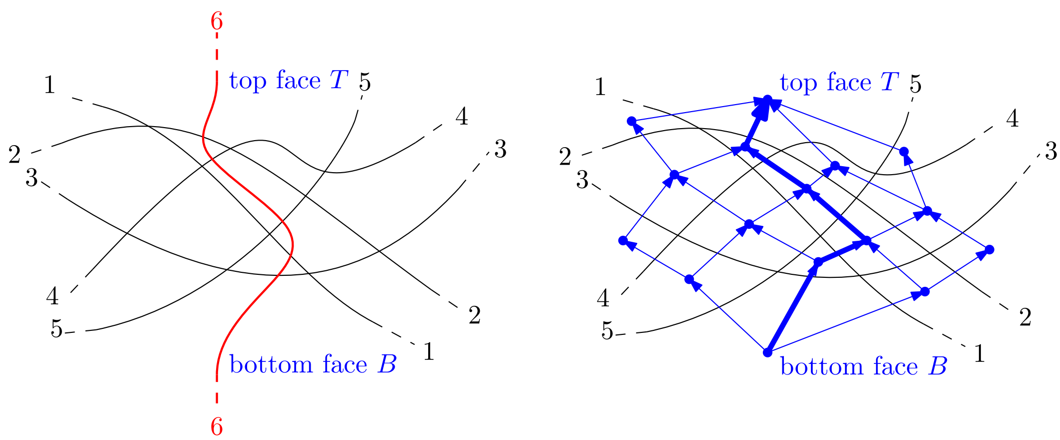


Figure 3 Left: Threading line 6 through a PSLA of 5 lines. Right: The dual DAG of this PSLA

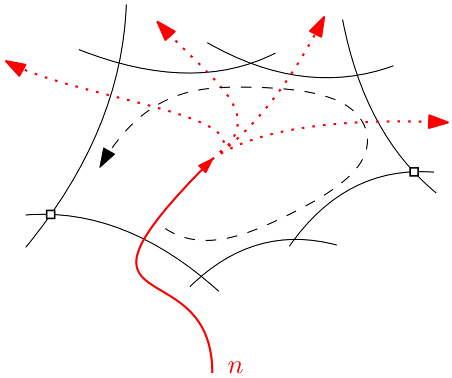


Figure 4 Continuing line n after entering a face.

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 in Figure 6. 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 negative infinity, around which the points are sorted.

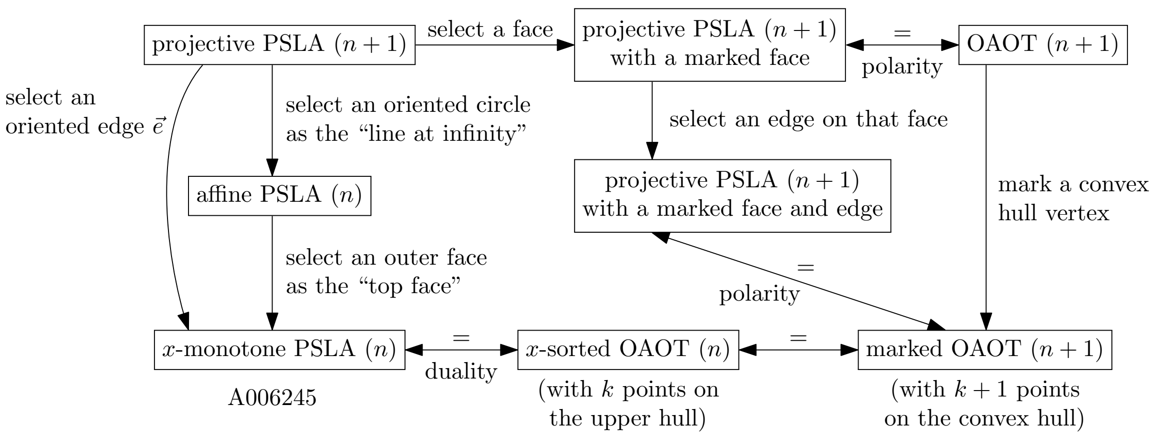


Figure 5 Relation between different concepts. An arrow in one direction indicates a specialization.

We can also move this extra point to a finite distance, sufficiently far below, without changing the order type. Moreover, if we move the point 0 to the left of all points, as indicated in Figure 6, 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 $1, \ldots, n$. The corresponding point 0 has a smaller x-coordinate than all other points, and the 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 $n + 1$ point where some *pivot point* on the convex hull is marked. By a projective transformation, the pivot point can be moved far down without changing the order type.

Thus we have explained the three boxes in the bottom row of Figure 5. The boxes refer to *oriented* abstract order types (OAOTs), because at this point, we still distinguish a point set from its reflection.

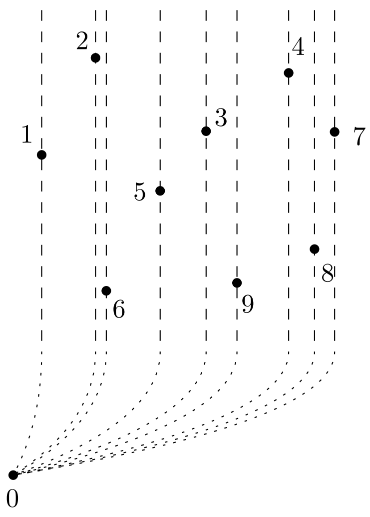


Figure 6 Modeling the x-sorted order of a point set by an extra point 0.

We are, however, interested in point sets without a marked pivot point. Therefore we must understand 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 7 shows a picture, where image of the PSLA to which we are used appears on the "front half" of the sphere in the left part of the picture and 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 indicate where the front part and the back part are stitched together.

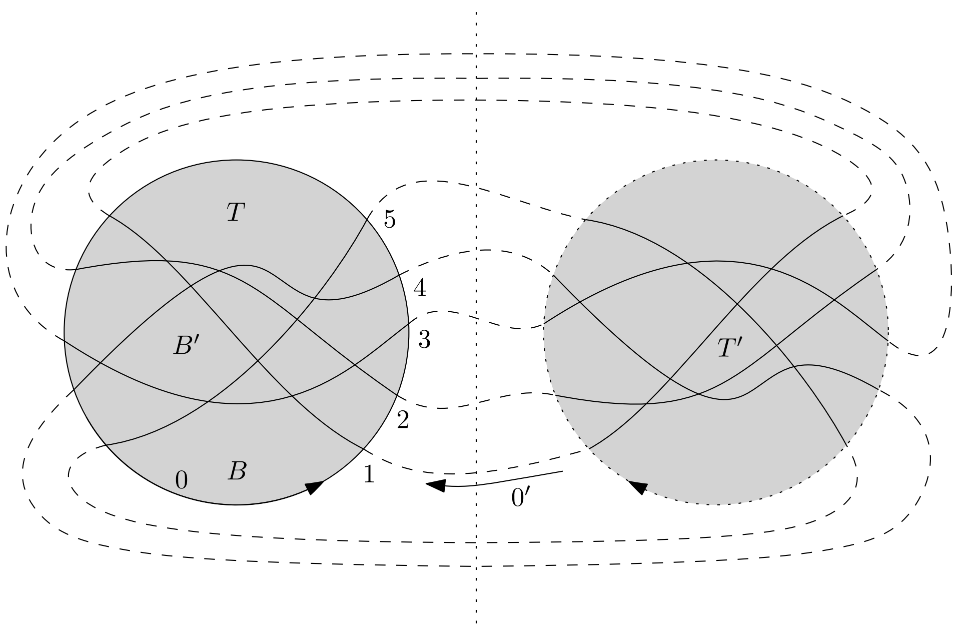


Figure 7 The spherical model of a projective PSLA, for the PSLA of Figure 3

Now, on this spherical model, we have $n+1$ lines. They are all are 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.

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 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 7. 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 5. 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. The boxes in Figure 5 refer to *oriented* abstract order types (OAOTs), because at this point, we still distinguish a point set from its reflection. Figure 5 includes some intermediate boxes, in which some data are partially fixed, and their translation between the pseudoline world and the point 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 $pred[j, k] = i'$.

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 8. 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 edges of the top face.

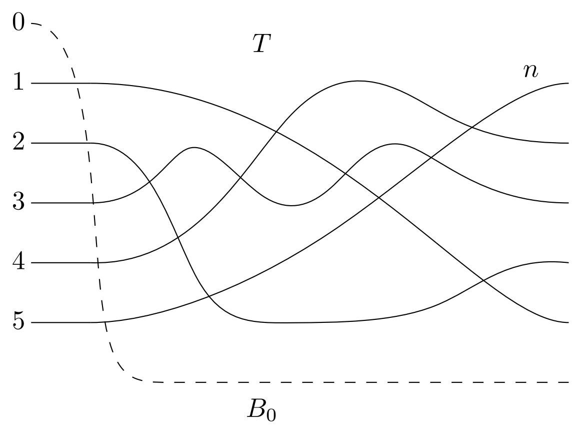


Figure 8 The convex hull in a pseudoline arrangement

4 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 9. 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 above 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 duality to a proper line arrangement. 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 if 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 provided by preprocessing.

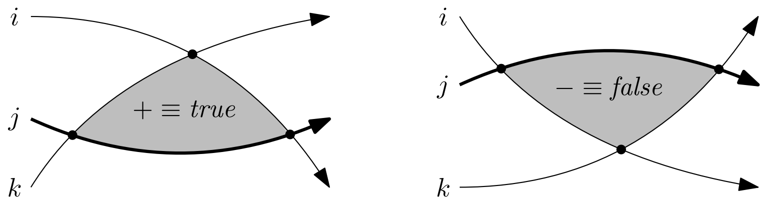


Figure 9 The orientation of three lines

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

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 by convention 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 determined. 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. The diagonal entries are irrelevant. From the *succ* links, it is straightforward to build the local sequences and the reverse 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.

5 Elimination of duplicates

An abstract order type with h hull vertices corresponds to $2h$ PSLAs: For each of the h hull edges, one has two choices of orientation. (If there are symmetries, some of these $2h$ PSLAs will coincide.) The standard approach to tackle this problem to compute some sort of canonical representation. In our program, we compare the local sequences matrices P lexicographically. The algorithm produces 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 look at the current local sequences matrix P^1 and its competitors $P^2, \dots, P^{2h}$. 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 (which must be a divisor of h). A mirror symmetry can occur alone or in combination with a rotational symmetry, and it will also be detected.

There are several considerations, that need to be taken into account 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 [7]. 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 statements holds both for AOTs [7, Theorem 10.2] and for (realizable) *order types* [7, Theorem 1.2]. In the latter setting of order types, convergence to 4 carries over to the unlabeled case [7, 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. Thus we can assume that only one out of 8 PSLAs is the lex-min PSLA, and 7 out of 8 are generated in vain. One can check this with the figures of Table 1. 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 runtime is spent in the lex-min test at the leaves of the tree.

In practice, if would be wasteful to compute the complete matrices $P^1, \dots, P^{2h}$ 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 [3, 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 small minority.

5.1 Screening

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.) 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 $P^2, \dots, P^{2h}$, we compute a renaming table for each matrix. This takes $O(n)$ time per matrix, by simply following the pointers along a 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 computer 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 PSLA 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 less than 6 entries in total before it could make a decision. This total is over all matrices $P^1, \dots, P^{2h}$ taken together. (This does not include the $2h$ entries P^i_{1n} that were compared in the screening test. The screening tests 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.)

5.1.1 More aggressing 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.

6 Parallelization

We implemented a trivial way to parallelize the enumeration. We 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 $k = 1, \dots, m$, the work is split into m roughly equal parts.

7 Enumerating only the realizable AOTs

We implemented a provision to enumerate only the (realizable) order types of points 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 8,699,559 entries and needs 184.6 MBytes. (With some technical effort, like eliminating common prefixes or a compressed binary format, one could reduce this space requirement significantly.)

8 Some 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 [4], which took about 3200 CPU days of computing time. The number h of hull vertices and the symmetry is 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 take some more statistics: about the number of halving-lines and the crossing number.

These data can be computed from the wiring-diagram: The number of crossings at level k in the wiring-diagram is the number of lines through pairs of points that have k points *below* them, and hence it is clear that these number are related to the 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 [12], 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 to make a statistical analysis. (The result file with these raw data is available in the repository.[4])

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 of [3], where the same data is given for (realizable) order types up to $n = 11$.

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 frequency of, say, triangular convex hulls. This should be related to the expected number of triangular faces in a “random” PSLA.

[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.)

[4] `crossing+halving-results-13.txt`

	$n=7$	$n=8$	$n=9$	$n=10$	$n=11$	$n=12$	$n=13$
$h=3$	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 value of formula (2), which has been discussed in Section 5.

	$h=3$	$h=4$	$h=5$	$h=6$	$h=7$	$h=8$
$n=10$	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

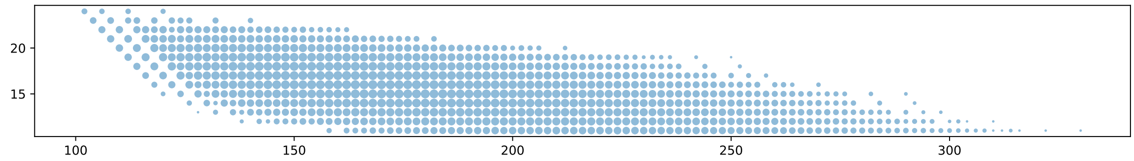
Table 3 Relative frequencies of convex hull sizes h

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 an 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

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	1354		253	10 549 648 258		454	119	463	21	474	1
157	4066	⋮	254	9 551 226 473	⋮	455	33	464	1	477	5
158	6966		255	9 720 622 387		456	46	465	10	479	1
159	13904		256	10 543 935 293		457	1	467	2	486	1
160	42950		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 10** Scatter-plot of crossing number (horizontal axis) versus number of halving-lines (vertical axis) for AOTs with $n = 11$ points. The size of the dots represents the frequency, on a logarithmic scale. 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.

vicinity of this value.

Figure 10 shows the joint distribution of both parameters, the crossing number and the number of halving-lines.

n	[A006247] (unoriented) AOTs	unsymmetric 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 Online Encyclopedia of Integer Sequences [13].

8.3 Symmetries

As mentioned in Section 5, we get 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

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						24			
10	13.058*	27	11		4				1	1				234	92*		3*	
11	76.186				1						1						12	

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 few differences to AOTs are highlighted. The majority of the difference set (column Δ in Table 1) belongs to class with no symmetries at all.

by their group of symmetries. 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 these 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, the symmetry group of an AOT must be isomorphic to a subgroup 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 $(n-1)$-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 C_6 or D_6 or 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. [7, 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 not allowed 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 unsymmetric AOTs in the second columns 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 is still manageable. Together with the number of PSLAs, which is known up to 16 lines, we can then calculate the number of non-symmetric AOTs, and hence the total number of AOTs.

8.3.1 Symmetries of affine PSLAs

An affine PSLA may be "rotated" in n different ways by choosing a different pair of top and bottom faces, and in addition, it can be reflected. An affine PSLA with n pseudolines it thus mapped to $2n$ x-monotone PSLAs, which are not necessarily different.

Note that symmetry of a PSLA is completely different from the symmetry of the AOT that is associated with it.

We can also classify PSLAs by symmetry. ...

8.3.2 Symmetries of an x-monotone PSLA

W x-monotone

For *x-monotone* PSLAs, there are two natural mirror symmetries that one could consider: A mirror reflection with a *vertical* symmetry axis keeps the bottom the top faces fixed and exchanges the *pred* and *succ* pointers; Mirror reflection with a horizontal symmetry axis, which swaps the bottom face with the top face and renumbers lines $1, 2, \ldots, n$ to $n, n-1, \ldots, 1$.

For an odd number $n \geq 3$ of lines, a horizontal symmetry axis cannot exist. 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.

(Such a vertical symmetry corresponds to a mirror symmetry of the associated AOT, with the mirror going through the pivot point.)

Similarly, a PSLA with $n \geq 4$ lines cannot have *both* a vertical and a horizontal symmetry axis, because then it would allow a 180° rotation, and this is impossible: Lines 1 and n partition the plane into four sectors. The crossing of lines 2 and $n-1$ is in one of theses four sectors, and rotation moves it to the opposite sector.

These two symmetries are the natural symmetries for x-monotone PSLAs. They preserve many properties of PSLAs, for example, the number of cutpaths.

....

8.4 The number of cutpaths of a PSLA

For a given PSLA, an extra pseudoline that runs from the bottom face to the top face is called a *cutpath*. The number of cutpaths is equal to the number of children of the corresponding node in the enumeration tree.

One clearly sees that PSLAs with many cutpaths tend to have children with many cutpaths.

For PSLAs with 8 or 9 pseudolines, the diagram looks qualitatively the same.

8.5 Exploring a random branch of the enumeration tree

9 Extensions

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

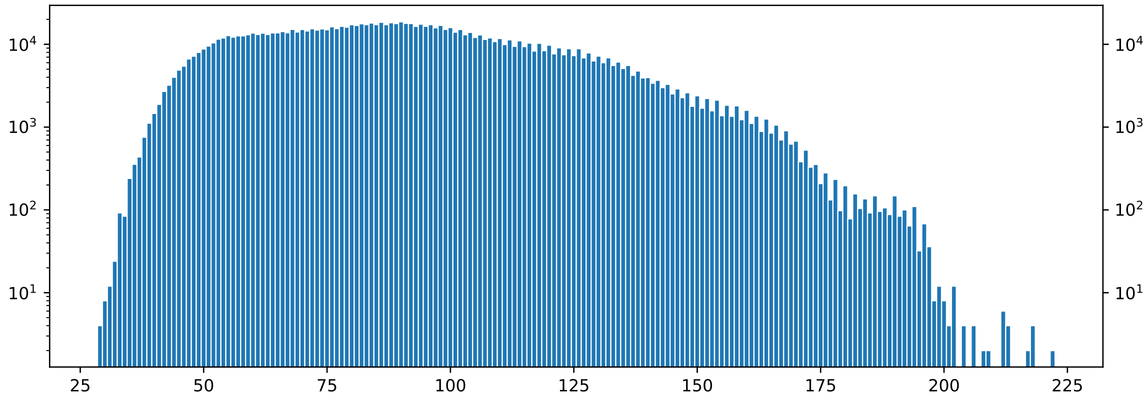


Figure 11 The distribution of the number of cutpaths of the 1,232,944 PSLAs with 8 pseudolines. The frequencies are on a logarithmic scale. For symmetry reasons, the frequency of each number of cutpaths is always even. The number of cutpaths ranges between 29 and 222, and the average is 90.85, which equals the quotient of the number of PSLAs with 9 and with 8 pseudolines, see Table 1.

1. We have concentrated on AOTs. PSLAs were used only as a tool to enumerate AOTs, but PSLAs could 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 5).
2. "Partial" pseudoline arrangements, in which lines are not forced to cross; see Figure 14 for an example.
3. Non-simple pseudoline arrangements, in which more than two pseudolines are allowed to cross in a point. These are much more numerous, and handling them would involve a redesign of the data structures from scratch.
4. Random generation. It is easy to generate a random PSLA by diving into the tree randomly. This will, however, not be a uniform selection.
5. Estimating the size of the deeper levels. Knuth [9] has observed that an unbiased estimate for the size of the tree can be obtained by iteratively proceeding to a random child and multiplying the encountered vertex degrees (see also [11, Sect. 7.2.2, pp. 46–51, Corollary E]). It would be interesting to investigate how much the degrees of the enumeration tree vary.
6. A side issue are nice drawings of pseudoline arrangements. The wiring diagram is simple to obtain but it is very jagged. Stretchability can be a very hard problem. Constructing a drawing in which the pseudolines don't "bend too much" would be an interesting challenge. (Maybe it would be an idea for a Geometric Optimization Challenge[5], perhaps in connection with the random generation method mentioned above.)

Acknowledgements. We thank the High-Performance-Computing Service of FUB-IT, Freie Universität Berlin [4] for computing time.

[5] `https://cgshop.ibr.cs.tu-bs.de/`

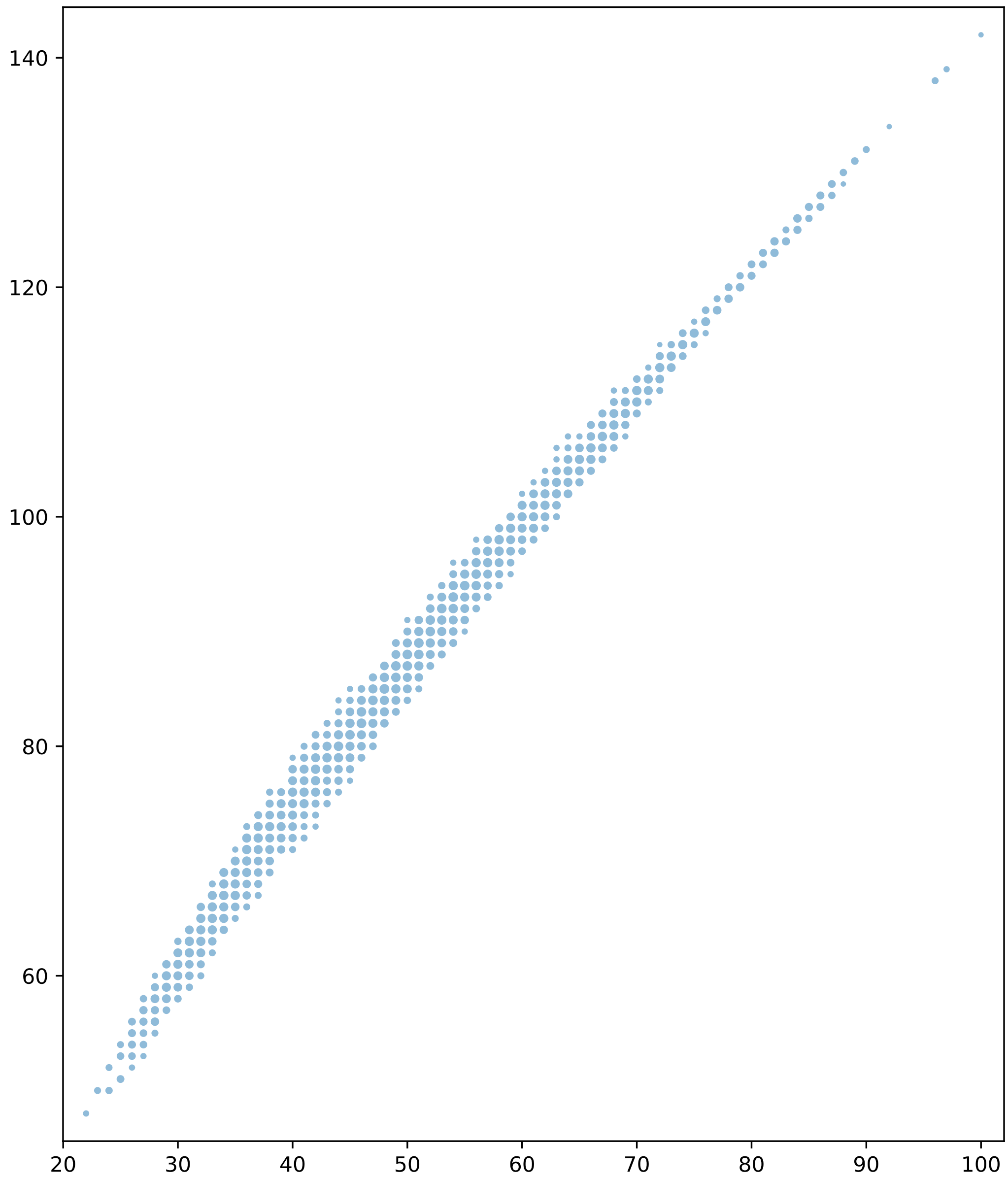


Figure 12 Scatter-plot of children versus grandchildren, for PSLAs with 7 pseudolines. The horizontal axis gives the number of cutpaths of each PSLA, or in other words, the degree of each node (the number of children) in the enumeration tree. The vertical axis gives the average degree of those children, or in other words, the number of grandchildren divided by the number of children, rounded to the nearest integer. The size of the dots represents the frequency, on a logarithmic scale. The number of cutpaths ranges between 22 and 100.

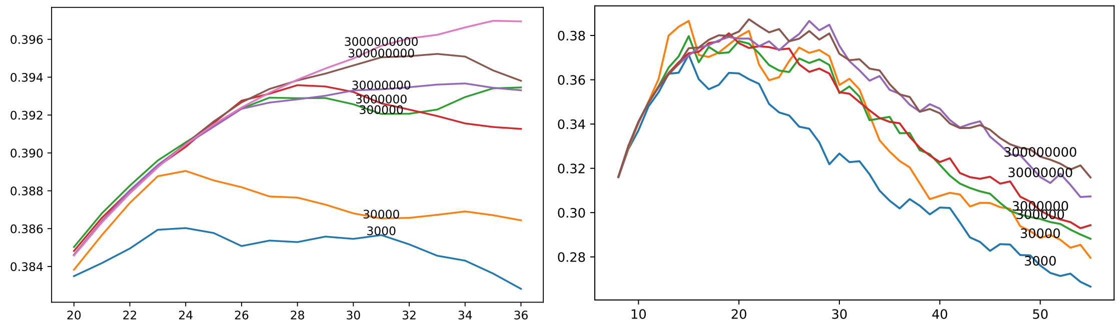


Figure 13 Estimating the constant $(\log_2 B_n)/n^2$.

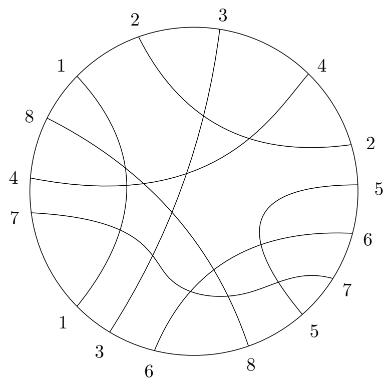


Figure 14 A partial PSLA

References

1 O. Aichholzer, F. Aurenhammer, and H. Krasser. Enumerating order types for small point sets with applications. In *Proc. 17^{th} Ann. ACM Symp. Computational Geometry*, pages 11–18, Medford, Massachusetts, USA, 2001.

2 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:https://doi.org/10.1016/j.jcta.2020.105236`.

3 Oswin Aichholzer and Hannes Krasser. Abstract order type extension and new results on the rectilinear crossing number. *Computational Geometry*, 36(1):2–15, 2007. Special Issue on the 21st European Workshop on Computational Geometry. `doi:10.1016/j.comgeo.2005.07.005`.

4 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`.

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

6 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`.

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

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

9 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/`.

10 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`.

11 Donald E. Knuth. *Combinatorial Algorithms, Part 2*, volume 4B of *The Art of Computer Programming*. Addison-Wesley, 2011.

12 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., Providence, RI, 2004. `doi:10.1090/conm/342/06138`.

13 The One-Line Encyclopedia of Integer Sequences. URL: `http://oeis.org/`.

A Benchmark comparison to 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 machine.

For task A, typically about 60 % of the total time were “system time”, for reading the file, and 40 % were “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 15 is a refined version of Figure 4, illustrating the meaning of the variables in the inner loop.

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

```
python3 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 `wiring_diagram.py`, one can for example modify the program to print wiring diagrams of all arrangements.

```python
"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 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])

# Start the generation proper:
PRED = {}; SUCC = {}
LINK(1, 0,0);
LINK(0, 1,1);

countPSLA = [0]*(n_max+1)
localCountPSLA = [0]*(n_max+1)
recursive_generate_PSLA_start(2)
# IPE_end() # finish and close ipe-file, in case it was used.
print ("Number of PSLAs:", *countPSLA[2:])
```

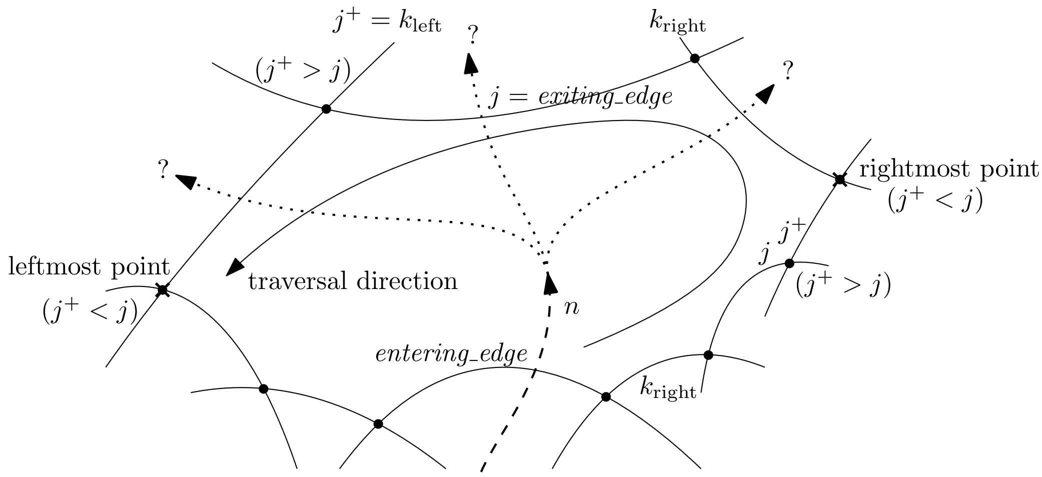


Figure 15 Threading line n through a face