Graz University of Technology
Institute for Software Technology
Bachelor Thesis Software Development
Course Number: 705.408

On Order Types, Projective Classes, and Realizations

Bachelor programme Technical Mathematics

Manfred Scheucher

student ID: 0930956
program code: 033 201

Supervisor: Assoc.Prof. Dipl.-Ing. Dr.techn. Oswin Aichholzer
Dipl.-Ing. Dr.techn. Thomas Hackl
Institute for Software Technology
Graz University of Technology

6. October 2014

Contents

1 Abstract **2**

2 Introduction **3**

3 Theoretical Part **4**
- 3.1 Pointsets . . . 4
- 3.2 Λ-Matrices . . . 5
- 3.3 Order Types . . . 6
- 3.4 Projective Classes . . . 7
- 3.5 λ-Matrices . . . 9
- 3.6 Realizations . . . 11
  - 3.6.1 Sub Order Types . . . 11
  - 3.6.2 The Rotation Functions . . . 12
- 3.7 Approaches for Realization . . . 14
  - 3.7.1 QCP-Realization . . . 14
  - 3.7.2 Grassmann-Plücker Heuristic . . . 16
  - 3.7.3 Back-Rotation-Realization . . . 18
- 3.8 Research Topics . . . 22
  - 3.8.1 k-Gons and k-Holes . . . 22
  - 3.8.2 Rectilinear Crossings . . . 22
  - 3.8.3 Universal Pointset . . . 23

4 pyotlib **24**
- 4.1 Programming Language and Modules . . . 24
- 4.2 Supported Filetypes . . . 24
- 4.3 Provided Functionality . . . 24
  - 4.3.1 Core Classes . . . 24
  - 4.3.2 Order Types and Projective Classes . . . 25
  - 4.3.3 Realization . . . 25
  - 4.3.4 Further Classes . . . 27
  - 4.3.5 Provided Scripts . . . 27
- 4.4 Some Benchmarks . . . 29
- 4.5 Realization of small Order Types . . . 29
- 4.6 Realization of large Order Types . . . 30

5 Overview of Results **32**
- 5.1 Results on Empty Convex 5-Holes: . . . 32
- 5.2 Results on Rectilinear Crossing Numbers and on Realizability of Large Order Types: . . . 32
- 5.3 Results on Realizations of Minimal Size: . . . 34
- 5.4 Results on Universal Order Types: . . . 35

1 Abstract

In this thesis we describe some basic methods to efficiently work with order types and projective classes and some realization techniques. In addition, we give a short documentation of the framework `pyotlib` which provides implementations of the mentioned methods. Furthermore we give an overview of the results obtained by using the provided methods.

2 Introduction

After finishing some calculations for Manfred Scheucher's Bachelor's thesis in Computer Science [36] there were exactly 168 abstract order types with 13 points that contained exactly 3 convex 5-holes. Most of them were easy to realize and some of them could be proven to be non-realizable. But there were also some order types that could not be realized for months. In general, the realization problem is known to be NP-hard. Using the implementation developed for this thesis, which can also make use of the power of Wolfram Mathematica [13], 156 order types could be realized and the 12 remaining abstract order types could be proven to be not realizable with just about 30 minutes computation time.

As there exist lots of small scripts from the preceding Bachelor's thesis and because there are lots of calculations that can be done with order types and projective classes, we decided to establish a whole framework called `pyotlib` to provide the core functionality. Thus, we analyzed and implemented all important algorithms from scratch to provide an efficient implementation.

After the BSc thesis [36] there were also some abstract order types with a larger number of points and a low number of convex 5-holes known. By realizing them, the upper bound on the minimal number of convex 5-holes $h_5(n)$ could be slightly improved for some n. For example, by realizing an order type with 19 points and 27 convex 5-holes it is proven that $h_5(19) \leq 27$.

3 Theoretical Part

There are infinite possibilites to pick three distinct (labeled) points in the Euclidean plane, as the number of points in the Euclidean plane is infinite. However, there are only three cases that can occur:

1. the third point lies on the directed line through point one and point two, i.e., these three points are "collinear";
2. the third point lies on the left side of the directed line through point one and point two, i.e, these three points are "left oriented";
3. the third point lies on the right side of the directed line through point one and point two, i.e, these three points are "right oriented".

Analogously, for any set of points with more than three points each point triple has a certain orientation. Hence, it is a good approach to group sets of points by their orientations. This concept was first introduced by Goodman and Pollack [28] who denoted these groups as order types. Consecutively, given an order type one might ask for a realization, i.e, a set of points with the desired orientations. One approach to enumerate all order types was proposed by Aichholzer, Aurenhammer and Krasser [15, 31]. Prior work has also been done by Bokowski, Laffaille and Richter-Gebert [21, 20], and by Finschi and Fukuda [26, 27].

In this section we give an introduction to the theory of order types as it is necessary to know the definitions and ideas to understand the methods that we use later for realization.

3.1 Pointsets

Definition 1. *For $n \in \mathbb{N}$ we define the* ***symmetric group***

$$S_n := \{\pi \colon \{1, \dots, n\} \to \{1, \dots, n\} \mid \pi \text{ bijective}\}$$

as the set of all permutations of length n. Given $\pi \in S_n$ we denote $\pi_i := \pi(i)$.

Definition 2. *We denote the tuple $P = (p_1, \dots, p_n)$ with pairwise different $p_i = (x_i, y_i) \in \mathbb{R}^2$ as (real)* ***pointset****.*
If $p_i \in \mathbb{Q}^2$ holds for all i we denote P as ***rational pointset****.*
If $p_i \in \mathbb{Z}^2$ holds for all i we denote P as ***integer pointset****.*
If $p_i \in \mathbb{N}_0^2$ holds for all i we denote P as ***natural pointset****.*
We define the ***set of real, rational, integer, and natural pointsets*** *as*

$$\mathcal{P}_n := \{(p_1, \dots, p_n) \mid p_i \in \mathbb{R}^2 \text{ pairwise different}\},$$
$$\mathcal{P}_n^Q := \{(p_1, \dots, p_n) \mid p_i \in \mathbb{Q}^2 \text{ pairwise different}\} = \mathcal{P}_n \cap (\mathbb{Q}^2)^n,$$
$$\mathcal{P}_n^I := \{(p_1, \dots, p_n) \mid p_i \in \mathbb{Z}^2 \text{ pairwise different}\} = \mathcal{P}_n \cap (\mathbb{Z}^2)^n, \text{ and}$$
$$\mathcal{P}_n^N := \{(p_1, \dots, p_n) \mid p_i \in \mathbb{N}_0^2 \text{ pairwise different}\} = \mathcal{P}_n \cap (\mathbb{N}_0^2)^n, \text{ respectively.}$$

Remark that in natural pointsets point coordinates can also be zero. We could also denote $\mathcal{P}_n^N$ as "non-negative integer pointset".

Definition 3. *Let $P = (p_1, \ldots, p_n) \in \mathcal{P}_n^I$ be an integer pointset. We define the* ***diameter*** *of the pointset P as*

$$\operatorname{diam}(P) := \max\left\{ \max_{i\in\{1,\ldots,n\}} |x_i|, \max_{i\in\{1,\ldots,n\}} |y_i| \right\}.$$

For $P \in \mathcal{P}_n \backslash \mathcal{P}_n^I$ we can set $\operatorname{diam}(P) := \infty$.

Definition 4. *Let $P = (p_1, \ldots, p_n) \in \mathcal{P}_n$ be a pointset. Three points p_i, p_j, p_k for i, j, k pairwise different are said to be* ***collinear*** *if and only if*

$$p_k = \lambda p_i + (1 - \lambda) p_j$$

holds for some $\lambda \in \mathbb{R}$. The points $p_1, \ldots, p_n$ are said to be in ***general position*** *and the pointset P is said to be* ***general*** *if and only if there are no collinear points.*

3.2 Λ-Matrices

As **line arrangements** and **pseudoline arrangements** are used only for the definition of Λ-matrices we refer to [31, 15] for the defintion and construction methods.

Definition 5. *Let $\Lambda = (\Lambda_{ijk})_{i,j,k\in\{1,\ldots,n\}} \in \{-1, 0, 1\}^{n\times n\times n}$ be a tensor of orientations such that Λ can be constructed via pseudoline arrangements.*

We will denote Λ as ***Λ-matrix*** *("big lambda matrix") of size n and define the* ***set of Λ-matrices*** *of size n as*

$$\Lambda_n^* := \{\Lambda \in \{-1, 0, 1\}^{n\times n\times n} \mid \Lambda \text{ is a } \Lambda\text{-matrix}\}.$$

Note that by defintion for $i, j, k \in \{1, \ldots, n\}$ the following three conditions must hold:

1. $\Lambda_{ijk} = \Lambda_{jki}$, i.e, relabeling three points (counter) clockwise does not change the orientation of this point triple;
2. $\Lambda_{ijk} = -\Lambda_{jik}$, i.e., switching two points of inverts the orientation;
3. $\Lambda_{ijk} \neq 0$ holds for i, j, k pairwise different, i.e., no collinear points.

Remark that these conditions are necessary but not sufficient! [31]

Note 1. *Note that we could also use a less restrictive definition for Λ-matrices such that collinear index triples are allowed where an index triple (i, j, k), i, j, k pairwise different, is said to be collinear if and only if $\Lambda_{ijk} = 0$. Otherwise it is said to be general. If there is a collinear index triple Λ is said to be collinear. Otherwise Λ is said to be general. But for our purposes we only consider so called "general" Λ-matrices. [26, 27]*

Definition 6. *Let $P \in \mathcal{P}_n$ be a general pointset. We denote*

$$\Lambda(P) := (\Lambda_{ijk})_{i,j,k\in\{1,\ldots,n\}}$$

as the ***induced*** Λ***-matrix*** *by the pointset* P *with the* ***induced orientation triples***

$$\Lambda_{ijk} := \operatorname{sgn} \det \begin{pmatrix} 1 & 1 & 1 \\ x_i & x_j & x_k \\ y_i & y_j & y_k \end{pmatrix}$$

where det *is the determinant function and* sgn *is the sign function.*

Proposition 1. *Any general pointset is the dual of a line arrangement which is per definition also a pseudo line arrangement [31]. Hence any induced* Λ*-matrix is a* Λ*-matrix by definition.*

Definition 7. *Let* $\Lambda \in \Lambda_n^*$*. For any* $i, j \in \{1, \dots, n\}$ *we define the* ***left set*** $L_{ij}(\Lambda)$ *and the* ***right set*** $R_{ij}(\Lambda)$ *as*

$$L_{ij}(\Lambda) := \{k \mid \Lambda_{ijk} > 0\} \text{ and}$$
$$R_{ij}(\Lambda) := \{k \mid \Lambda_{ijk} < 0\}, \text{ respectively.}$$

Remark that this definition of $L_{ij}(\Lambda)$ was proposed by Goodman and Pollack [28] as defintion of (induced) Λ-matrices.

3.3 Order Types

Motivation: There is a large number of Λ-matrices even for small n which store the same information so one might want to group them.

Definition 8. *Let* n *be fixed. We define the* ***lexicographical ordering*** $\preceq$ *on* $\mathbb{R}^n$ *as*

$$a \preceq b :\Leftrightarrow a = b \vee \exists k \in \{1, \dots, n\} \colon (a_k < b_k \wedge \forall i \in \{1, \dots, k-1\} \colon a_i = b_i)$$

for any $a, b \in \mathbb{R}^n$*.*

Proposition 2. $\preceq$ *defines a total order on* $\mathbb{R}^n$*.*

Definition 9. *Let* n *be fixed and let* $\pi \in S_n$ *be a permutation. We define*

$$\begin{aligned} \Pi_\pi \colon \Lambda_n^* &\to \Lambda_n^* \\ (\Lambda_{i,j,k})_{i,j,k \in \{1,\dots,n\}} &\mapsto (\Lambda_{\pi_i, \pi_j, \pi_k})_{i,j,k \in \{1,\dots,n\}}. \end{aligned}$$

For $\Lambda \in \Lambda_n^*$ *we denote* $\Pi_\pi(\Lambda)$ *as the* ***permutation of*** Λ ***induced by*** π*.*

Since this Π_π is a bijection on Λ^* we can define an equivalence-relation $\overset{\text{OT}}{\sim}$ on Λ^*:

Definition 10. *We define*

$$\Lambda \overset{\text{OT}}{\sim} M :\Leftrightarrow \exists \pi \in S_n \colon \Pi_\pi(\Lambda) = M \vee \Pi_\pi(\Lambda) = -M,$$

where $-M$ *denotes the mirrored* Λ*-matrix, i.e., the* Λ*-matrix where every single orientation is inverted. We denote the equivalence class*

$$[\Lambda]_{\overset{\text{OT}}{\sim}} =: [\Lambda] =: \text{OT}(\Lambda)$$

as the ***order type of*** Λ *and* Λ *is said to be a* ***representative*** *of the order type* $[\Lambda]$*. We define the* ***set of order types*** *as the set of all equivalence classes as*

$$\text{OT}_n^* := \left(\Lambda_n^* \Big/ \overset{\text{OT}}{\sim} \right).$$

Definition 11. *For an order type $O \in \mathrm{OT}_n^*$ we denote the unique* ***lexicographically minimal representative*** *(or "lexicographical minimum") as $\Lambda_{\min}(O)$ or $\Lambda_{\min}$ if O is given by the context, i.e.,*

$$\Lambda_{\min}(O) := \operatorname{lexmin} O.$$

For $\Lambda \in \Lambda_n^*$ we also define

$$\Lambda_{\min}(\Lambda) := \Lambda_{\min}([\Lambda]).$$

Note that, since $\preceq$ is a total order, the lexicographically minimal representative $\Lambda_{\min}$ is well defined.

3.4 Projective Classes

Since there are lots of order types that contain similar information one might want to group them.

Definition 12. *For $\Lambda \in \Lambda_n^*$ we define the* ***set of extremal points*** *as*

$$\partial\Lambda := \{i \in \{1,\dots,n\} \mid \exists j \in \{1,\dots,n\}\backslash\{i\} \colon \forall k \in \{1,\dots,n\} \colon \Lambda_{ijk} \neq 1\}.$$

We can also write $\partial\Lambda = \{e_i\}_{i\in\{1,\dots,k\}} \subseteq \{1,\dots,n\}$ for some $k \in \{1,\dots,n\}$ with $\Lambda_{e_i,e_{i+1},f} \neq 1$ for $i \in \{1,\dots,k-1\}$ and $\Lambda_{e_k,e_1,f} \neq 1$ for every $f \in \{1,\dots,n\}$.

Proposition 3. *Let $\Lambda \in \Lambda_n^*$. There exists a unique j for every $i \in \partial\Lambda$ since there are no collinear indices in the extremal set by our definition of Λ-matrices.*

Definition 13. *Let n be fixed. For $i_0 \in \{1,\dots,n\}$ we define the* ***flipping functions***

$$\begin{aligned} f_{i_0} \colon \Lambda^*_{n;i_0} &\to \Lambda^*_{n;i_0} \\ \Lambda &\mapsto \widetilde{\Lambda} \end{aligned}$$

where $\Lambda^*_{n;i_0}$ is the set of Λ-matrices where i_0 is extremal, i.e.,

$$\Lambda^*_{n;i_0} := \{\Lambda \in \Lambda_n^* \mid i_0 \in \partial\Lambda\}$$

and

$$\widetilde{\Lambda}_{ijk} := \begin{cases} -\Lambda_{ijk} & \text{if } i_0 \in \{i,j,k\} \\ \Lambda_{ijk} & \text{otherwise.} \end{cases}$$

Note that the flipping function f_{i_0} inverts all orientation triples containing index i_0 but does not change any other orientations. An extremal index i_0 stays extremal if f_{i_0} is applied.

Proposition 4. *The flipping function f_{i_0} fulfills the following properties:*

- $f_{i_0} \circ f_{i_0} = \mathrm{id}$ *where* id *is the identity function;*
- $f_{i_0} = f_{i_0}^{-1}$*;*
- f_{i_0} *is bijective.*

Definition 14. *Let n be fixed. We define an equivalence relation $\overset{\mathrm{PC}}{\sim}$ on Λ_n^* with $\Lambda \overset{\mathrm{PC}}{\sim} M$ if and only if there exist*

- $k \in \mathbb{N}$,
- $\Lambda^{(1)}, \dots, \Lambda^{(k)} \in \Lambda_n^*$, *and*
- $i_1, \dots, i_{k-1} \in \{1, \dots, n\}$

such that

- $\Lambda^{(1)} \in [\Lambda]$,
- $\Lambda^{(k)} \in [M]$, *and*
- $i_j \in \partial(\Lambda^{(j)})$ *and* $f_{i_j}(\Lambda^{(j)}) = \Lambda^{(j+1)}$ *holds for all* $j \in \{1, \dots, k-1\}$.

*We denote the equivalence class $[\Lambda]_{\overset{\mathrm{PC}}{\sim}} =: \langle\Lambda\rangle =: \mathrm{PC}(\Lambda)$ as the **projective class of** Λ. We further define the **set of projective classes** as the set of all equivalence classes, i.e.,*

$$\mathrm{PC}_n^* := \left(\Lambda_n^* \Big/ {}_{\overset{\mathrm{PC}}{\sim}}\right).$$

Remark that Λ and M are in the same projective class if and only if there is a flipping sequence of extremal indices from $\widetilde{\Lambda} \in [\Lambda]$ to $\widetilde{M} \in [M]$.

Proposition 5. *We can write the projective class as the union of all order types spanned by elements in $\langle\Lambda\rangle$, i.e.,*

$$\langle\Lambda\rangle = \bigcup_{M \in \langle\Lambda\rangle} [M].$$

Further, there exists $k \in \mathbb{N}$ and $\Lambda^{(1)}, \dots, \Lambda^{(k)} \in \langle\Lambda\rangle$ such that

$$\langle\Lambda\rangle = \biguplus_{i \in \{1,\dots,k\}} [\Lambda^{(i)}],$$

where $\biguplus$ denotes the disjoint union operation.

Definition 15. *We can also define an equivalence relation $\overset{\mathrm{PC}}{\sim}$ on OT_n^* as*

$$O_1 \overset{\mathrm{PC}}{\sim} O_2 :\Leftrightarrow \exists \Lambda \in O_1, M \in O_2 \colon \Lambda \overset{\mathrm{PC}}{\sim} M$$

*for any $O_1, O_2 \in \mathrm{OT}_n^*$. We denote the equivalence class $[O]_{\overset{\mathrm{PC}}{\sim}} =: \langle O\rangle$ as **projective class of an order type**. Furthermore, we can split the set of order types into equivalence classes which we will also denote as **projective classes of the order types** and denote the set as PC_n^{**}, i.e.,*

$$\mathrm{PC}_n^{**} := \left(\mathrm{OT}_n^* \Big/ {}_{\overset{\mathrm{PC}}{\sim}}\right) = \left(\left(\Lambda_n^* \Big/ {}_{\overset{\mathrm{OT}}{\sim}}\right) \Big/ {}_{\overset{\mathrm{PC}}{\sim}}\right).$$

Remark that k in the previous proposition is equal to $|\langle[\Lambda]\rangle|$.

Corollary 1. *According to Preposition 5 for every $P \in \mathrm{PC}_n^{**}$ there is exactly one $Q \in \mathrm{PC}_n^*$ (and vice versa) such that*

$$\biguplus_{O \in P} O = Q.$$

Thus, we can write

$$P = \left(Q \Big/ \underset{\sim}{\mathrm{OT}} \right).$$

According to this Corollary any projective class (a set of Λ-matrices) can be partitioned into a disjoint set of order types.

Definition 16. *For a projective class $P \in \mathrm{PC}_n^*$ we denote the **lexicographically minimal representative** (or simply "lexicographical minimum") as $\Lambda_{\min}^{\mathrm{PC}}(P)$ or simply as $\Lambda_{\min}^{\mathrm{PC}}$ if P is given by the context, i.e.,*

$$\Lambda_{\min}^{\mathrm{PC}}(P) := \operatorname{lexmin} P.$$

For $\Lambda \in \Lambda_n^*$ we also define

$$\Lambda_{\min}^{\mathrm{PC}}(\Lambda) := \Lambda_{\min}^{\mathrm{PC}}(\langle \Lambda \rangle).$$

Note that, since $\preceq$ is a total order, the lexicographically minimal representative $\Lambda_{\min}^{\mathrm{PC}}$ is well defined.

3.5 λ-Matrices

Because there are n^3 entries in a Λ-matrix one might want to find a structure to store the information more efficiently.

Definition 17. *We define $\lambda(\Lambda) := (\lambda_{ij})_{i,j \in \{1,\dots,n\}} \in \{0, \dots, n\}^{n \times n}$ with $\lambda_{ij} := |L_{ij}(\Lambda)|$ as the **(induced) λ-matrix** ("small lambda matrix") where $L_{ij}(\Lambda)$ denotes the left set of Λ (see Definition 7). We further define the **set of (induced) λ-matrices** of size n as*

$$\lambda_n^* := \{\lambda \in \{0, \dots, n\}^{n \times n} \mid \exists \Lambda \in \Lambda_n^* \colon \lambda(\Lambda) = \lambda\}.$$

Proposition 6. *Let $\lambda \in \lambda_n^*$. Then $\lambda_{ij} + \lambda_{ji} = n - 2$ holds for any $i \neq j$.*

Note 2. *Note that for a definition which allows collinear index triples another set for collinear indices $C_{ij}(\Lambda)$ would be necessary. Using such a definition the equation from the previous proposition would hold if and only if*

$$C_{ij}(\Lambda) := \{k \mid \Lambda_{ijk} = 0\} = \emptyset.$$

We will show that a Λ-matrix and its induced λ-matrix store the same information such that we can use "Λ-matrix" and "λ-matrix" as synonyms and just talk about "lambda matrices". We show that the mapping $\Lambda \mapsto \lambda(\Lambda)$ is injective and thus bijective by definition.

Lemma 1. *The mapping $\lambda \colon \Lambda_n^* \to \lambda_n^*; \Lambda \mapsto \lambda(\Lambda)$ is injective.*

Proof. To proof this lemma we give an algorithm to find Λ:

Let $i_0 \in \partial\Lambda$. Since there are no collinear indices there exists a unique permutation $\pi \in S_n$ such that $\pi_1 = i_0$ and $L_{i_0,\pi_j} = j-2$ for $j \geq 2$. According to ordering around the index i_0 we can see that $\Lambda_{\pi_1,\pi_j,\pi_k} = 1$ must hold for $1 < j < k$.

One can remove the index i_0 and continue iteratively until no more points are left. □

Note 3. *Note that using a definition which allows collinear index triples it can also be shown that given $\lambda \in \lambda_n^*$ there exists a unique $\Lambda \in \Lambda_n^*$ with $\lambda(\Lambda) = \lambda$. To proof this collinear version via an algorithm we might need to consider more than one extremal point.*

Corollary 2. *We denote the inverse function as $\lambda^{-1} =: \Lambda$ and have a mapping $\Lambda\colon \Lambda_n^* \to \lambda_n^*$ that gives the **induced Λ-matrix by** λ.*

We now identify

$$\Lambda(\lambda) = \Lambda \mathrel{\widehat{=}} \lambda = \lambda(\Lambda).$$

Since a λ-matrix has n^2 entries of size $\mathcal{O}(\log n)$, $\mathcal{O}(n^2 \log n)$ bits are needed for encoding. Hence, we already got an improvement for storing Λ-matrices.

Definition 18. *For $\lambda \in \lambda_n^*$ with $\lambda = \lambda(\Lambda)$ we can define the order type $[\lambda]$ and the projective classes $\langle\lambda\rangle$ as*

$$\lambda([\Lambda]) := \{\lambda(M) \mid M \in [\Lambda]\} \text{ and}$$
$$\lambda(\langle\Lambda\rangle) := \{\lambda(M) \mid M \in \langle\Lambda\rangle\}, \text{ respectively.}$$

Proposition 7. *$\Lambda([\lambda]) = \{\Lambda(\mu) \mid \mu \in [\lambda]\}$ and $\Lambda(\langle\lambda\rangle) = \{\Lambda(\mu) \mid \mu \in \langle\lambda\rangle\}$ hold by definition.*

According to this proposition we can identify $[\lambda] \mathrel{\widehat{=}} [\Lambda]$ and $\langle\lambda\rangle \mathrel{\widehat{=}} \langle\Lambda\rangle$.

Instead of calculating the lexicographical minimum of $[\Lambda]$ we can also use the lexicographical minimum of $[\lambda]$ as representative which is also unique according to the previous proposition. The same holds for the representative of $\langle\lambda\rangle$.

Definition 19. *For $\lambda \in \lambda_n^*$ we define the lexicographical minimum as*

$$\lambda_{\min}(\lambda) := \operatorname{lexmin}[\lambda] \text{ and}$$
$$\lambda_{\min}^{\mathrm{PC}}(\lambda) := \operatorname{lexmin}\langle\lambda\rangle.$$

Proposition 8. *Let $\lambda \in \lambda_n^*$ and $\lambda = \lambda(\Lambda)$. Then*

$$i \in \partial\Lambda \Leftrightarrow \exists j : \lambda_{ij} = 0.$$

Proposition 9. *Let $\lambda \in \lambda_n^*$. For the lexicographical minimum*

$$\lambda_{\min} = (\lambda_{ij})_{i,j\in\{1,\dots,n\}} \in [\lambda]$$

it is necessary (but not sufficient) that $\lambda_{1j} = j-2$ holds for $j \geq 2$.

According to the previous proposition, there at most $2n$ λ-matrices in $[\lambda]$ that fulfill this property since $|\partial\Lambda(\lambda)| \leq n$ (+mirrored).

Corollary 3. *$\lambda_{\min}$ can be calculated by comparing the $\mathcal{O}(n)$ candidates. Since there are n^2 integer entries in a λ-matrix of size $\mathcal{O}(n)$ ($\mathcal{O}(\log n)$ bits) the calculation of $\lambda_{\min}$ can be done in $\mathcal{O}(n^3 \log n)$ time.*

Theorem 1. *Let $\lambda \in \lambda_n^*$. The complete projective class $P := \langle\lambda\rangle$ and thus also the lexicographical minimum $\lambda_{\min}^{\mathrm{PC}}$ can be calulated in $\mathcal{O}(|P| \log |P| n^2 \log n)$ time using $\mathcal{O}(|P| n^2 \log n)$ space (output sensitive).*

Proof. Let $G = (V, E)$ be a undirected graph with $V = P$ and $\{u, v\} \in V$ if and only if $u \overset{\mathrm{PC}}{\sim} v$.

There are at most $|n|$ neighbors for each $v \in V$ because at most n flip-operations are possible. Thus, the calculation can be done by enumerating the whole graph recursively. There are $|V|$ nodes and every node is entered at most n times. All duplicate checks can be done with $\mathcal{O}(|V| log |V|)$ comparisons using $\mathcal{O}(|V|)$ space. Each operation on a node (e.g., storing, comparison) can be performed in $\mathcal{O}(n^2 \log n)$ time if λ-matrices are used. □

Instead of storing one Λ-matrix per order type we can also just store one Λ-matrix per projective class since all order types in a projective class can be enumerated according to this theorem.

3.6 Realizations

Definition 20. *Let $\Lambda \in \Lambda_n^*$. We define the set of **realizations of** Λ as*

$$R(\Lambda) := \{P \in \mathcal{P}_n \mid \Lambda(P) = \Lambda\}.$$

*We further define the **set of rational, integer, and natural realizations** as*

$$\begin{aligned} R^Q(\Lambda) &:= R(\Lambda) \cap \mathcal{P}_n^Q, \\ R^I(\Lambda) &:= R(\Lambda) \cap \mathcal{P}_n^I, \text{ and} \\ R^N(\Lambda) &:= R(\Lambda) \cap \mathcal{P}_n^N, \text{ respectively.} \end{aligned}$$

*Furthermore, Λ is said to be **realizable** if $R(\Lambda) \neq \emptyset$, otherwise it is said to be **non-realizable** or **pure abstract**.*

Lemma 2. *For $O \in \mathrm{OT}_n^*$ either all $\Lambda \in O$ are realizable or all $\Lambda \in O$ are pure abstract. An order type Λ is said to be **realizable** or **pure abstract**, respectively.*

Proof. Let $\{p_1, \ldots, p_n\} \in R(\Lambda)$ be a realization of Λ and $M = \Pi_\pi(\Lambda) \in [\Lambda]$. Then $\{p_{\pi_1}, \ldots, p_{\pi_n}\}$ is a realization of M. □

3.6.1 Sub Order Types

Definition 21. *Let $\Lambda = (\Lambda_{ijk})_{i,j,k \in \{1,\ldots,n\}} \in \Lambda_n^*$ and let $A = \{a_1, \ldots, a_m\} \subseteq \{1, \ldots, n\}$. By construction of Λ-matrices (via pseudoline arrangements), it holds that*

$$\Lambda' := (\Lambda_{a_i, a_j, a_k})_{i,j,k \in \{1,\ldots,m\}} \in \Lambda_m^*.$$

We denote

- Λ' *as the* ***sub*** Λ***-matrix induced by the index set*** A *of* Λ*;*
- $[\Lambda']$ *as a* ***sub order type*** *of* $[\Lambda]$*;*
- $\langle\Lambda'\rangle$ *as a* ***sub projective class*** *of* $\langle\Lambda\rangle$*.*

Note 4. *These definitions give ordering relations on* Λ_n^**, on* OT_n^**, and on* PC_n^**.*

Lemma 3. *Let* $P = (p_1, \ldots, p_n)$ *be a realization of* $\Lambda \in \Lambda_n^*$ *and let* $\Lambda' \in \Lambda_m^*$ *be the sub* Λ*-matrix induced by the set* $A = \{a_1, \ldots, a_m\} \subseteq \{1, \ldots, n\}$*. Then* $P' := (p_{a_1}, \ldots, p_{a_m})$ *is a realization of* Λ'*.*

Corollary 4. *Let* $\Lambda \in \Lambda_n^*$*. If there exists a non-realizable sub* Λ*-matrix of* Λ *then* Λ *is also non-realizable.*

According to this corollary we can search for a (small) non-realizable sub order type to proof the non-realizability of a (large) order type.

3.6.2 The Rotation Functions

Let $\Lambda \in \Lambda_n^*$ be realizable and let $i_0 \in \partial\Lambda$. Since Λ is realizable there exists $P \in R(\Lambda)$. Since there are no collinear points, we can find a line that separates the extremal point p_{i_0} from the others. Formally: There exists a function $g(x, y) = ax + by + c$ such that $g(p_{i_0}) > 0$ and $g(p_i) < 0$ for $i \neq i_0$.

As any general pointset $P \in R(\Lambda)$ is invariant to affine linear transformations that conserve the orientation of each point triple, i.e.,

$$(x, y) \mapsto (x, y) \cdot M^T + (x_0, y_0)$$

where $\det M > 0$ (e.g., rotation, translation, shearing, and certain scalings), we can find a bijective affine linear transformation s_{P,i_0} such that $P' = s_{P,i_0}(P)$ holds, where

- $p'_{i_0} = (1, 0)$,
- $\forall i \neq i_0 \colon x(p'_i) \leq -1$, and
- $\exists! j, k \colon p'_j = (-1, 1),\ p'_k = (-1, -1)$.

By construction, $a = 1, b = 0, c = 0$ (i.e., the y-axis) gives a separation line in P'. We can define a rotation function r that maps the y-axis to the line at infinity in a way such that p'_{i_0} is moved over the line at infinity and others are not. Formally that is

$$\begin{aligned} r \colon \{(x, y) \in \mathbb{R}^2 \mid x \neq 0\} &\to \{(x, y) \in \mathbb{R}^2 \mid x \neq 0\} \\ (x, y) &\mapsto (-\frac{1}{x}, -\frac{y}{x}). \end{aligned}$$

Note that this function corresponds to the -90 degree rotation around the y-axis in the three dimensional space, i.e., the mapping $(x, y, z) \mapsto (z, y, -x)$, that maps a point $(x, y, 1)$ to $(1, y, -x)$. Projecting the resulting point into the $z = 1$ plane gives $(-\frac{1}{x}, -\frac{y}{x})$.

Definition 22. *Using the functions* s_{P,i_0} *and* r *as above we can define the rotation function* r_{P,i_0} *as*

$$r_{P,i_0} := r \circ s_{P,i_0}.$$

Corollary 5. *Let $\Lambda \in \Lambda_n^*$, $P \in R(\Lambda)$ and $i_0 \in \partial\Lambda$. Then*

$$P' := r_{P,i_0}(P)$$

is a realization of $\Lambda' = f_{i_0}(\Lambda) \in \langle\Lambda\rangle$, i.e., $P' \in R(\Lambda')$.

Theorem 2. *For $P \in \mathrm{PC}_n^*$ either all $\Lambda \in P$ are realizable or all $\Lambda \in P$ are abstract. A projective class P is said to be **realizable** or **pure abstract**, respectively.*

Proof. Let $k \in \mathbb{N}_0$ and let $i_0, \ldots, i_{k-1} \in \{1, \ldots, n\}$ be the flipping sequence such that $\Lambda^{(0)}, \ldots, \Lambda^{(k)} \in \Lambda_n^*$ with $\Lambda^{(0)} \in [\Lambda], \Lambda^{(k)} \in [M]$. (see Definition 14)

Assume that Λ is realizable. According to Lemma 2 there also exists a realization P_0 of $\Lambda^{(0)}$. For $j = 1, \ldots, k$ (iteratively) $P_j := r_{P_{j-1},i_{j-1}}(P_{j-1})$ gives a realization of $\Lambda^{(j)}$. Thus, P_k is a realization of $\Lambda^{(k)} \in [M]$. According to Lemma 2 there exists a realization of M. □

Corollary 6. *The problem of realizing $\Lambda \in \Lambda_n^*$ can be solved by realizing another $M \in \langle\Lambda\rangle$ and transforming back the solution to the original problem using rotation functions.*

We denote this realization method as **back-rotation-realization**.

Theorem 3. *Let $\Lambda \in \Lambda_n^*$. There are realizations with real coordinates if and only if there are natural realizations. Formally,*

$$R(\Lambda) = \emptyset \Leftrightarrow R^Q(\Lambda) = \emptyset \Leftrightarrow R^I(\Lambda) = \emptyset \Leftrightarrow R^N(\Lambda) = \emptyset.$$

Proof. By the defintion of real, rational, integer, and natural pointsets we have

$$\mathcal{P}^N \subseteq \mathcal{P}^I \subseteq \mathcal{P}^Q \subseteq \mathcal{P}.$$

So only one direction needs to be shown for each of the three equivalences.

- The third equivalence holds because of the translation invariance and because there exists a minimum (pointsets are finite by definition):

  Let $P^I = ((x_1, y_1), \ldots, (x_n, y_n)) \in \mathcal{P}_n^I$ be an integer pointset. We define $X := \min_{i\in\{1,\ldots,n\}} x_i$ and $Y := \min_{i\in\{1,\ldots,n\}} y_i$. Then

  $$P^N := ((x_1 - X, y_1 - Y), \ldots, (x_n - X, y_n - Y))$$

  is a natural pointset that induces the same Λ-matrix.

- The second equivalence holds because of the invariance of multiplication with positiv numbers:

  Let $x_i = \frac{a_i}{b_i}$ and $y_i = \frac{c_i}{d_i}$ with $a_i, c_i \in \mathbb{Z}$ and $b_i, d_i \in \mathbb{N}$ be the coordinates of the points in the pointset $P^Q = ((x_1, y_1), \ldots, (x_n, y_n)) \in \mathcal{P}_n^Q$. We define $B := \prod_{i\in\{1,\ldots,n\}} b_i$ and $D := \prod_{i\in\{1,\ldots,n\}} d_i$. Then

  $$P^I := ((B \cdot x_1, D \cdot y_1), \ldots, (B \cdot x_n, D \cdot y_n))$$

  is an integer pointset that induces the same Λ-matrix.

- A proof for the first equivalence:

  Let $P \in R(\Lambda)$. Let p_{i_0} be the first point that has non-rational coordinates, i.e., $p_i \in \mathbb{Q}^2$ for $i < i_0$ and $p_{i_0} \notin \mathbb{Q}^2$. Since $\Lambda_{i_0,j,k} \neq 0$ for any j, k (i_0, j, k pairwise different), the Euclidean distance from p_{i_0} to any line is greater than some $\epsilon > 0$ for any j, k-tuple. Hence, the point p_{i_0} can be moved arbitrarily in an ϵ-neighborhood without changing any orientation triple. As the rational numbers are dense in the real numbers there exists a point $p'_{i_0} \in \mathbb{Q}^2$ with $||p_{i_0} - p'_{i_0}||_2 < \epsilon$. Hence,

  $$P' = (p_1, \ldots, p_{i_0-1}, p'_{i_0}, p_{i_0+1}, \ldots, p_n)$$

  is a realization of Λ where the first i_0 points all have rational coordinates. Applying this inductively yields a rational realization.

□

Corollary 7. *Deciding whether there exists a real realization is as hard as deciding whether there exists a natural (or integer) realization.*

Theorem 4. *The decision problem whether there exists a realization of a given $\Lambda \in \Lambda_n^*$ is known to be NP-hard. [32, 31]*

Theorem 5 (Goodman and Pollack)**.** *There exists $\Lambda \in \Lambda_n^*$ such that for every natural realization* $\operatorname{diam} P = \Omega(2^{2^n})$ *holds. [29, 31]*

Instead of storing a λ-matrix using $O(n^2 \log n)$ space we can also store the realization of λ. For small n this method has been shown to be more efficient.

For $n \leq 11$ an enumeration of all order types and projective classes was given by Aichholzer et. al. [15, 31]. A data base up to $n = 10$ was also generated by Bokowski, Laffaille, and Richter-Gebert but has never been published [21, 20].

Using a definition that allows collinear index triples an enumeration of all order types was given by Lukas Finschi and Komei Fukuda for $n \leq 9$ [26, 27].

3.7 Approaches for Realization

3.7.1 QCP-Realization

Theorem 6. *Let $\Lambda \in \Lambda_n^*$. There is a QCP (Quadratic Constrained Program) with $\binom{n}{3}$ quadratic constraints in $2n$ variables that decides whether realization is possible or not. For every orientation triple $i, j, k \in \{1, \ldots, n\}$ with $\Lambda_{ijk} = 1$ there is a quadratic constraint*

$$x_i y_j - x_j y_i + x_j y_k - x_k y_j + x_k y_i - x_i y_k > 0$$

that is equivalent to $\Lambda_{ijk} = 1$ by definition. By construction, any solution $x_1, \ldots, x_n, y_1, \ldots, y_n$ of this problem yields a realization

$$P := ((x_1, y_1), \ldots, (x_n, y_n)) \in R(\Lambda)$$

and vice versa.

Remark that we can also write

$$x_i y_j - x_j y_i + x_j y_k - x_k y_j + x_k y_i - x_i y_k \geq 1$$

since "≥ 1" is not more restrictive than "> 0", is as we only consider finite pointsets. For every real pointset $P = ((x_1, y_1), \ldots, (x_n, y_n))$ there exists

$$min\{x_i y_j - x_j y_i + x_j y_k - x_k y_j + x_k y_i - x_i y_k \mid \Lambda_{ijk}(P) = 1\} =: \epsilon > 0$$

and we have the desired property "≥ 1" in the scaled pointset

$$\widetilde{P} := ((\frac{1}{\epsilon}x_1, \frac{1}{\epsilon}y_1), \ldots, (\frac{1}{\epsilon}x_n, \frac{1}{\epsilon}y_n)).$$

According to the previous theorem we can either let $x_i, y_i \in \mathbb{R}$, $x_i, y_i \in \mathbb{Z}$, or $x_i, y_i \in \mathbb{N}_0$ in the QCP - the feasibility will not change.

Proposition 10. *Finding a solution of an arbitrary QCP is known to be NP-hard in general even if some certain instances (e.g., if the problem is convex) can be solved efficiently.*

Proof. Quadratic Programming is known to be NP-hard [34] and a QP (Quadratic Program) is a QCP without quadratic constraints. □

Note 5. *As $x(1-x) = 0$ is a quadratic constraint that forces a variable x to be boolean, a QCP can be used to find (or disproof) a solution to a given SAT-instance.*

Proposition 11. *A feasible point of the problem (P)*

$$\begin{aligned} &\textit{minimize } 0 \\ &\textit{subject to } g(x_1, \ldots, x_n) \leq 0, \; 1 \leq j \leq m \\ &\qquad x_i \in \mathbb{R} \qquad\qquad , \; 1 \leq i \leq n \end{aligned}$$

can be found by solving the problem (P')

$$\begin{aligned} &\textit{minimize } \sum_{i=1}^{m} \phi_i(g_i(x_1, \ldots, x_n)) \\ &\textit{subject to } x_i \in \mathbb{R} \qquad\qquad\qquad , \; 1 \leq i \leq n \end{aligned}$$

where $\phi_i : \mathbb{R} \to \mathbb{R}$ maps any non-positive number to zero and any positive number to a positive number, e.g.,

$$x \mapsto \begin{cases} x^{\alpha} & \textit{if } x > 0 \\ 0 & \textit{else.} \end{cases}$$

By construction, any optimal solution of (P') with an objective value 0 is also a solution of (P) and vice versa. Hence, if the minimal objective value of (P') is greater than 0 then (P) is infeasible.

According to this preposition a feasible point of a QCP can be found by minimizing a function.

3.7.2 Grassmann-Plücker Heuristic

Theorem 7 (Grassmann-Plücker Heuristic. "GP-Heuristic")**.** *An LP (Linear Program) with* $10\binom{n}{5}$ *constraints in* $\binom{n}{3}$ *variables can be constructed to a given* $\Lambda \in \Lambda^*$*. If this LP has an optimal solution then* Λ *is not realizable (and thus also the order type* $[\Lambda]$ *and the projective class* $\langle\Lambda\rangle$ *are not realizable as well). Otherwise no statement about realizability can be made. [31, 21]*

Proof. Let $\Lambda \in \Lambda_n^*$. Assume that Λ is realizable. Let P be a realization of Λ. For $i, j, k \in \{1, \ldots, n\}$ let

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

Remark that there are no collinear points and thus $X_{i,j,k} \neq 0$ holds for pairwise different i, j, k.

For $a, b, c, d, e \in \{1, \ldots, n\}$ pairwise different, the Grassmann-Plücker equation must hold, i.e.,

$$X_{abc} \cdot X_{ade} + X_{abd} \cdot X_{aec} + X_{abe} \cdot X_{acd} = 0$$

We can observe two properties of the GP-equation:

1. $(a', b', c', d', e') := (a, c, b, d, e)$ (i.e., switching b and c) yields the same equation:

$$\begin{aligned} 0 &= X_{abc} \cdot X_{ade} + X_{abd} \cdot X_{aec} + X_{abe} \cdot X_{acd} \\ &= X_{abc} \cdot X_{ade} + X_{acd} \cdot X_{abe} + X_{aec} \cdot X_{abd} \\ &= (-X_{acb}) \cdot X_{ade} + X_{acd} \cdot (-X_{aeb}) + (-X_{ace}) \cdot X_{abd} \\ &= -(X_{ab'c'} \cdot X_{ade} + X_{ab'd} \cdot X_{aec'} + X_{ab'e} \cdot X_{ac'd}) \end{aligned}$$

2. $(a', b', c', d', e') := (a, c, d, e, b)$ (i.e., rotating b, c, d, e) yields the same equation:

$$\begin{aligned} 0 &= X_{abc} \cdot X_{ade} + X_{abd} \cdot X_{aec} + X_{abe} \cdot X_{acd} \\ &= X_{acd} \cdot X_{abe} + X_{aec} \cdot X_{abd} + X_{abc} \cdot X_{ade} \\ &= -X_{acd} \cdot X_{aeb} - X_{ace} \cdot X_{abd} - X_{acb} \cdot X_{ade} \\ &= -(X_{ab'c'} \cdot X_{ad'e'} + X_{ab'd'} \cdot X_{ae'c'} + X_{ab'e'} \cdot X_{ac'd'}) \end{aligned}$$

According to this two properties any permutation of b, c, d, e yields the same equation. Hence, for a fixed a there are only $\binom{n-1}{4}$ equations.

As $\Lambda_{ijk} = \operatorname{sgn} X_{ijk}$ holds by definition, exactly one or two of the following terms must have negative sign to fulfil the GP-equation:

$$\Lambda_{abc} \cdot \Lambda_{ade},\ \Lambda_{abd} \cdot \Lambda_{aec},\ \Lambda_{abe} \cdot \Lambda_{acd}.$$

W.l.o.g. one term has negative sign. Otherwise, switch c and e.

W.l.o.g. $\Lambda_{abd} \cdot \Lambda_{aec}$ is the negative term. Otherwise $(c', d', e') := (d, e, c)$ or $(c', d', e') := (e, c, d)$ fulwills this property.

We can rewrite our equation to

$$X_{abc} \cdot X_{ade} + X_{abe} \cdot X_{acd} = X_{abd} \cdot X_{ace}$$

where every product has positive sign. This yields two inequalities:

$$X_{abc} \cdot X_{ade} < X_{abd} \cdot X_{ace}$$
$$X_{abe} \cdot X_{acd} < X_{abd} \cdot X_{ace}$$

Let $L_{ijk} := \log |X_{ijk}|$ for any $i, j, k \in \{1, \ldots, n\}$. We can apply the logarithm on both equations

$$L_{abc} + L_{ade} < L_{abd} + L_{ace}$$
$$L_{abe} + L_{acd} < L_{abd} + L_{ace}$$

and rewrite them to

$$L_{abc} + L_{ade} - L_{abd} - L_{ace} < 0$$
$$L_{abe} + L_{acd} - L_{abd} - L_{ace} < 0$$

Hence, a linear system of $2n\binom{n-1}{4} = 10\binom{n}{5}$ strict inequalities in $\binom{n}{3}$ variables is established. □

When modelling the LP to check the GP-Heuristic for a given $\Lambda \in \Lambda_n^*$ we need to handle the strict inequalities for every constraint, i.e., we have a problem (P) of the form

$$\begin{aligned} &\text{maximize } 0 \\ &\text{subject to } a_{j1}x_1 + \ldots + a_{jn}x_n < 0, \ 1 \leq j \leq m \\ &\qquad x_i \in \mathbb{R} \qquad\qquad\qquad\qquad , \ 1 \leq i \leq n \end{aligned}$$

We can find a solution of (P) by solving a similar linear problem (P')

$$\begin{aligned} &\text{maximize } \epsilon \\ &\text{subject to } a_{j1}x_1 + \ldots + a_{jn}x_n + \epsilon \leq 0, \ 1 \leq j \leq m \\ &\qquad \epsilon \geq 0, x_i \in \mathbb{R} \qquad\qquad\qquad , \ 1 \leq i \leq n \end{aligned}$$

By construction, any feasible (not necessarily optimal) solution $(\epsilon, x_1, \ldots, x_n)$ of (P') with $\epsilon > 0$, i.e., objective value greater than 0, yields a solution of (P) since

$$a_{j1}x_1 + \ldots + a_{jn}x_n \leq -\epsilon < 0.$$

Also, every solution $(x_1, \ldots, x_n)$ of (P) yields an

$$\epsilon := \min_{1 \leq j \leq m} -(a_{j1}x_1 + \ldots + a_{jn}x_n) > 0$$

such that $(\epsilon, x_1, \ldots, x_n)$ is a feasible solution of (P') with objective value greater than 0.

Furthermore, assume that there exists a solution $(\epsilon, x_1, \ldots, x_n)$ of (P'). Then $(\lambda\epsilon, \lambda x_1, \ldots, \lambda x_n)$ is also a feasible solution of (P') for any $\lambda > 0$ and thus (P') is unbounded.

Lemma 4. *The following three statements are equivalent:*

1. *(P) is infeasible;*
2. *(P') has optimal value 0;*
3. *(P') is bounded.*

Note 6. *Note that there are lots of ways to model the GP-Heuristic as an LP. For example:*

- *One could add the constraint $\epsilon \leq 1$ to (P') such that the optimal value of (P') is either 1 or 0.*
- *One could add the constraint $\epsilon \geq 1$ to (P') such that (P') is feasible if and only if (P) is feasible.*
- *One could use the constraints $a_{j1}x_1 + \ldots + a_{jn}x_n \leq -1$ to model (P').*

Proposition 12 (Some facts about LPs)**.**

- *It is known that an LP can be solved using interior-point methods in weakly polynomial time. [30] Thus, it is possible to find an optimal solution, i.e., a proof for infeasibility in polynomial elementary operations.*
- *It is also known that the simplex algorithm takes expected polynomial time. [22]*
- *If an optimal solution is already known the validation can be done much faster than solving the problem from scratch because the optimality conditions for LPs can be checked by performing only one simplex step, i.e., by inverting the matrix of coefficients once. [37, Chapter 6]*

Corollary 8. *Let $\Lambda \in \Lambda^*$.*

- *If a realization P of Λ is known we can check its validity in $\mathcal{O}(n^3 \log n)$ by calculating $\Lambda(P)$.*
- *If the LP for the GP-Heuristic has an optimal value 0 then Λ is non-realizable.*
- *Given a solution of the LP (GP-Heuristic) the optimality can be validated efficiently by checking the optimality conditions. Therefore, such an optimal solution yields a non-realizability-certificate.*

3.7.3 Back-Rotation-Realization

Given $\Lambda \in \Lambda_n^*$ and a realization P' of Λ' for some $\Lambda' = f_{i_1} \circ \ldots \circ f_{i_k} \circ \Lambda \in \langle \Lambda \rangle$ we want to find a realization P of Λ by rotating the pointset P'. This can be done by applying the rotation functions $r_{i_1}, \ldots, r_{i_k}$ one after another. Since every single rotation blows up the coordinates we might want to find a better way to perform back rotation to get a realization with smaller coordinates.

We also have to be careful when rotating, since rotation around arbitrary axes and arbitrary degree might get hard to calculate. For example, a rotation

by 45 degrees requires an evaluation of $sin(\frac{\pi}{4}) = \frac{1}{\sqrt{2}}$ and $cos(\frac{\pi}{4}) = \frac{1}{\sqrt{2}}$ which can only be approximated since $\sqrt{2}$ is irrational. Hence, the resulting coordinates might be useless because of numerical instability.

Note that instead of generating approximated irrational values of the sine and cosine function we could also use pythagorean triples which also fulfill the required properties needed to perform the rotation.

We use another approach to avoid these problems: From now on we only consider the rotation function r (from Subsection 3.6.2) that moves the y-axis to infinity, i.e., the mapping

$$r\colon (x, y) \mapsto (-\frac{1}{x}, -\frac{y}{x}).$$

By definition, the induced Λ-matrix of a pointset is invariant to affine linear transformations that keep the positive orientation of a triple (e.g., rotation, translation, shearing), i.e., any mapping

$$(x, y) \mapsto (x, y) \cdot M^T + (x_0, y_0)$$

with $\det M > 0$ and x_0, y_0 arbitrary.

Let $P = \{p_1, \ldots, p_n\} \in \mathcal{P}_n$ be a pointset and let $A \uplus B = \{1, \ldots, n\}$ be a partition for some non-empty, disjoint sets A and B. We can write

$$P = P_A \uplus P_B := \{p_a \mid a \in A\} \uplus \{p_b \mid b \in B\}.$$

Furthermore, let $L = \{(x, y) \mid \alpha x + \beta y + \gamma = 0\}$ be a line that separates sets P_A and P_B, i.e.,

- $\alpha x + \beta y + \gamma > 0$ for $(x, y) \in P_A$
- $\alpha x + \beta y + \gamma < 0$ for $(x, y) \in P_B$

We can transform P to $P' = f(P)$ such that $x(p'_a) > 0$ for $a \in A$ and $x(p'_b) < 0$ for $b \in B$ by moving L to the y-axis. Since $L = g(y\text{-axis}) = g(\{(0, y) \mid y \in \mathbb{R}\})$ for the mapping

$$g : (x, y) \mapsto (x, y) \cdot \underbrace{\begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix}^T}_{M^T} + \underbrace{\frac{(-\gamma)}{\alpha^2 + \beta^2}(\alpha, \beta)}_{(x_0, y_0)}$$

the inverse $g^{-1} =: f$ is the affine transformation we were looking for. Note that in P' the y-axis separates P'_A and P'_B.

Applying the rotation function r to P' we get a pointset $P'' = r(P')$ and $\Lambda'' := \Lambda(P'')$ such that

- the orientation of 3 points in P''_B stays the same as in P' and thus the same as in P:
$$i, j, k \in B \Rightarrow \Lambda''_{ijk} = \Lambda_{ijk}$$

- the orientation of 2 points in P''_B and 1 point in P''_A alters:

$$i \in A,\ j,k \in B \Rightarrow \Lambda''_{ijk} = -\Lambda_{ijk}$$

- the orientation of 1 point in P''_B and 2 points in P''_A stays the same:

$$i,j \in A,\ k \in B \Rightarrow \Lambda''_{ijk} = \Lambda_{ijk}$$

- the orientation of 3 points in P''_A alters:

$$i,j,k \in A \Rightarrow \Lambda''_{ijk} = -\Lambda_{ijk}$$

Hence, the induced Λ-matrix of P'' is the same as applying the flip functions $f_{a_1}, \ldots, f_{a_k}$ (i.e., $f_{a_1} \circ \ldots \circ f_{a_k}$) to the Λ-matrix of P.

Note that there exists $\pi \in S_k$ such that $a_{\pi_1}, \ldots, a_{\pi_k}$ is a flipping sequence, i.e., $a_{\pi_k} \in \partial(a_{\pi_{k-1}} \circ \ldots \circ a_{\pi_1} \circ \Lambda)$. Such a permutation π can be found by sorting P'_A by descending x-coordinates since the rightmost point of a pointset is always extremal by definition.

Lemma 5. *Let $\Lambda \in \Lambda^*_n$, $P \in R(\Lambda) \cap \mathcal{P}^I_n$, $\Lambda' \in \langle\Lambda\rangle$ with $\Lambda' = f_{i_k} \circ \ldots \circ f_{i_1} \circ \Lambda$ and let $A := \{i_1, \ldots, i_k\}$. Then there exists a seperating line $L = \{(x,y) \mid \alpha x + \beta y + \gamma = 0\}$ with $\alpha, \beta \in \mathbb{Z}$, $\alpha, \beta \in \mathcal{O}(\operatorname{diam} P)$, that separates P_A and P_B.*

Proof. We know that A and B can be separated. This holds since Λ' is known to be realizable by applying the rotations function as mentioned above that gives a realization with large coordinates.

W.l.o.g.

- $x(p_a) > 0$ holds for $a \in A$ and $x(p_b) < 0$ holds for $b \in B$ (transformation from above);
- $y(p_{a_0}) = y(p_{b_0}) = 0$ holds for some $a_0 \in A$ and some $b_0 \in B$ (can be assumed since orientations are shearing invariant).

We construct two affine mappings l_1, l_2 such that

- $l_i(x,y) := \alpha_i x + \beta_i x + \gamma_i$;
- $l_i(p_a) \geq 0$ for $a \in A$ and $l_i(p_b) \leq 0$ for $b \in B$;
- the slope $\frac{\beta_1}{\alpha_1}$ is maximal and the slope $\frac{\beta_2}{\alpha_2}$ is minimal.

Note that $\frac{1}{\alpha_i}$ exists since α_i needs to be positive by our construction.

Since $l_i(p_a) \geq 0$ for $a \in A$ and $l_i(p_b) \leq 0$ for $b \in B$ must hold, there must exist points $a^1, a^2 \in A$ and $b^1, b^2 \in B$ such that the inequality is fulfilled with equality. Otherwise, there would exist a mapping with a better slope (i.e., larger or smaller, respectively). Since there are at least 3 points in P, $a^1 \neq a^2$ or $b^1 \neq b^2$ must hold. W.l.o.g. $b^1 \neq b^2$ holds (otherwise switching A and B does the trick).

Hence, we only need to iterate over $a \in A$ and $b \in B$ and check for maximality or minimality, respectively, to find l_1 and l_2. The mapping given by $a = (x_a, y_a)$ and $b = (x_b, y_b)$ and $n := (x_n, y_n) := (y_b - y_a, x_a - x_b)$:

$$(x, y) \mapsto n_x(x - x_a) + n_y(y - y_a) = \underbrace{n_x}_{\alpha} x + \underbrace{n_y}_{\beta} y + \underbrace{(-n_x x_a - n_y y_a)}_{\gamma}$$

We know l_1 and l_2 now and can set $l_3 := l_1 + l_2$.

- In case $a^1 \neq a^2$ we have $l_1(p_{a^1}) \neq l_2(p_{a^1})$ and thus $l_3(p_{a^1}) > 0$. Otherwise, there would exist collinear points by the construction of l_i. The same argument also holds for a^2. For any other $a \in A \backslash \{a^1, a^2\}$ the same holds since a^i is the only point where $l_i(p_{a^i}) = 0$ holds. The same holds for B and therefore $l_3(p_a) > 0$ for $a \in A$ and $l_3(p_b) < 0$ for $b \in B$.

  Hence, l_3 yields a line $L = \{(x, y) \mid l_3(x, y) = 0\}$ seperating P_A and P_B.

- In case $a^1 = a^2$ we have $l_3(p_{a^1}) = 0$, and thus $l_3(p_a) \geq 0$ for $a \in A$ and $l_3(p_b) \leq \epsilon < 0$ for $b \in B$ with $\epsilon := \min_{b \in B} l_3(p_b) \in \mathbb{Z}$.

  The line $L = \{(x, y) \mid 2 \cdot l_3(x, y) = \epsilon\}$ separates P_A and P_B.

□

Lemma 6. *Let $\Lambda \in \Lambda_n^*$, $P \in R(\Lambda) \cap \mathcal{P}_n^I$, $\Lambda' \in \langle \Lambda \rangle$ with $\Lambda' = f_{i_k} \circ \ldots \circ f_{i_1} \circ \Lambda$ and let $A := \{i_1, \ldots, i_k\}$. Then we can calculate a realization $P' \in R(\Lambda')$ by calculating*

$$P' = r(f(P))$$

where f is an affine transformation that moves the separating line L to the y-axis.

Furthermore, let $B := \lceil \log_2(\operatorname{diam} P) \rceil$. We can find $P' \in R(\Lambda') \cap \mathcal{P}_n^I$ such that $\log_2(\operatorname{diam} P') = \mathcal{O}(nB)$. Therefore, the encoding of P' succeeds with $\mathcal{O}(n^2 B)$ bits.

Proof. The first part was already proven on page 19 in the text above lemma 5 and we only have to show the second part.

Let $P \in R(\Lambda) \cap \mathcal{P}_n^I$ and $B = \lceil \log_2(\operatorname{diam} P) \rceil$. Let $\alpha, \beta, \gamma \in \mathbb{Z}$ with $\log_2 \alpha, \log_2 \beta, \log_2 \gamma \in \mathcal{O}(B)$ be the parameters of the separating line L. Instead of using the inverse $g^{-1} =: f$ with $f(x, y) = (x - x_0, y - y_0) \cdot M^{-T}$ we can use $\widetilde{f}(x, y) := (x - x_0, y - y_0) \cdot (M^{\#})^T$ where

$$M^{\#} := \begin{pmatrix} \alpha & \beta \\ -\beta & \alpha \end{pmatrix}$$

is the adjoint matrix of M. Since $M^{\#} = \det M \cdot M^{-1}$ and $\widetilde{f} = f \cdot \det M$ hold, $\widetilde{f}$ also yields the desired transformation as a scaling of f. So $\widetilde{f}$ maps integer points to integer points by construction and thus $P' := \widetilde{f}(P)$ has integer coordinates.

When rotating $P'' := r(P')$ we get rational points where the denominator of $x(p_i'')$ and $y(p_i'')$ is $x(p_i')$ for any i.

Let

$$C := \prod_{p' \in P'} x(p').$$

Since $\alpha, \beta, x_i, y_i \in 2^{\mathcal{O}(B)}$ holds for $p_i = (x_i, y_i)$ for any i it follows that

$$\log_2 x(p'_i) = \log_2(\alpha \cdot (x_i - x_0) + \beta \cdot (y_i - y_0)) = \log_2(2^{\mathcal{O}(B)} + 2^{\mathcal{O}(B)}) = \mathcal{O}(B)$$

and thus

$$\log_2(C) = \sum_{b \in B} \log_2 x(p'_i) = n \cdot \mathcal{O}(B).$$

So we can scale P'' by C and get an integer pointset

$$P''' := C \cdot P'' := \{(Cx, Cy) \mid (x, y) \in P''\}$$

with $\log_2(\operatorname{diam} P''') = \mathcal{O}(nB)$. □

Corollary 9. *Let $\Lambda \in \Lambda_n^*$ such that for every integer realization $P \in R^I(\Lambda)$ $\log_2(\operatorname{diam} P)$ is superpolynomial in n (i.e., can not be bounded by any polynomial in n). Then for every $\Lambda' \in \langle \Lambda \rangle$ and every $P' \in R(\Lambda')$ it holds that $\log_2(\operatorname{diam} P')$ is superpolynomial.*

3.8 Research Topics

Since we mainly focused on realization techniques, none of the following definitions was needed so far, but as `pyotlib` also provides certain implementations on other topics and as some results on these topics could be achieved using these implemenations, we decided to give a short listing of those definitions and the corresponding references.

3.8.1 k-Gons and k-Holes

For a general pointset P it is natural to define a **k-gon** as a simple polygon spanned by k points of P. A k-gon is said to be **empty** if and only if it does not contain any points of P in its interior. An empty k-gon is also said to be a **k-hole**. A k-gon spanned by a convex polygon is said to be **convex**. We also denote a 3-gon as triangle, a 4-gon as quadrilateral, a 5-gon as pentagon, a 6-gon as hexagon, and so on. Further, we can define $g_k(n)$ ($h_k(n)$) as the least number of convex k-gons (k-holes) determined by any set of n points in general position. [18, 31, 19, 24, 25]

Remark that there exists exactly one Λ-matrix $\Lambda^{(k)} \in \Lambda_k^*$ with $\Lambda^{(k)}_{abc} = 1$ for $a < b < c$ which corresponds to any pointset with k points in convex position and counterclockwise labeling. For an abstract order type we denote the tuple $(a_1, \ldots, a_k)$ as an **abstract convex k-gon** if and only if the sub order type induced by $\{a_1, \ldots, a_k\}$ is equal to $[\Lambda^{(k)}]$. Analogously we denote an abstract convex k-gon as **empty** (i.e., an **abstract convex k-hole**) if and only if any index b is in the extremal set of the sub order type induced by $\{a_1, \ldots, a_k, b\}$. Note that for realizable order types the definitions are conform.

3.8.2 Rectilinear Crossings

The **rectilinear crossing number** $\bar{cr}(n)$ is the minimum number of intersections in any drawing of K_n in the plane that has straight-line-segment edges. It can be shown that $\bar{cr}(n)$ is exactly the number of convex quadrilaterals $g_4(n)$. [35, 18, 31, 19, 9, 23]

3.8.3 Universal Pointset

A realizable order type $O \in \mathrm{OT}_n^*$ and any realization $P \in \cup_{\Lambda \in O} R(\Lambda)$ is said to be **universal of order** m if any realizable order type $O' \in \mathrm{OT}_m^*$ occurs as an induced sub order type of O.

Analogously we can denote a projective class $C \in \mathrm{PC}_n^*$ as **universal of order** m if any realizable projective class $C' \in \mathrm{PC}_m^*$ occurs as an induced sub projective class of C.

4 pyotlib

The provided library `pyotlib` contains implementations of all features described in this thesis. Using the class PYOTScript it is easy to write a script that iterates over a given input and performs a certain action. As the basic read- and write-operations are done in the background automatically, only the desired action needs to be implemented by when using `pyotlib`.

4.1 Programming Language and Modules

Most of the source code was written in Python 2.7 [11] but also some parts (especially bottlenecks) that were written in Cython [1] to gain a little speedup. As IPython [7] allows interactive computation with the python programming language this framework can also be used interactively.

For the realization of order types we wrote a small C implemenation `simple` which tries to find a realization by geometric insertion on an $n \times n$ integer grid. To speed up this implementation forward checking, backtracking, and a quadtree search technique have been implemented. This implementation can be used by `pyotlib` as executable (`simple_c`) or as shared library (`libsimple`).

Certain classes, especially Realization testers (see Subsection 4.3.3), require miscellaneous software to be installed when in use.

4.2 Supported Filetypes

The most important filetypes supported by `(merged) cape` [39, 38, 36] are also supported in `pyotlib`. Hence, order types enumerated by `cape` can be processed using `pyotlib` for example.

Since 4-byte integer coordinates (`b32` filetype) can be insufficient for a large number of points, the filetypes `pt` and `asc` were added to allow arbitrary large coordinates.

- `lt` - λ-matrix text
- `pb` - pointset binary (1, 2, and 4 byte encoding, i.e., `b08`, `b16`, and `b32`)
- `pt`, `asc` - pointset text

4.3 Provided Functionality

4.3.1 Core Classes

The core classes of `pyotlib` are:

- `SmallLambda` - This class can store a λ-matrix and provides basic functionality.
- `BigLambda` - This class can store a Λ-matrix and provides basic functionality.
- `PointSet` - This class can store a Pointset (not necessary integer) and provides basic functionality.

A `SmallLambda` and a `PointSet` instance can be transformed into an instance of `BigLambda` using the `toBigLambda` method. A `BigLambda` and a `PointSet` instance can be transformed into an instance of `SmallLambda` using the `toSmallLambda` method.

Since realizations are not that trivial there is no "toPointSet" method anywhere but if there is a realization known for a `SmallLambda` or `BigLambda` instance, it is stored as member variable `realization`.

4.3.2 Order Types and Projective Classes

As many applications require order types or projective classes each of the three classes provide a `getLexMin` method that returns an instance (with realization if a realization was given as member) that represents the lexicographical minimum of the order type $\lambda_{\min}$.

To calculate lexicographical minimum of the projective class $\lambda_{\min}^{\mathrm{PC}}$ the class `ProjectiveClass` provides a method `getRepresenterOT` that returns an instance of `BigLambda`. There is also a method `getAllOTs` that returns all order types in a projective class.

4.3.3 Realization

As there are many different methods to find (or disproof) a realization of an order type `pyotlib` provides an abstract class `RealizationTester` that allows to combine different RealizationTesters. Every RealizationTester instance can store a so called "parent-tester", that will be called whenever the realization (or the disproof) fails. So it is possible to build a RealizationTester-chain.

```
...
RealizationTester = None
if self.useMathematica:
    RealizationTester = MathematicaRealizationTester(RealizationTester)
else:
    RealizationTester = PascalWrapperRealizationTester(RealizationTester)

RealizationTester = PCRealizationTester(RealizationTester)

if not self.skipGP:
    RealizationTester = GPRealizationTester(RealizationTester)
...
```

Listing 1: Example code snipped from pyotlib.Realize.py

A list currently implemented Realization Testers:

- `GlpkQCPRealizationTester` - This tester formulates the QCP (see Theorem 6) as LP with integer variables and try to solve it using GLPK. Since there are no efficient methods to find optimal solution of (Mixed) Integer Linear Programs this implementation only works for a small number of points and small integer grids.

- `GPRealizationTester` - This tester allows to check if a given order type can be proven to be non-realizable by checking the GP-Heuristic using a simplex algorithm implementation provided by GLPK (GNU Linear Programming Kit) [2, 3]. Because GLPK works much faster when using floating points instead of rational numbers we try to find a solution with floating points first. Then we make use of a nice property of the simplex

algorithm: Whenever an optimal solution is found given by a corner in the polyhedron we can proof the optimality of this solution using rational numbers by validating the optimality of this corner. Therefore, only one simplex iteration needs to be done using rational numbers. We also force GLPK to solve the dual problem since this can be done much faster than the primal problem.

Whenever an order type can be proven to be non-realizable with GPRealizationTester a certificate is created:

- filename.OT#.proof.lp - The LP (textfile)
- filename.OT#.proof.lp.log.txt - Logfile generated by GLPK
- filename.OT#.proof.lp.sol - Solution in plain text format (can be loaded by GLPK and verified)
- filename.OT#.proof.lp.sol.txt - Solution in printable format

Using the parameter `useCPLEXpresolver` it is also possible to use IBM ILOG CPLEX [6] to find a solution that we will use as initial basis for GLPK. If CPLEX already found an optimal solution GLPK will only need to perform one Simplex Iteration to verify the solution - otherwise GPLK will continue as usual.

The `CPLEXremote` parameter allows to use CPLEX on a remote computer.

- `InnerPointHeuristicRealizationTester` - This tester tries to remove as many points as possible using an inner point heuristic and realize the resulting sub order type. If a realization can be found the removed points are re-inserted and the thus obtained realization of the initial order type is returned. There are two possibilites: Either only points are removed that can be guaranteed to be re-insertable or some other points can also be removed that can probably be re-inserted.

- `MathematicaRealizationTester` - This tester formulates a problem to decide whether a given order type is realizable and tries to solve it using Wolfram Mathematica [13]. As Mathematica provides lots of methods to solve problems, we decided to implement two similar methods which can be chosen by the parameter `formulateProblemAs`:

  - QCP - a Quadratic Constrained Program (QCP) (see Theorem 6) is formulated and solved that yields a realization for a given order type. The parameter `speedup_for_numerical_instability` can be used to gain a speedup but the solution might also not be feasible as a consequence of numerical instability.
  - MP - a Minimization Problem (MP) is formulated and solved that yields a solution of the corresponding QCP. The `MPmethod` parameter can be used to specify the method used by Mathematica for minimization.

  For both methods it is possible to specify the precision by the parameter `Precision` but we recommend to use the default since higher precision requires additional computation time.

  It is also possible to use Mathematica on a remote computer using the `remote` parameter.

- `MatlabRealizationTester` - This tester formulates a problem to decide whether a given order type is realizable and tries to solve it using Matlab [8] or Octave [4] (similar to `MathematicaRealizationTester`).
- `PascalWrapperRealizationTester` - This tester uses the Pascal implementation `realizeot` to realize an order type. Note that the executable `realizeot` has to be callable when using this tester.
- `PCRealizationTester` - This tester picks a (random) order type in the given projective class (given by a representative order type) and tries to realize it by calling the parent-tester. If the parent-tester fails another order type in the projective class will be tested, until either one order type can be realized, proven to be not realizable, or if there are no more order types left to be tested.
- `ScipyRealizationTester` - This tester formulates a problem to decide whether a given order type is realizable and tries to solve it using SciPy [12]. This tester implements the same functionality as `MathematicaRealizationTester` to provide a performant iterative realization with open source software.
- `SimpleRealizationTester` - This tester uses the C implementation `simple` to realize an order type. Note that the shared library `libsimple.so` and `libsimple.so.1` needs to be accessible when using this tester. Also make sure that this library exists when `pyotlib` is built and installed - otherwise rebuild `pyotlib`.
- `SimpleWrapperRealizationTester` - This tester uses the C implementation `simple` to realize an order type. Note that the executable `simple_c` has to be callable when using this tester.
- `PySimpleRealizationTester` - This tester gives a python implementation of `simple` to realize an order type.

4.3.4 Further Classes

- `PointSetOptimizer` - This class provides some experimental methods to minimize or beautify the integer coordinates of a given pointset.
- `PolygonCount` - This class provides some methods to count the number of (abstract) k-gons and k-holes. When counting 4-gons (i.e., rectilinear crossings) the method descriped by Aichholzer et. al. is used [17]. Otherwise we refer to [36].

4.3.5 Provided Scripts

There are some scripts provided by `pyotlib` that demonstrate its functionality:

- `pyotlib.BeautifyCoords.py` - try to beautify a given realization
- `pyotlib.CountSubOTs.py` - count the number of sub order types of size k
- `pyotlib.CountSubPCs.py` - count the number of sub projective classes of size k

- `pyotlib.EnumeratePC.py` - enumerate all order types in a projective class given by a representative order type
- `pyotlib.EnumerateSubOTs.py` - enumerate all sub-order types of size k
- `pyotlib.FindSubOTs.py` - given a list of order types of size n and a list of 'reference' order types of size rn ('reference n'), search for reference order types that occure as sub order types (size rn) of an order type in the first list (size n)
- `pyotlib.FindUniversalOT.py` - search for universal order types by counting the number of sub order types of size k
- `pyotlib.FindUniversalPC.py` - search for universal projective classes by counting the number of sub projective classes of size k
- `pyotlib.IterativeRealization.py` - script to find realizations of very large order types or to proof non-realizability:
  - phase 1: choose sub order types of size kmin..n to realize
  - phase 2: try to realize sub order types iteratively
  - phase 3: try to proof non-realizability
- `pyotlib.ListProperties.py` - prints some properties of a given order type
- `pyotlib.MinimizeCoords.py` - try to minimize the diameter of a given realization
- `pyotlib.PointMove.py` - try to move points in a given realization to generate pointsets with similar properties (BFS)
- `pyotlib.PointWalk.py` - try to move points in a given realization to generate pointsets with minimal number of (empty) convex k-gons (DFS)
- `pyotlib.PolygonCount.py` - count the number of (empty) convex k-gons in a given order type and output all order types with at most maxcount (or minimimal count so far)
- `pyotlib.Realize.py` - script to find realizations of order types or to proof non-realizability
- `pyotlib.UnifyOTs.py` - remove duplicates from a given list of order types
- `pyotlib.UnifyPCs.py` - remove duplicates from a given list of projective classes

When executing these scripts without parameters a usage information, i.e., a description of available parameters, is printed.

Another script `check_integer_grid` was written to count (enumerate) all order types with n points on an $N \times N$ integer grid that have realizations with pairwise different coordinates.

```
...
for X in combinations(range(N),n):
        for Y in permutations(range(N),n):
                PS = PointSet(n,[(X[i],Y[i]) for i in range(n)])
                ct += 1
                if not PS.hasCollinearPoints():
                        OT = PS.toSmallLambda().getLexMin()
                        OTstr=  OT.toString()

                        if OTstr not in OTs:
                                OTs.add(OTstr)
...
```

Listing 2: Code snipped from check_integer_grid.py

4.4 Some Benchmarks

The environment we used for most of the calculations and all benchmarks listed beneath:

- Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz, 6 GB memory, Gentoo Linux 3.13.6
- Python 2.7.6
- Cython version 0.20.1
- Mathematica 9.0 for Linux x86 (64-bit)
- GLPSOL: GLPK LP/MIP Solver, v4.48
- GNU Octave, version 3.8.1
- ILOG CPLEX 11.000

4.5 Realization of small Order Types

When realizing or proving the non-realizability of the 168 $n = 13$ order types with 3 convex 5-holes [36] we got the following results:

- Using MathematicaRealizationTester + QCP with the parameter `speedup_for_numerical_instability` the whole calculation took about 30 minutes.
- Using MathematicaRealizationTester + QCP without the parameter `speedup_for_numerical_instability` the whole calculation took about 60 minutes. Even if the parameter `speedup_for_numerical_instability` yields a speedup for small order types, the resulting realizations are not valid in most cases when realizing larger order types with $n \geq 20$.
- Using MathematicaRealizationTester + QCP without the PCRealizationTester the calculation took about 75 minutes and 6 order types remained unknown. Hence, we recommend to use PCRealizationTester or to enumerate the projective class using the `pyotlib.FullPCs.py` script manually.

- Using MathematicaRealizationTester + MP the whole calculation took about 23 minutes.
- Using PascalWrapperRealizationTester the whole calculation took about 4 hours.
- Using MatlabRealizationTester after about 4 hours 136 OTs could be realized (20 were still missing).

As $n = 13$ is relatively small the GP-Heuristic can be checked using GLPK in some seconds and CPLEX is not needed. The GP-Heuristic checks for the 168 order types took about 8 minutes, hence this time has to be subtracted when comparing the needed time for realization.

4.6 Realization of large Order Types

When comparing GLPK and CPLEX it is obvious that CPLEX is much faster than GLPK for larger n. For $n > 17$ GLPK already takes some minutes to find a solution where CPLEX (with the right parameters) can solve problems with $n \leq 25$ in a very short time. The most important feature of CPLEX is the so called sifting optimizer that solves the LP by solving certain sub-problems. [5]

The script `pyotlib.IterativeRealization.py` provides three phases to realize an Order Type (or to proof non-realizability) by realizing certain Sub Order Types. In phase 1 the sub order types are chosen by removing indices iteratively. Calculations have shown that the removal order can be crucial for the success or failure of the realization process, i.e., for phase 2 and phase 3. For re-inserting points in a realization of a sub order type there might be points that could be inserted easier than others and also for proving non-realizability it is desirable to find a non realizable sub order type of minimal size. Thus, some labeling approaches have been implemented and tested. When using Mathematica's Nelder-Mead Method [14, 33] for realization, i.e., to reinsert a point, we observed that inserting inner points (i.e., points inside the convex hull) is much easier than inserting outer points (i.e., points outside the convex hull). The parameter `labeling` can be set to

- `standard`: insert the points according to the given labeling (i.e., $1, 2, \ldots, n$);
- `onion`: insert the points according to the convex-hull-onion from the outside to the inside;
- `heuristic`: start with the convex hull and insert the point that is contained in the fewest number of triangles (i.e., yields the most crossings).

As listed above there are also RealizationTesters which can make use of Matlab, Octave and SciPy. Unfortunately, even though each of them also provides an implementation of the Nelder-Mead algorithm [33], none of them could compete with the MathematicaRealizationTester when realizing large order types. The ScipyRealizationTester takes much time and/or fails when realizing order types with 20 points and more, whereas the Mathematica implementation also works fine for 30 points and more. The MatlabRealizationTester seems to work

pretty good even for 20 points and more when using Octave (out of the box, i.e., no special parameters needed). Even though the MatlabRealizationTester (+Octave) is not as fast as the MathematicaRealizationTester it is also quite good for larger order type realization. When using Matlab the realization of order types of size about 20 seems to fail even with different parameters. Hence, we recommend to use MatlabRealizationTester+Octave as an alternative to MathematicaRealizationTester in case Mathematica is not available.

5 Overview of Results

5.1 Results on Empty Convex 5-Holes:

As already mentioned in the introduction all 168 resulting abstract order types with $n = 13$ points and exactly 3 convex 5-holes [36] could be shown to be realizable (156) or non-realizable (12).

We also managed to improve the upper bound on the minimal number of convex 5-holes $h_5(n)$ [18, 36] for $n = 19$ and $n = 20$ by realizing certain order types with `pyotlib`:

$$h_5(19) \leq 27,\ h_5(20) \leq 34$$

Using the scripts `pyotlib.PointMove.py` and `pyotlib.PointWalk.py` most known pointsets [36] could be altered in a way, such that the number of convex 5-holes was slightly decreased. For $n = 20$ an order type with 33 convex 5-holes could be found. Table 1 lists the old and the currently known upper bound on the minimal number of convex 5-holes for $n = 17..36$.

n	$h_5(n)$ (old bound)	$h_5(n)$ (improved bound)
17	$\leq$16	
18	$\leq$21	
19	$\leq$28	$\leq$27
20	$\leq$36	$\leq$33
21	$\leq$43	$\leq$41
22	$\leq$52	$\leq$48
23	$\leq$63	$\leq$59
24	$\leq$71	$\leq$67
25	$\leq$82	$\leq$75
26	$\leq$90	$\leq$88
27	$\leq$103	$\leq$100
28	$\leq$117	$\leq$115
29	$\leq$134	$\leq$131
30	$\leq$155	$\leq$150
31	$\leq$177	
32	$\leq$200	
33	$\leq$223	$\leq$222
34	$\leq$255	$\leq$251
35	$\leq$290	$\leq$283
36	$\leq$330	$\leq$322

Table 1: The old [36] and the improved upper bound on the minimal number of convex 5-holes $h_5(n)$ for $n = 17..36$ points.

5.2 Results on Rectilinear Crossing Numbers and on Realizability of Large Order Types:

Using `pyotlib` we managed to proof that three known $n = 96$ abstract order types with a small number of crossings [16, 9] (given as λ-matrices) are non-

realizable. For each of these three abstract order types we reduced the order type to sub order types of size $95, 94, \ldots, 11, 10$ using the bash command

```
for i in {95..10}; do
  pyotlib.EnumerateSubOTs.py ft lt n $(($i+1)) k $i \
  fp $(($i+1)).txt maxcount 1 && mv *$i* $i.txt;
done
```

Note that with this reduction approach only extremal points are removed. Then we started to realize the sub order types using the `pyotlib.Realize.py` script with the parameter `skipGP=1` (i.e., GP-Heuristic disabled) starting with $10, 11, \ldots$, i.e., the bash command

```
for i in {10..21}; do
  pyotlib.Realize.py n $i ft lt fp $i.txt skipGP 1 useMathematica 1;
done
```

Realizations of all sub order types of size $10, \ldots, 21$ could be found in about 30 minutes but the realization of the sub order type of size 22 failed every single time. Because GLPK [2] is not that fast for such a large n we generated the LP-file for the GP-Heuristic (see Theorem 7) and tested it using IBM ILOG CPLEX [6] which can also handle larger instances of linear programs in very short time (about 1 minute for this instance). The problem using CPLEX is that calculations can not be done using rational numbers. Hence, a floating point result calculated by CPLEX might not be a proof!

As CPLEX could not proof the LP (for the GP-Heuristic) to be unbounded, the order type seemed to be non-realizable by construction. Therefore, we decided to look for sub order types of this one for which CPLEX also could not find a better solution, i.e., sub order types that also seemed to be non-realizable.

We managed to find sub order types of size $21, 20, 19, 18$ and 17 iteratively by enumerating all sub order types of size $k-1$ of the sub order type of size k using the script `pyotlib.EnumerateSubOTs.py` and testing each of them for "probable non-realizability" using CPLEX. As GLPK could proof (using rational numbers) that the sub order type of size 17 is indeed not realizable, the $n = 96$ order type was proven to be not realizable as well.

This method worked for each of the three $n = 96$ order types. Hence, for order types of a large size this method might also work pretty good. An implemenation of the basic steps of this method is provided by the script `pyotlib.IterativeRealization.py` but we also recommend an interactive usage since a variety of ideas and tricks can be applied manually when realizing larger order types iteratively.

Using the `pyotlib.IterativeRealization.py` script with Mathematica an order type with $n = 40$ points and 33070 crossings could be realized. Hence, the upper bound on the number of crossings was slightly improved:

$$\bar{cr}(40) \leq 33070$$

Using the script `pyotlib.PointMove.py` and `pyotlib.PointWalk.py` some known pointsets [16, 9] could be altered in a way, such that the number of crossings was slightly decreased. Table 2 lists the attained improvements on the upper bound on the minimal number of rectilinear crossings. We refer to [9] for the current values and more information.

n	$\bar{cr}(n)$
40	≤ 33070
52	≤ 99161
57	≤ 145170
58	≤ 156042
60	≤ 179523
61	≤ 192267
98	≤ 1347651
99	≤ 1404666

Table 2: Improvements on the minimal number of rectilinear crossings $\bar{cr}(n)$.

5.3 Results on Realizations of Minimal Size:

For this subsection when talking about "realizations", any realization needs to provide different x and y coordinates and must not contain cocircular points! That is because the order type database [10, 15] provides these properties.

Using `pyotlib` we could find realizations of all (realizable) $n = 9$ order types with 7 bit integer coordinates by minimizing the known coordinates and re-realizing the order types. Remark that the previous realizations from the order type database needed 16 bits. Except for some $n = 9$ order types that require 7 bits, for the others 6 bits are sufficient. Furthermore, every $n = 9$ order type could be realized on a 6-bit integer grid by dropping the desired properties.

For $n = 10$ the minimization task is not that easy and there are still order types that we could not realize with less than 13 bit integer coordinates. Except for some $n = 10$ order types that require 9 bits, for the others 8 bits are sufficient (without properties).

n	3	4	5	6	7	8	9	10
# bits needed (with properties)	2	2	3	3	4	5	≤ 7	≤ 13
# bits needed (without properties)	1	2	2	3	4	5	≤ 6	≤ 9

For example, to realize an order type with 5 or more points and pairwise different x and y coordinates at least 3 bits are needed since a 4×4 grid is insufficient. Therefore, 3 is the exact lower bound on the number of necessary bits as any $n = 5$ order type is realizable with 3 bits.

To show that there exists an order type with 7 points such that every realization needs more than 3 bits we wrote a script `check_integer_grid` that enumerates all order types with n points on a $N \times N$ grid. By executing

```
python check_integer_grid.py 8 7
```

it can be calculated in about one minute that only 70 of 135 order types can be realized on an 8×8 grid. Thus, at least 4 bits are needed to realize any order type with 7 or more points. So every order type with 7 points is realizable on a 16×16 grid as we could find a realization for every order type with 4 bit integer coordinates. As this script tests all possible pointsets it would take some time

to proof that any order type with k or more points for $k \geq 8$ can not be realized on a 16×16 grid since $\binom{16}{k} \cdot \binom{16}{k} \cdot k! \geq 10^{12}$ pointsets will be tested.

By executing `python check_integer_grid_withoutproperties.py 4 5` it can be shown that at most 2 bits are needed for any $n = 5$ order type (without properties).

By executing `python check_integer_grid_withoutproperties.py 4 6` it can be shown that at least 3 bits are needed for some $n = 6$ order types (without properties) as only 8 different $n = 6$ order types can be found on a 2-bit grid.

By executing `python check_integer_grid_withoutproperties.py 8 6` it can be shown that at most 3 bits are needed for any $n = 6$ order types (without properties).

By executing `python check_integer_grid_withoutproperties.py 8 7` it can be shown that at least 4 bits are needed for some $n = 7$ order types (without properties) as only 70 different $n = 7$ order types can be found on a 3-bit grid.

A better approach to show that a certain order type can not be realized on a k-bit integer grid is to use `PySimpleRealizationTester` which tries to realize it by geometric insertion. If no realization can be found, there is no such realization as all possibilites are checked. Using this approach we could show that there exist $n = 8$ order types that can not be realized on a 4-bit integer (even without properties). As any $n = 8$ order type can be realized (with properties) on a 5-bit integer grid, 5 is the exact lower bound on the number of necessary bits.

5.4 Results on Universal Order Types:

Using pyotlib an $n = 30$ order type (a result of other calculations) could be shown to be universal of order 8. Also an $n = 20$ projective class could be shown to be universal of order 8. The number of points could be improved slightly by removing certain points and moving points via using the script `pyotlib.PointMove.py`. The calculations on order 9 universal order types and projective classes is very time consuming and hence, we did not try to minimize the number of points for these results. The following table states the number of points of the smallest universal order types and projective classes of order m known so far:

m	3	4	5	6	7	8	9
OT	3	5	6	9	12	≤ 22	?
PC	3	4	5	7	9	≤ 14	≤ 30

As there can be at most $\binom{n}{m}$ sub order types of size m in an order type of

size n, a lower bound on the number of points needed can be calculated as

$$\binom{n}{8} \geq 3315 \Rightarrow n \geq 15,$$
$$\binom{n}{9} \geq 158817 \Rightarrow n \geq 20,$$
$$\binom{n}{10} \geq 14309547 \Rightarrow n \geq 29,$$
$$\binom{n}{11} \geq 2334512907 \Rightarrow n \geq 41.$$

Remark that the number of realizable order types ($n \leq 11$) was stated in [15, 31].

References

[1] Cython. http://docs.cython.org/src/quickstart/overview.html, Aug. 2014.

[2] GLPK (GNU Linear Programming Kit). http://www.gnu.org/software/glpk/, Aug. 2014.

[3] GLPK Reference Manual. http://kam.mff.cuni.cz/~elias/optimed/glpk.pdf, Aug. 2014.

[4] GNU Octave. http://www.gnu.org/software/octave/, Aug. 2014.

[5] IBM ILOG CPLEX - Sifting optimizer. http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r2/topic/ilog.odms.cplex.help/Content/Optimization/Documentation/CPLEX/_pubskel/CPLEX419.html, Aug. 2014.

[6] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/, Aug. 2014.

[7] IPython. http://ipython.org/ipython-doc/stable/interactive/tutorial.html, Aug. 2014.

[8] MathWorks MATLAB. http://www.mathworks.de/products/matlab/, Aug. 2014.

[9] On the Rectilinear Crossing Number. http://www.ist.tugraz.at/aichholzer/research/rp/triangulations/crossing/, Aug. 2014.

[10] Order Type Database. http://www.ist.tugraz.at/aichholzer/research/rp/triangulations/ordertypes/, Aug. 2014.

[11] Python. https://docs.python.org/2.7/, Aug. 2014.

[12] SciPy. http://www.scipy.org/about.html, Aug. 2014.

[13] Wolfram Mathematica. http://www.wolfram.com/mathematica/, Aug. 2014.

[14] Wolfram Mathematica - Nelder-Mead Method. http://mathworld.wolfram.com/Nelder-MeadMethod.html, Aug. 2014.

[15] 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.

[16] O. Aichholzer, F. Aurenhammer, and H. Krasser. On the Crossing Number of Complete Graphs. In *Proc.* 18^{th} *Ann. ACM Symp. Computational Geometry*, pages 19–24, Barcelona, Spain, 2002.

[17] O. Aichholzer, J. García, D. Orden, and P. Ramos. New results on lower bounds for the number of ($\leq k$)-facets. In *Proceedings EuroComb'07, Electronic Notes in Discrete Mathematics*, volume 29C, pages 189–193, 2007.

[18] O. Aichholzer, T. Hackl, and B. Vogtenhuber. On 5-Gons and 5-Holes. In A. Marquez, P. Ramos, and J. Urrutia, editors, *Computational Geometry: XIV Spanish Meeting on Computational Geometry, EGC 2011, Festschrift Dedicated to Ferran Hurtado on the Occasion of His 60th Birthday, Alcalá de Henares, Spain, June 27-30, 2011, Revised Selected Papers*, volume 7579 of *Lecture Notes in Computer Science (LNCS)*, pages 1–13. Springer, 2012.

[19] O. Aichholzer and H. Krasser. The Point Set Order Type Data Base: A Collection of Applications and Results. In *Proc. 13th Annual Canadian Conference on Computational Geometry CCCG 2001*, pages 17–20, Waterloo, Ontario, Canada, 2001.

[20] J. Bokowski, G. Laffaille, and J. Richter. Classification of non-stretchable pseudoline arrangements and related properties. *Manuscript*, 1991.

[21] J. Bokowski and J. Richter. On the Finding of Final Polynomials . *European Journal of Combinatorics*, 11(1):21–34, 1990.

[22] K.-H. Borgwardt. The Average number of pivot steps required by the Simplex-Method is polynomial. *Zeitschrift für Operations Research*, 26(1):157–177, 1982.

[23] P. Brass, W. O. J. Moser, and J. Pach. *Research problems in discrete geometry.* Springer, 2005.

[24] K. Dehnhardt. *Leere konvexe Vielecke in ebenen Punktmengen.* 1987.

[25] P. Erdös. On some problems of elementary and combinatorial geometry. *Annali di Matematica Pura ed Applicata*, 103(1):99–108, 1975.

[26] L. Finschi and K. Fukuda. Complete combinatorial generation of small point configurations and hyperplane arrangements. In *CCCG*, pages 97–100, 2001.

[27] L. Finschi and K. Fukuda. Generation of oriented matroids - a graph theoretical approach. *Discrete & Computational Geometry*, 27(1):117–136, 2002.

[28] J. E. Goodman and R. Pollack. Multidimensional Sorting. *SIAM J. Comput.*, pages 484–507, 1983.

[29] J. E. Goodman, R. Pollack, and B. Sturmfels. Coordinate Representation of Order Types Requires Exponential Storage. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 405–410, New York, NY, USA, 1989. ACM.

[30] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979.

[31] H. Krasser. *Order Types of Point Sets in the Plane.* PhD thesis, Institute for Theoretical Computer Science, Graz University of Technology, Austria, October 2003.

[32] N. Mnev. The universality theorems on the classification problem of configuration varieties and convex polytopes varieties. In O. Viro and A. Vershik, editors, *Topology and Geometry — Rohlin Seminar*, volume 1346 of *Lecture Notes in Mathematics*, pages 527–543. Springer Berlin Heidelberg, 1988.

[33] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[34] P. Pardalos and S. Vavasis. Quadratic programming with one negative eigenvalue is np-hard. *Journal of Global Optimization*, 1(1):15–22, 1991.

[35] E. R. Scheinerman and H. Wilf. The rectilinear crossing number of a complete graph and sylvester's "four point problem" of geometric probability. *American Mathematical Monthly*, 101:939–943, 1994.

[36] M. Scheucher. Counting Convex 5-Holes. Bachelor's thesis, Institute for Software Technology, Graz University of Technology, Austria, 2013.

[37] R. J. Vanderbei. Linear Programming: Foundations and Extensions, 1996.

[38] M. Zöhrer. *merged cape - Software Documentation*, 2008. Projektbericht, TU Graz.

[39] M. Zöhrer. *merged cape - User Manual*, 2008. Projektbericht, TU Graz.