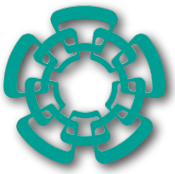


Center of research and advanced studies of
the national politechnic institute

CAMPUS ZACATENCO
DEPARTMENT OF MATHEMATICS

Clusters in Optimal Rectilinear Drawings of the Complete Graph: Insights into Potential Recursive Patterns

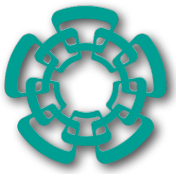
T H E S I S

Presented by
Juan Pablo Serrano Pérez

To obtain the degree of
Master of science
on the speciality of
mathematics

Thesis director: Dr. Ruy Fabila-Monroy

Mexico City. May, 2024



Centro de investigación y en estudios
avanzados del Instituto Politécnico
Nacional

UNIDAD ZACATENCO
DEPARTAMENTO DE MATEMATICAS

Clusters en dibujos rectilíneos óptimos de la gráfica completa: descubriendo posibles patrones recursivos

T E S I S

Que presenta
Juan Pablo Serrano Pérez

Para obtener el grado de
Maestro en ciencias
en la especialidad de
Matemáticas

Director de la tesis: Dr. Ruy Fabila-Monroy

Ciudad de México. Mayo, 2024

A mis padres, Rocío y Raúl

Agradecimientos

El presente trabajo no es fruto de una sola persona, es el reflejo de las virtudes que un grupo de personas han concedido para su realización. Este espacio es para agradecerles.

A mi mamá, Rocío Pérez, quien ha otorgado parte de su vida a formar la mía. Porque su amor y cariño llenaron en cuerpo y alma las raíces de mis sentimientos y convicciones.

A mi papá, Raúl Serrano, quien entregó su tiempo y esfuerzo a construir mi vida. Por ser el hombre por el cual me guiaré en mente y espíritu.

A mi hermano, Ariel Serrano, por acompañarme en cada camino en el que estoy. Porque su palabra y obra son un faro de luz cuando me hace falta.

A mi profesor, Ruy Fabila, por ser el eje central en mis estudios de maestría. Porque su humildad en la enseñanza me dejó conocimientos que siempre llevaré en obra y pensamiento.

Finalmente, agradezco al Centro de Investigación y de Estudios Avanzados y al Consejo Nacional de Humanidades Ciencias y Tecnología por sustentar y permitirme realizar mis estudios de posgrado. Esta tesis fue elaborada como parte del proyecto Ciencia de Frontera "Funciones y estructuras en gráficas y digráficas" (39570).

Deseo que este trabajo honrre el suyo

Juan Pablo

Contents

List of Figures

List of Tables

List of Algorithms

Resumen

El matemático húngaro Paul Turán, planteó el problema del número de cruces en una gráfica bipartita completa, que consiste en encontrar una representación en el plano que *minimice* los cruces entre las aristas. A pesar de su aparente simplicidad, este problema sigue sin resolverse. Numerosos estudios se han realizado en el área demostrando que es un problema NP-completo e incluso considerando la variante del *número de cruce rectilíneo* donde las aristas son dibujadas como segmentos rectos.

El presente trabajo está dedicado a aplicar un algoritmo de agrupamiento a dibujos que son *casi* óptimos de la gráfica completa K_n para diferentes valores de n. Los dibujos utilizados son el resultado de las investigaciones más recientes del área. Los dibujos empleados provienen de las investigaciones más recientes en el área. El algoritmo permite redescubrir subpatrones presentes en estos dibujos óptimos.

Abstract

The Hungarian mathematician Paul Turán proposed the problem of determining the number of crossings in a complete bipartite graph, which involves finding a planar representation that *minimizes* edge crossings. Despite its apparent simplicity, this problem remains unsolved. Numerous studies have been done and it has been shown to be an NP-complete problem, even when considering the *rectilinear crossing number* variant, where edges are drawn as straight line segments.

This work focuses on applying a clustering algorithm to nearly optimal drawings of the complete graph K_n for different values of n. The drawings used are the result of the most recent research in the field. The algorithm enables the rediscovery of sub-patterns present in these optimal drawings.

Chapter 1

Introduction

Even though scientific research often involves deep mental exercises, artificial intelligence has not only become integrated into industrial applications but has also made a significant impact on scientific exploration. For instance, it has been utilized in the study of structure-based modeling for protein drug discovery. In the field of informatics, artificial intelligence has been instrumental in developing Auto Machine Learning, which aims to generate algorithms in artifitial intelligence. Xu Jongjun et al. illustrate in [Xu+21] how artificial intelligence has been adopted not only in the pharmaceutical industry but also in various other scientific domains. We firmly believe that mathematical research will be no exception.

The primary objective of this work is to apply an unsupervised learning algorithm[1] to cluster rectilinear drawings of the complete graph achieving a small *crossing number*. The crossing number problem involves finding a layout of the complete graph such that the number of crossings between edges is minimized. Frank Harary and Anthony Hill conjectured a general formula for the rectilinear crossing number of K_n [HH63], but it remains unproven. We conclude Chapter 2 providing a replication of the NP-hardness proof by Michael Garey and David Johnson given in [GJ83].

A rectilinear drawing of a graph depicts the graph in the plane using straight line segments. The rectilinear crossing number of a graph is the minimum number of pairs of edges that cross in a rectilinear drawing of the graph. Unlike the crossing number, which refers to the minimum number of intersections in a graph drawing, there is currently no conjectured formula for the rectilinear crossing number of K_n. We utilize a clustering algorithm to potentially enhance our comprehension of the best rectilinear drawings that have been discovered [Aic+20]. An illustration of these patterns is exemplified in Figure 1.1.

Specifically, we employ a k-means algorithm, which identifies optimal k centers of a point set using a "minimum distance rule". This aspect is pivotal to our research as it determines the similarity between point sets. Given that the crossing number problem is fundamentally a geometric-combinatorial problem, we introduce in 1.4.3 a distance metric for a set of points in the plane. This metric draws inspiration from the field of combinatorial geometry, which explores geometric properties of sets with a finite number of points[2]. Based on the aforementioned discussion, this work is structured as follows: we begin with preliminaries covering graph theory and combinatorial geometry. This material sets the stage for Chapter 2, where we delve into the crossing number problem and discuss the advancements made since its inception. Finally,

[1] Machine learning may be divided in three principal subjects: supervised learning, unsupervised and reinforcement learning.

[2] In a broader context, this can encompass finite collections of various geometric entities.

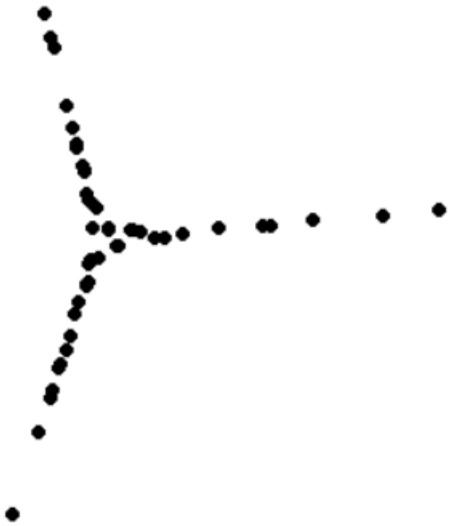


Figure 1.1: Best known drawing of K_{50}

in the last chapter, we focus on the k-means algorithm and its implementation, presenting the algorithms and providing general observations in the Results section.

1.1 The fundamentals of Graph Theory

Graphs are mathematical structures used to represent pairwise relations between objects of any kind. In this study, our focus lies on combinatorial and emergent problems within the realm of graph theory development. We will begin by introducing fundamental, yet formally defined concepts in graph theory.

A (simple) *graph* G is an ordered pair (V, E) of sets: V and E. The elements of the set E, are subsets containing two elements of V. There is no restriction on the cardinality of V but in this work, we will exclusively deal with finite sets V. The set V is referred to as the set of *vertices*, and E the set of *edges* of the graph G. Therefore, an element $v \in V$ is known as a *vertex* of G; and an element $e \in E$ will be termed as an *edge* of the graph G. Figure 1.2 illustrates a typical representation of a graph. In this depiction, two vertices, denoted as x and y are identified. These vertices establish the edge $\{x, y\} \in \binom{V}{2}$, which can also be written simply as xy. In cases where the vertices are self-evident, the edge may be referenced using a lowercase letter. The vertices x and y are termed the endpoints of the edge $xy \in E$. When two vertices share an edge, such as xy, they are considered adjacent or neighbors; otherwise, they are deemed *independent*. Similarly, an edge e is considered *incident* to a vertex x if x is one endpoint of e. In a graph $G = (V, E)$, each vertex $v \in V$, is assigned an integer value representing the number of edges incident to v. This number, denoted usually by $d(v)$, is termed as the *degree* of the vertex v. In Figure 1.2, for example, the vertex x has degree 2. The order of the graph G is the number $|V|$ representing the total number of vertices.

The *complete* graph in n vertices is a graph in which every two vertices are adjacent. The complete graph in n vertices is denoted by K_n.

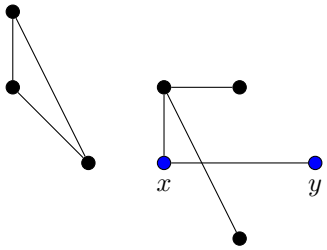


Figure 1.2: A graph.

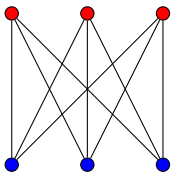


Figure 1.3: Drawing of $K_{3,3}$.

A *subgraph* of a graph $G = (V, E)$, is a subgraph $G' = (V', E')$ such that $V' \subset V$ and $E' \subset E$. Thus, every finite graph can be considered a subgraph of a well-known graph: the complete graph. A fundamental combinatorial property of finite graphs is the order of magnitude of the number of edges. Initially, we observe that the complete graph with n vertices, K_n, has exactly $\binom{n}{2}$ edges. Consequently, for a graph with n vertices, then the number of edges is $O(n^2)$. A *path* in a graph G is a subgraph P with its edge set $\{v_0v_1, v_1v_2 \ldots, v_{m-1}v_m\}$ and every vertex is visited exactly once. In this context, it is stated that the vertices v_0 and v_m are connected by P. A *cycle* in a graph G is a subgraph C with its edge set $\{v_0v_1, v_1v_2 \ldots, v_{m-1}v_0\}$ and every vertex is visited exactly once. A *connected* graph is one in which every pair of vertices are joined by a path. In the case where a graph is not connected, it can be decomposed into maximal connected subgraphs. Consequently, the graph becomes the disjoint union of these maximal connected subgraphs, known as the *connected components* of the graph. Figure 1.2 illustrates a graph with two connected components. Given a graph, where the set of vertices V can be partitioned into two independent sets, is denoted as *bipartite* graph. The bipartite graph with set of vertices $V = A \sqcup B$ where $|A| = m$, $|B| = n$ and every vertex of A is adjacent to every vertex of B is denoted as $K_{m,n}$ and is known as the *complete bipartite* graph on n and m vertices. See Figure 1.3 for an illustration. Lastly, a *multigraph* is defined as an ordered pair (V, E) consisting of a set V, representing the vertices, and a *multiset* E containing unordered pairs of vertices, known as multi-edges. In multigraphs, loop edges are allowed.

1.2 Graph drawings

In the following section, we present basic but fundamental results related to the focus of this work: the number of crossings that appear in a graph drawing.

When considering a graph, it is natural to envision it as a network of vertices connected by lines, as depicted in Figure 1.2. However, this perception aligns more with a topological point of view. For practical purposes, it is more convenient to define a graph in combinatorial terms, as we have done previously. To accommodate the network conception of a graph, we introduce the terms *drawing* or *embedding* of a graph. Just as T. Tucker [GT01] suggests, a *drawing* of a

graph is simply the process of placing a dot in the plane for every vertex and drawing lines to represent edges between the corresponding endpoints of the edge. The endpoints of the edge are in correspondence with the endpoints of the line representing the edge in the plane. A drawing is a correspondence between the set of vertices V and $\mathbb{R}^2$ and another correspondence between E and continuous curves in the plane:

$$V \to \mathbb{R}^2$$

and

$$E \to \{\text{continuous curves}\}.$$

Multiple drawings can be created to represent the same graph, and it is not a requirement that edges avoid crossing within their interiors. However, when a graph can be drawn without interior crossings between edges, it is referred to as a *planar* graph.

Lemma 1.2.1 (Euler's formula)**.** *Every planar connected graph with n vertices, m edges and $f-1$ bounded regions delimited by edges, in addition to the outer, infinite, and unbounded region, satisfies the equation*

$$n - m + f = 2.$$

Euler's formula 1.2.1 reveals an initial non-combinatorial property of (planar) graphs, stemming from the topological characteristics of graphs.

Proof. Let G be a planar graph with n vertices, m edges and f regions. We prove Lemma 1.2.1 by induction on the number of edges m. If $m = 0$ then G consists of a single vertex since it is connected, defining a single region, and the formula follows.

Now, suppose the formula holds for every connected graph with $m-1$ edges. We first consider the case where G has no cycles. In this scenario, as G is connected, it contains $n-1$ edges and the formula holds since G defines only a single region (the unbounded one). If G contains a cycle C, The cycle C divides the plane into two regions according to the Jordan curve theorem [Die17, Theorem 4.1.1 (Jordan Curve Theorem for Polygons) on p.91]. Consequently, by removing a single edge, these two regions will merge, decreasing the number of regions f by one while maintaining the number of vertices. We can apply the induction hypothesis, yielding

$$n - (m-1) + (f-1) = 2$$

which is nothing else than

$$n - m + f = 2.$$

□

From Euler's formula, we can observe that there exists a threshold for the existence of planar graphs, as the formula $n - m + f = 2$ represents a topological property of the plane. In other words, there must be a specific point where it becomes impossible to embed a graph in the plane without crossings. This concept is formalized in the following result.

Proposition 1.2.2. *The number of edges in a connected planar graph with n vertices is at most $3n - 6$. In other words, every connected planar graph has a linear number of edges.*

Proof. Let G a planar graph with n vertices, m edges, and f regions. Let's count the number of edges connected in the boundary of faces. A bounded face is formed by at least three edges, and an edge belongs to the boundary of its two faces. Thus, we have

$$m \geq \frac{3}{2} f.$$

By Euler's formula we know that $n - m + f = 2$, and so, $2 \leq n - m/3$. Therefore

$$m \leq 3n - 6.$$

□

The last two criteria allow us to explore the "first" non planar graphs.

Proposition 1.2.3. *There is no way to draw $K_{3,3}$ in the plane without crossings.*

Proof. Suppose $K_{3,3}$ can be embedded in the plane without crossings. Then Euler's formula tells us that $n - m + f = 2$ where $n = 2 \times 3 = 6$ and $m = 3^2 = 9$, and so, $f = 5$. Since $K_{3,3}$ is a bipartite graph, it does not contain cycles of length three[3] because in such a case, two independent vertices would have to be adjacent, which is not possible. Therefore, every bounded face is bounded by at least four edges. As counting faces implies counting each edge twice, we must have $f \leq 2m/4 = 9/2$ which contradicts $f = 5$. □

Proposition 1.2.4. *There is no way to draw K_5 in the plane without crossings.*

Proof. According to Proposition 1.2.2, a planar graph cannot have more than $3n - 6$ edges, and this limit does not hold for K_5 which has 5 vertices and $\binom{5}{2} = 10$ edges. □

The converse of Proposition 1.2.2 generally does not hold true. There exist graphs with the appropriate number m of edges that cannot be drawn in the plane without crossings. However, it is possible to determine in linear time whether a graph is planar or not with the algorithm of John Hopcroft and Robert Tarjan [HT74]. As we stated before, $K_{3,3}$ and K_5 seem to be the "starting" graphs where crossings begin. In the first half of the **20th** century, Kazimierz Kuratowski proved in [Kur30] necessary and sufficient conditions for a graph to be planar, such that certain specific "deformations" of neither $K_{3,3}$ nor K_5 can be subgraphs of a planar graph. We describe a simplified version of Kuratowski's theorem using graph subdivisions. A first approach for Kuratowski's theorem is as follows: since both K_5 and $K_{3,3}$ are non-planar graphs, adding edges to either of them will result in non-planar graphs. The precise meaning of "adding edges" is now explaining.

Definition 1.2.1 (Edge subdivision)**.** *A subdivision of an edge is the operation where the edge is replaced by a path of length 2.*

Definition 1.2.2. *A graph G is considered a subdivision of a graph H if G is obtained by a finite number of edge-subdivision operations.*

With this understanding of edge subdivisions, we can now state Kuratowski's theorem.

Theorem 1.2.5 (Kuratowski)**.** *A graph is planar if and only if it does not contain any subdivision of either K_5 or $K_{3,3}$.*

The full development of planar graphs, including the presentation and proof of Kuratowski's theorem, can be found in [Die17, Chapter 4 (Planar Graphs)].

[3] Actually a bipartite graph cannot contain odd cycles.

1.3 Notions of Combinatorial Geometry

As the last section illustrates, it is quite natural to visualize graphs as dots connected by curves in the plane. Moreover, Euler's formula (Lemma 1.2.1) demonstrates how topological properties emerge in graph theory results. Therefore, studying the geometric properties of a finite number of geometric objects and their relations can lead to insights in graph theory. This area of study is known as *Combinatorial Geometry*. In the following sections, we will introduce basic notions of Combinatorial Geometry. The primary objective is to define a combinatorial premetric on a finite set S of points in $\mathbb{R}^2$, which forms the foundational basis for the purpose of this work: an algorithmic approach to the *crossing number problem*. Following the exposition of Jiri Matoušek [Mat13], we delve into the realm of Combinatorial Geometry, starting with a fundamental concept.

Definition 1.3.1. *A set $C \subset \mathbb{R}^d$ is convex if for every pair of points, $x, y \in C$, the point $tx + (1-t)y$ belongs to C for every $t \in [0,1]$.*

The *convex hull* of a set $X \subset \mathbb{R}^d$ is the minimum convex set containing X. In Combinatorial Geometry, an important concept is the notion of an affine map, which plays a crucial role in understanding geometric transformations. An *affine map* from $\mathbb{R}^k \to \mathbb{R}^d$ is a map of the form $x \mapsto Ax + c$ where A is an $d \times k$ matrix and $c \in \mathbb{R}^d$ is a constant vector. Essentially, an affine map represents the composition of a linear transformation followed by a translation. Geometrically, it can be visualized as a transformation that preserves parallel lines but not necessarily distances and angles. The image under affine maps are referred to as $k-$flats or *affine subspaces*. An affine subspace of dimension $d-1$ is specifically known as an *hyperplane*.

Definition 1.3.2. *An arrangement $\mathscr{H}$ of a finite set of hyperplanes in $\mathbb{R}^d$ is a partition of $\mathbb{R}^d$ into relatively open convex sets, bounded by these hyperplanes. These sets are referred to as faces of the arrangement $\mathscr{H}$ and their dimensions ranges from 0 (vertices) to $d-1$ (cells).*

In the study of finite sets of points in $\mathbb{R}^d$, it becomes evident that topological notions alone are insufficient to capture their combinatorial properties. While any two finite sets in $\mathbb{R}^d$ may appear identical from a topological perspective (both being compact sets), their combinatorial properties can vastly differ. For instance, a set of three points can form a triangle or lie on a common line, illustrating that topological conceptions do not fully capture combinatorial aspects. The concept of general position for a finite set (or configuration) of points in $\mathbb{R}^d$ addresses this issue. General position implies that no "unlikely coincidences" occur within the configuration of points. In other words, when considering a configuration of points, the occurrence of coincidences is highly improbable. Just as Jiri Matoušek [Mat13] states:

> *"The precise meaning of general position is not fully standard; It may depend on the particular context."*

Definition 1.3.3 (General position)**.** *A finite set of points in $\mathbb{R}^d$ is said to be in general position if any choice of $k \leq d+1$ of them does not lie in a common $(k-2)$-flat.*

Indeed, general position can be interpreted as avoiding "unlikely coincidences" because $(k-2)$-flats are sets of measure 0 in $\mathbb{R}^d$. Therefore, in a probabilistic sense, the probability of randomly and independently choosing k points, with random distribution lying in a common $(k-2)$-flat is 0. See Figure 1.4b for an example illustrating points in general position, and Figure 1.4a for an example demonstrating points not in general position.

(a) Non general position. (b) General position.

Figure 1.4: Configuration of 3 points in $\mathbb{R}^2$.

1.4 Order types

In this section, our focus shifts to defining a suitable equivalence relation for configurations of points in $\mathbb{R}^d$ in general position. Although this work primarily concerns planar point sets, we introduce a general definition of order types as presented in Matoušek [Mat13, Section 9.3 (Order type)]. Given that there are infinitely many ways to arrange a set of n points in the plane, a natural question arises: when are two configurations combinatorially the same? In other words, how can we group configurations of points into equivalence classes such that each class represents a specific combinatorial configuration? To initiate this exploration, we turn to a natural generalization of the notion of 'order' in the ordinary real line $\mathbb{R}$. Recall that the real line is partitioned by the set of positive or negative numbers, along with the number 0, thereby associating each number with its sign. This concept can be extended to higher dimensions.

Definition 1.4.1. *[Orientation]*

1. *Let $p_1, \ldots, p_d$ be d vectors in $\mathbb{R}^d$. Consider Ω to be the matrix whose $i-$th column is p_i. The orientation of $p_1, \ldots, p_d$ is defined as the sign of $|\Omega|$.*
2. *The orientation of $d+1$ vectors $p_1, \ldots, p_d, p_{d+1}$ in $\mathbb{R}^d$ is the orientation of the d vectors $p_2 - p_1, \ldots p_d - p_1, p_{d+1} - p_1$.*

Now we can define what the order type of a configuration is.

Definition 1.4.2 (Order type)**.** *The order type of a configuration $\mathscr{C}$ of points in $\mathbb{R}^d$ in general position is determined by the orientations of every choice of $d+1$ points from $\mathscr{C}$. This assignment of orientations characterizes the relative arrangement of points in the configuration. Naturally, this concept is meaningful only when the cardinality of $\mathscr{C}$ is at least $d+1$, ensuring that enough points are available to establish orientations.*

Two configurations have the same order type if there exists a bijective correspondence between them such that the orientation is preserved. This means that the relative orientation of any chosen $d+1$ points in one configuration matches that of the corresponding $d+1$ points in the other configuration. See Figure 1.5 for an illustration.

In what follows, we shift our focus to points on the plane. The orientation $\mathscr{o}$ of three points $(a,b),(c,d),(e,f)$ in the plane can be computed using a small number of sums and multiplications:

$$\mathscr{o} = (c-a)(f-b) - (e-a)(d-b). \tag{1.1}$$

If $\mathscr{o}$ is positive, we say that the three points define a turn to the *left*; if it is negative, they define a turn to the *right*. We shall omit the case $\mathscr{o} = 0$ since, in most cases, we assume general position for a configuration $\mathscr{C}$.

Equation 1.1 provides a convenient method for computing the orientation of three points in the plane, with a computational complexity of $O(1)$. Furthermore, working with programming languages that allow arbitrary integer representations helps avoid numerical errors. Jacob Goodman and Richard Pollack, as detailed in their seminal work [GP83], pioneered the concept of

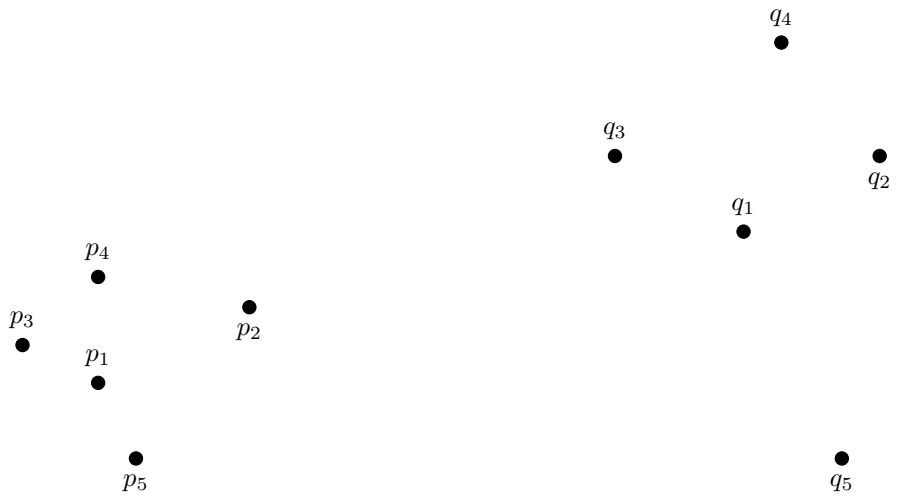


Figure 1.5: Configurations with the same order type.

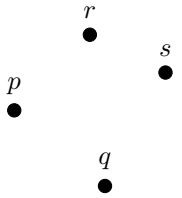


Figure 1.6: $\delta(p,q) = 0$ and $\delta(p,s) > \delta(p,r) + \delta(r,s)$.

order types as a foundational framework for implementing a "geometric sorting" process within $\mathbb{R}^d$. In their groundbreaking contribution, they devised a geometric sorting algorithm with a time complexity of $O(n^d \log n)$.

We conclude the introduction by defining our combinatorial δ-premetric.

Definition 1.4.3 (δ-premetric)**.** *Let $\mathscr{C}$ be a configuration of points in $\mathbb{R}^2$ in general position. Given two points p and q in $\mathscr{C}$, we define $\delta(p,q)$ as the number of pairs of points r and s in $\mathscr{C} \setminus \{p,q\}$ such that the line passing through r and s intersects the straight line passing through p and q.*

We refer to δ as a "premetric" because it does not satisfy all the properties of a metric or pseudo-metric. Figure 1.6 illustrates this with the four vertices of a quadrilateral, where neither the identity of indiscernibles nor the triangle inequality are satisfied.

Chapter 2

The crossing number

The crossing number of a graph is defined as the minimum number of crossings between edges occurring in a drawing of the graph in the plane. This problem, known as the crossing number problem, seeks to determine this minimum number.

The Hungarian mathematician Pál Turán encountered a specific instance of this problem during wartime. In 1944, while working at a brick factory transporting bricks on rails to storage yards, Turán faced difficulties when the crossing of two rails caused the bricks to fall out of the truck, requiring them to be reloaded. Hence, minimizing the number of crossings became crucial for minimizing time lost. Turán proposed this problem in a letter [Tur77] to the Journal of Graph Theory, where it became known as *"Turán's brick factory problem"*.

In defining the crossing number of a graph, it is stipulated that two edges cross at most once, and crossings at the endpoints of edges are disregarded. The crossing number of a graph G is denoted by $\mathrm{cr}(G)$. In a drawing of a graph, the edges are homeomorphic images of the unit interval $[0, 1]$, including the endpoints of the edge. A rectilinear drawing of G is a drawing in which the edges are drawn as straight line segments and its vertices are in general position. The rectilinear crossing number of G is the minimum number of crossings in every rectilinear drawing of G in the plane. The rectilinear crossing number of a simple graph G is denoted by $\overline{\mathrm{cr}}(G)$. It is worth noting that $\mathrm{cr}(G) \leq \overline{\mathrm{cr}}(G)$, as rectilinear drawings are a subset of all possible drawings. While the exact determination of the crossing number remains an open problem, Theorem 2.0.1 provides insight into the order of magnitude of the crossing number of a graph.

Theorem 2.0.1 (Crossing number lemma)**.** *The crossing number of a simple graph with n vertices and m edges is $\Omega\left(m^3/n^2\right)$.*

Proof. We follow the proof given in [Mat13]

Let $G = (V, E)$ be a simple graph with $n := |V|$ vertices and $m := |E|$ edges. A first lower bound for $\mathrm{cr}(G)$ is $m - 3n$ since if G had a drawing with fewer than $m - 3n$ edges, after deleting a single edge of each crossing, we would obtain a planar graph with more than $3n$ edges, which is impossible for planar graphs. Let r be the crossing number of a draw of G.

Consider choosing a random subset V' of V, in which, with probability p, we independently select a vertex $v \in V$ and include it in V'. The expectation of $|V'|$ is then

$$\mathbb{E}(|V'|) = np.$$

Now, let G' be the subgraph induced by V', and let E' be its set of edges. For an edge $e \in E$ to appear in G' we must choose both endpoints of e. Therefore, the expectation of E' is

$$\mathbb{E}(|E'|) = mp^2.$$

If r' is the crossing number of the induced drawing of G', then

$$\mathbb{E}(r') = rp^4$$

since for a crossing to appear in the drawing of G, four vertices of V must have been choosen. From the first lower bound, it follows that

$$rp^4 = \mathbb{E}(r') \geq mp^2 - 3np.$$

By choosing $p = 4n/m$, we have

$$r \geq \frac{1}{64} \cdot \frac{m^3}{n^2}.$$

Note that for p to be a probability it is required that $p \leq 1$, so we assume that $m \geq 4n$. Otherwise, the claimed bound would be negative. □

Once the war was over, Kazimierz Zarankiewicz [Zar55] proposed a solution for the *Turán's brick factory problem*. The solution by K. Zarankiewicz is resumed by the formula

$$\operatorname{cr}(K_{m,n}) = \left\lfloor \frac{m}{2} \right\rfloor \cdot \left\lfloor \frac{m-1}{2} \right\rfloor \cdot \left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lfloor \frac{n-1}{2} \right\rfloor. \tag{2.1}$$

The construction of K. Zarankiewicz for the drawing achieving (2.1) is quite straightforward. Let A, B be the partitions of $K_{m,n}$ with $|A| = m$ and $|B| = n$. If $m = 2k$, the points of A are placed along the $x-$axis with the following first coordinates:

$$-k, -(k-1), \ldots, -2, -1, 1, 2, \ldots, k-1, k.$$

If $m = 2k + 1$, they are placed as follows:

$$-k, -(k-1), \ldots, -2, -1, 1, 2, \ldots, k-1, k, k+1.$$

Similarly, the points of B are placed along the $y-$axis. Then, every point of A is connected with every point of B, achieving 2.1.

The Figure 2.1 illustrates an example of Zarankiewicz's construction for $m = 8$ and $n = 7$. The work of K. Zarankiewicz [Zar55] demonstrates that 2.1 holds when $n = 3$. However, an error in Zarankiewicz's demonstration in [Zar55] of equation 2.1 was discovered by Richard Guy [Guy68]. Equation (2.1) is now recognized as Zarankiewicz's conjecture and is believed to be optimal. Given that every graph is a subgraph of a complete graph, it is reasonable to initially attempt to compute $\overline{\operatorname{cr}}(K_n)$. Various constructions of drawings of K_n are known to yield a small number of crossings. One such construction is provided by Frank Harary and Anthony Hill in [HH63] for $n = 8$, which results in a *cylindrical drawing* of K_n. Additionally, Jaroslav Blazek and Milan Koman [BK64] devised a construction for each n that achieves the following upper bound:

$$\operatorname{cr}(K_n) \leq \frac{1}{4} \cdot \left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lfloor \frac{n-1}{2} \right\rfloor \cdot \left\lfloor \frac{n-2}{2} \right\rfloor \cdot \left\lfloor \frac{n-3}{2} \right\rfloor \tag{2.2}$$

Frank Harary and Anthony Hill [HH63] approached the task of determining $\overline{\operatorname{cr}}(K_n)$ as a novel problem. They also proposed the conjecture that $\operatorname{cr}(K_n)$ exactly matches the right-hand side of inequality 2.2, which is expressed in the following conjecture:

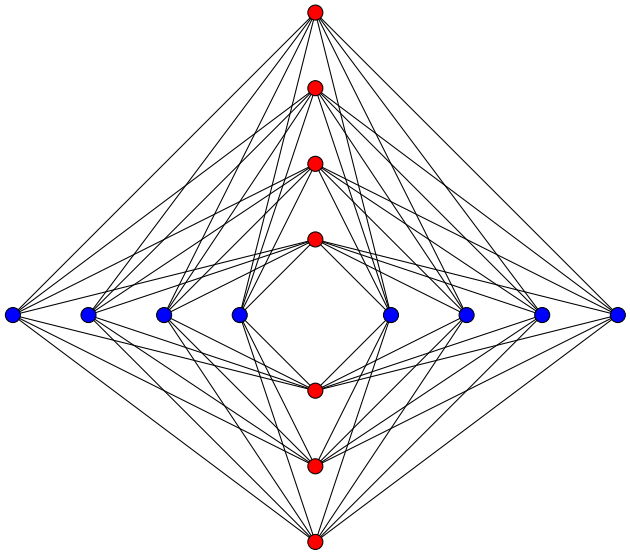


Figure 2.1: Zarankiewicz's draw of $K_{8,7}$.

Conjecture 2.0.2 (Harary and Hill)**.** *The exact value of the crossing number of the complete graph with n vertices, K_n, is given by*

$$\frac{1}{4} \cdot \left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lfloor \frac{n-1}{2} \right\rfloor \cdot \left\lfloor \frac{n-2}{2} \right\rfloor \cdot \left\lfloor \frac{n-3}{2} \right\rfloor \tag{2.3}$$

Tracing the developments in the crossing number problem, Richard Guy [Guy72] demonstrated that $\overline{\text{cr}}(K_5) = 1$ as an immediate consequence of Proposition 1.2.4, illustrated by a drawing of K_5 with a single crossing as depicted in Figure 2.2a.

In a clever manner, R. Guy [Guy72] presented a lower bound for $\text{cr}(K_m)$ involving $\text{cr}(K_n)$ as follows:

The complete graph K_m contains $\binom{m}{n}$ subgraphs K_n. Each crossing occurs due to the selection of four vertices, and therefore, it appears in $\binom{m-4}{n-4}$ such subgraphs K_n. Hence,

$$\text{cr}(K_m) \geq \frac{\binom{m}{n} \cdot \text{cr}(K_n)}{\binom{m-4}{n-4}}. \tag{2.4}$$

Let us consider the case where $m = n + 1$.

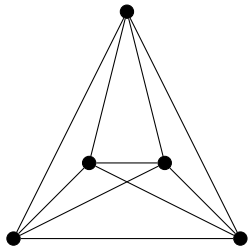
When n is an odd number, the crossing number conjecture (2.3) simplifies to the formula:

$$\frac{1}{4} \left(\frac{n-1}{2} \right)^2 \cdot \left(\frac{n-3}{2} \right)^2. \tag{2.5}$$

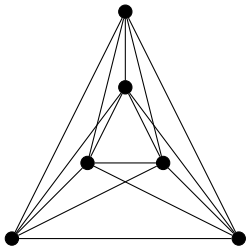
If we assume the conjecture 2.0.2 true when n is an odd number; and by applying the factor $(n+1)/(n-3)$ corresponding to the binomial coefficients in 2.4, expression 2.5 leads to

$$\text{cr}(K_{n+1}) \geq \frac{1}{64} \cdot (n+1)(n-1)^2(n-3) > \frac{1}{64} n(n-2)^2(n-4)$$

which matches with 2.3 for n even. Therefore, if the Harary-Hill conjecture is true for n odd, then it is true for $n + 1$.



(a) Optimal drawing of K_5.



(b) Optimal drawing of K_6.

Figure 2.2: Optimal drawings of the complete graph for $n = 5, 6$.

Thus, the Harary-Hill conjecture is true for $n = 6$ vertices since it is true for $n = 5$. Figure 2.2b shows an optimal drawing for six vertices with three crossings.

Given that $\binom{m}{n} \cdot \binom{n}{4} = \binom{m-4}{n-4} \cdot \binom{m}{4}$, inequality 2.4 demonstrates that the ratio $\mathrm{cr}(K_n)/\binom{n}{4}$ forms an increasing sequence bounded by 1. This is because each crossing results from the selection of 4 vertices, a condition satisfied by $\overline{\mathrm{cr}}(K_n)$.

Therefore, the limit

$$\lim_{n\to\infty} \frac{\overline{\mathrm{cr}}(K_n)}{\binom{n}{4}} \tag{2.6}$$

exists and is known as the *rectilinear crossing number constant* and will be denoted as q^*.

Richard Guy, in his work [Guy72], determined the crossing numbers $\mathrm{cr}(K_n)$ for n ranging from 6 to 10 by introducing the concept of a vertex's "responsibility," defined as the total number of crossings on all arcs incident with that vertex. Using this concept, he derived optimal drawings of K_7 from the optimal drawing of K_6 (see Figure 2.2b), demonstrating that there is only one optimal drawing of K_7 up to isomorphism. Applying this strategy, he obtained three non-isomorphic optimal drawings of K_8. However, R. Guy soon realized that his method was inadequate for larger values of n. Additionally, he proved that $\overline{\mathrm{cr}}(K_n) = \mathrm{cr}(K_n)$ for $n \leq 7$ and $n = 9$. Nevertheless, this pattern did not hold for K_8. None of the convex hulls of the three optimal drawings of K_8 were triangles, as stated in Theorem 2.0.3. Thus, $\overline{\mathrm{cr}}(K_8) > 18$, indicating that K_8 is a graph for which $\overline{\mathrm{cr}}(K_8) \neq \mathrm{cr}(K_8)$.

Theorem 2.0.3. *The convex hull of an optimal and rectilinear drawing of K_n is a triangle.*

Proof of Theorem 2.0.3 is complicated yet it has been elucidated by Oswin Aichholzer *et al.* in [Aic+07].

2.1 Crossing number complexity

In this section, we consider the decision problem related to the crossing number of a simple graph G: given an integer k, is $\mathrm{cr}(G) \leq k$? We refer to this problem as **CROSSING NUMBER** (CN). The related problem (concerned with this work) would be to consider the decision problem: given an integer k, is $\overline{\mathrm{cr}}(G) \leq k$?. This is the **RECTILINEAR CROSSING NUMBER** (RCN).

Firstly, in order to establish that CN belongs to the class of NP problems, we must be able to describe a certificate (a drawing of the graph) of the problem in polynomial size. This can be achieved using a concept known as *rotational systems*. Given a graph G drawn in the plane, a rotational system is a list $\pi = (\pi_v)_{v\in V}$, where π_v describes the cyclic ordering of the edges incident with the vertex v. However, this approach does not apply to RCN, as a rectilinear drawing of G would require an exponential number of bits to describe [GPS89].

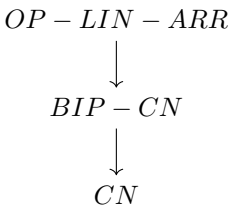


Figure 2.3: Diagram reduction for NP-completness proof.

We proceed with the NP-hardness proof of Michael Garey and David Johnson [GJ83] for the crossing number decision problem. The proof is facilitated by two reductions from two decision problems, as illustrated in diagram 2.3. The reduction problems are:

1. **OPTIMAL LINEAR ARRANGEMENT** (OP-LIN-ARR): Given a graph $G = (V, E)$ and an integer k, is there a one-to-one function $f : V \to \{1, 2, \ldots, |V|\}$ such that
$$\sum_{(u,v)\in E} |f(u) - f(v)| \leq k?$$

2. **BIPARTITE CROSSING NUMBER** (BIP-CN): Given a connected bipartite multigraph $G = (V_1, V_2, E)$ and an integer k, can G be embedded in a unit square so that all vertices of V_1 are on the northern boundary, all vertices in V_2 are on the southern boundary, all edges are within the square and there are at most k crossings?

OP-LIN-ARR has been proved to be NP-hard [Coh+06], hence, Lemma 2.1.1 demonstrates that BIP-CN is NP-hard.

Lemma 2.1.1. *OP-LIN-ARR reduces in polynomial time to BIP-CN.*

Proof. Let $\{G = (V, E), k\}$ be an instance of OP-LIN-ARR. Suppose that G is connected and let $V = \{v_1, \ldots, v_n\}$. We construct an instance $\{G', k'\}$ for BIP-CN in the following manner. Create a duplicate vertex v_i' for each $v_i \in V$, and join each v_i with the corresponding v_i'. Also, connect the vertex v_i with the vertex v_j' if $i < j$ and (v_i, v_j) is an edge of G. Set
$$k' := |E|^2(k - |E|) + (|E|^2 - 1).$$

It is clear that G' is bipartite with bipartitions V and $V' = \{v_i' \mid v_i \in V\}$ and is connected provided that G is connected. The construction of G' is in polynomial time. Let us proceed to proof that $\{G, k\}$ is satisfied if and only if $\{G', k'\}$ is satisfied. Suppose $\{G', k'\}$ is satisfied by a one-to-one function f. Place the vertices $v_i \in V$ the top edge of the unite square $[0, 1]^2$ and uniformly spaced. That is, place v_i at $(f(v_i)/n, 1)$. Place the vertex v_i' in a similar fashion at the bottom edge. See Figure 2.4. Each edge (v_i, v_j') will contribute $(|f(v_i) - f(v_j)| - 1)\,|E|^2$ crossings. Thus, the number of crossings will be at most
$$\sum_{(u,v)\in E} (|f(v_i) - f(v_j)| - 1)\,|E|^2 \leq (k - |E|)|E|^2 < k'.$$

Now assume that G' can be embedded into the unit square $[0, 1]^2$. From left to right, this embedding defines two one-to-one functions $f, f' : V \to \{1, 2, \ldots, |V|\}$ determined by the orderings

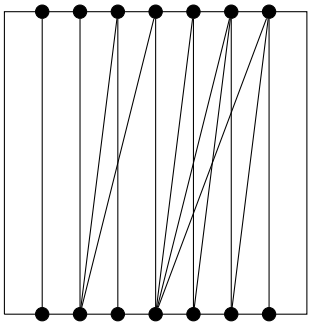


Figure 2.4: Embedding of G onto the unit square.

of V and V' respectively. We claim that $f = f'$. If $f(v_i) < f'(v_i)$ for some vertex v_i, by the pigenhole principle, there exists at least one vertex v_j such that $f(v_j) > f(v_i)$ and $f'(v_i) > f'(v_j)$. Therefore, there is at least one crossing of (v_i, v_i') with (v_j, v_j'). The same argument works for all $|E|^2$ edges (v_i, v_i'). Hence, at least $|E|^4$ crossings occur in the embedding which contradicts the hypothesis. So, the embedding resambles Figure 2.4. Thus, every edge (v_i, v_j') contributes with no less than $(|f(v_i) - f(v_j)| - 1) \cdot |E|^2$ crossings. Therefore

$$\sum_{(v_i,v_j)\in E} (|f(v_i) - f(v_j)| - 1) \cdot |E|^2 \leq k' < |E|^2(k - |E|).$$

□

Lemma 2.1.2. *BIP-CN reduces in polynomial time to CN*

Proof. The authors [GJ83] present a proof for another version of CN, specifically focusing on the crossing number of multigraphs. As part of their proof, they introduce a method involving the addition of a vertex of degree two at the midpoint of each multiedge.

Let $G = (V_1, V_2, E)$ represent an instance of BIP-CN. We construct a multigraph $G' = (V', E \cup E_1 \cup E_2 \cup E_3)$ as follows.

$$\begin{aligned} V' &= V_1 \cup V_2 \cup \{u, w\}. \\ E_1 &= \{3k+1 \text{ copies of the edge } (u, u') \mid u' \in V_1\}. \\ E_2 &= \{3k+1 \text{ copies of the edge } (w, w') \mid w' \in V_2\}. \\ E_3 &= \{3k+1 \text{ copies of the edge } (u, w)\}. \end{aligned} \tag{2.7}$$

If G can be embedded into the unit square with no more than k crossings, the desired drawing of G' can be achieved by placing u and w at the points $(1/2, 2)$ and $(1/2, -1)$ respectively, and joining the edges of E_3 outside the unit square. This will result in no more than k crossings.

Now, suppose that a drawing of G' exists with fewer than k crossings. The objective is to transform this drawing into one that resembles the structure defined by 2.7. We can assume that each pair of edges crosses at most once. We can always draw edges in a proper manner to avoid multiple crossings. Thus, each set of $3k + 1$ multiedges defines $3k$ bounded regions. The edges in E_1 create a series of bounded regions and one region that is unbounded. If w is inside one of the bounded regions, transform the edges that define the bounded area to remove w from the bounded region. Consequently, every vertex of V_2 is in the interior of the unbounded region as well. This is because every vertex w' is incident to w, $3k + 1$ times, and so, if w' belongs to a bounded area, it will contribute at least $3k + 1$ crossings. Let us consider two observations:

1. There are $3k$ bounded areas defined by the $3k+1$ copies of the edge (u', u). In the middle k regions, there cannot be a vertex $u'' \in V_1$.

Indeed, since G is connected, if u'' were in one of the middle k regions, it would have an edge connecting it to a vertex in V_2, and this edge would necessarily have at least $k+1$ crossings.

2. There cannot be an edge crossing any of the k middle regions because doing so would result in an increase in the number of crossings by more than k. This is because, based on the first observation, the endpoints of such an edge must lie in the first or last k regions.

We achieve the desired embedding with the exception of the original edges in E and the $3k+1$ edges between u and w. There are $3k$ bounded regions delimited by the $3k+1$ multiedges between u and w. We will prove that all vertices of V_1 and V_2, besides all the edges of E, lie in the same bounded region. Let R_0 denote the unbounded region of the multiedges between u and w, and let R_j, $j = 1, \ldots, 3k$, denote the other $3k$ bounded regions. For each $j \in \{1, \ldots, 3k\}$, if a is a vertex contained in the region R_j, then there is no vertex b in any of the regions $R_{j+k+1 \mod 3k+1}, \ldots, R_{j+2k \mod 3k+1}$. This is because if such a vertex b existed, there would be a path joining a with b (since G' is connected), and this path would cross at least $k+1$ edges of those (u, w). Similarly, there cannot be edges going through any of the k regions $R_j, R_{j+k+1 \mod 3k+1}, \ldots, R_{j+2k \mod 3k+1}$. Therefore, every edge (u, w) can be moved in a single empty area without increasing the number of crossings. This transformation of the $3k+1$ edges (u, v) leaves all of V_1, V_2 and E into a single ubounded area. Finally all G can be sent into a single bounded region. □

Corollary 2.1.3. *CN is NP-hard.*

We conclude this section with M. Garey and D. Johnson's [GJ83] quote .

> *"Future research into crossing numbers will be justified in focusing on inexact methods that only estimate crossing numbers, and the quest for exact values of cr(G) will have to be restricted to promising special cases."*

2.2 On the drawings with small crossing number

The rectilinear crossing number constant, denoted by q^* (see 2.6), quantifies the rectilinear crossing number of the complete graph K_n as a factor of $\binom{n}{4}$. It provides valuable insight into identifying drawings with minimal crossing numbers. Several approaches aimed at reducing upper bounds on the crossing number emerged in the years following the initial publications on the topic. However, the rectilinear crossing number $\overline{\text{cr}}(K_n)$ was approached as a new problem by F. Harary and A. Hill [HH63]. The hope that $\text{cr}(K_n)$ and $\overline{\text{cr}}(K_n)$ coincide was shattered by R. Guy, who proved that this does not hold for $n = 8$. Subsequently, efforts were made to derive improved upper bounds for $\overline{\text{cr}}(K_n)$. H. Jensen [Jen71] presented a general construction achieving $q^* < 0.3888$, while around the same time, D. Singer [Sin71] claimed $q^* < 0.38462$. Since then, further improvements have been achieved. Bernardo Abrego and Silvia Fernández-Merchant presented a construction with a smaller crossing number, establishing an upper bound of $q^* < 0.380559$ [ÁF07].

2.2.1 Heuristics

In the following, we outline some heuristics presented in [Aic+20] that have been employed to enhance the rectilinear crossing number. Recently, Fabila and López in [FL14] introduced an

$O(n^2)$ time algorithm to compute $\overline{\text{cr}}(S)$, where S is a set of n points in the plane in general position. This algorithm has led to significant improvements in the best-known crossing number records for $n \leq 100$. The heuristic presented by Fabila and López is as follows.

1. Take randomly a point $p \in S$.
2. Select randomly a point q "near" p.
3. If $\overline{\text{cr}}(S \setminus \{p\} \cup q)$ improves $\overline{\text{cr}}(S)$, update $S := S \setminus \{p\} \cup q$.
4. Go to step 1.

To select the point q to replace p, they generate two natural numbers (t, r) with an exponential distribution with a pre-specified mean and round it to the nearest integer. Then, q is set as $p + (t, r)$. If no improvement is found, the mean in the exponential distribution is halved. The selection of q can be enhanced in the following way: consider $\mathscr{A}$, the line arrangement spanned by the lines passing through two points of $S \setminus \{p\}$. Compute the cell C of $\mathscr{A}$ that contains p and produce a sequence of consecutive cells $C, C_2, \ldots, C_m$. Then, pick a point q_i from each C_i as a possible replacement candidate for p. The advantage of this approach is that $\overline{\text{cr}}(S \setminus \{p\} \cup q_i)$ can be computed more quickly because the points q_i belong to adjacent cells. Hidalgo implemented an $O(n^2 \log n)$ time algorithm in his Master's thesis [Hid15] to compute C. The following heuristic builds upon the observation that drawings of K_m with few crossings frequently include drawings of K_n with similarly few crossings for $n < m$. By refining the drawings of K_m using the aforementioned heuristics, it becomes possible to derive drawings of K_n with reduced crossings. One significant advantage of this strategy is that it allows exploration across the $\binom{m}{n}$ subsets of n points within a set of m points, potentially leading to improvements in crossing numbers across this spectrum. In [DF17], Duque and Fabila demonstrate that computing the set $\{\overline{\text{cr}}(S \setminus \{p\}) \mid p \in S\}$ can be achieved in $O(n^2)$ time. Similarly, they show that computing the set $\{\overline{\text{cr}}(S \cup \{q\}) \mid q \in S\}$ can also be done in $O(n^2)$ running time. Leveraging these results, the process of removing the point p to minimize $\overline{\text{cr}}(S \setminus \{p\})$ can be executed in polynomial time, along with finding an optimal drawing of K_{m-1}.

To conclude this chapter, we showcase the predominant pattern observed in optimal drawings resulting from the application of the aforementioned heuristics. Refer to Figures 2.5, 2.6, 2.7, and 2.8. This recurring pattern serves as the primary motivation for considering machine learning techniques, aiming to explore and gain a deeper understanding of these patterns. As established by Theorem 2.0.3, if a drawing of a graph is optimal, it is necessary that its convex hull defines a triangle. The heuristics presented earlier yield the following best-known rectilinear drawings of K_n. Noticeably, all the best-known rectilinear drawings of K_n consist of three "arms".

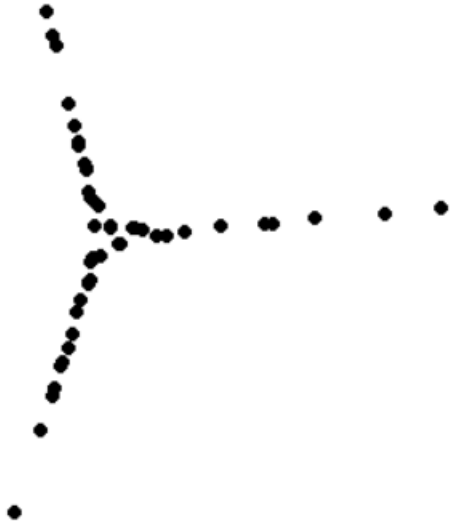


Figure 2.5: Best known drawing of K_{50}

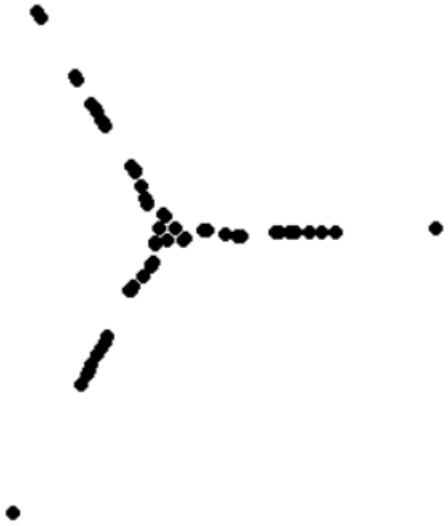


Figure 2.6: Best known drawing drawing of K_{75}

Figure 2.7: Best known drawing drawing of K_{100}

Figure 2.8: Best known drawing drawing of K_{1000}

Chapter 3

Clustering optimal drawings

Cluster analysis is the process of categorizing a dataset into groups, or clusters, based on the degree of similarity among its elements. The notion of "similarity" is context-dependent and varies depending on the problem under consideration. Consequently, there is no universally agreed-upon definition of clustering, and different clustering algorithms are tailored to suit specific research domains. However, a widely accepted and intuitive concept is outlined in [DY15]:

1. The instances must be as similar as possible.
2. Different clusters must contain elements as different as possible.
3. The measure taken into account to perform the cluster analysis must have a practical meaning on the particular problem that is being studied.

The third point outlined above is known as "dissimilarity," denoted by δ, and forms the central concept of cluster analysis as it encapsulates the relationships within the dataset. Xu Dongkuan and Tian Yingjie provide a comprehensive overview of both traditional and modern clustering algorithms in their chapter on clustering [DY15].

3.1 k-means clustering

Our discussion largely aligns with the treatment provided by Thomas H. Cormen *et al.* in their work on clustering, as presented in [Cor+22]. The initial step involves formally defining what constitutes a cluster.

Definition 3.1.1 ($k-$means clustering)**.** *A k-clustering of a finite set $S \subset \mathbb{R}^d$ is defined as a sequence of k subsets $S_1, \ldots, S_k$ satisfying the condition:*

$$S = S_1 \sqcup S_2 \sqcup \cdots \sqcup S_k$$

where $\sqcup$ denotes disjoint union.

A k-clustering is constructed using a sequence of centroids (or centers) $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$ such that $\boldsymbol{c}_\ell \in S_\ell$ and every point of S_ℓ is closer (via δ) to the centroid $\boldsymbol{c}_\ell$ than to any other center $\boldsymbol{c}_i$. This means:

$$\boldsymbol{x} \in S_\ell \text{ if and only if } \delta(\boldsymbol{x}, \boldsymbol{c}_\ell) = \min_{1 \le i \le k} \delta(\boldsymbol{x}, \boldsymbol{c}_i) \tag{3.1}$$

A tie may comes up but it will be broken arbitrarily. In essence, the algorithm iteratively updates centroids, gradually improving their positions until a convergence criterion is met. This emphasis on clustering leads to the k-means problem: given a finite set S and an integer k, the goal is to find a sequence of k centers $c_1, \ldots, c_k$ that minimizes the following expression:

$$\sum_{x \in S} \min_{1 \leq i \leq k} \delta(x, c_i). \tag{3.2}$$

Initially, the centers $c_1, \ldots, c_k$ are permitted to be any points in the space and need not necessarily belong to S. However, in our approach, we will stipulate that the centers must belong to S in order to compute our combinatorial premetric. The utilization of centers not belonging to the underlying set is predominantly considered in Euclidean clustering, where the distance is typically measured using the Euclidean distance metric. When the centers are fixed, one can construct the cluster S_ℓ using the points that satisfy

$$\delta(x, c_\ell) = \min_{1 \leq i \leq k} \delta(x, c_i)$$

and hence, equation (3.2) can be expressed as

$$\sum_{\ell=1}^{k} \sum_{x \in S_\ell} \delta(x, c_\ell).$$

Thus, to compute centroids in the subset S_ℓ, our criterion will be the point $c_\ell \in S_\ell$ that minimizes

$$\sum_{x \in S_\ell} \delta(x, c_\ell). \tag{3.3}$$

The k-means problem gives rise to a natural decision problem: Given $R > 0$ and $S \subset \mathbb{R}^d$, is there a set of k centers $c_1, \ldots, c_k$ such that

$$\sum_{x \in S} \min_{1 \leq i \leq k} \delta(x, c_i) \leq R?$$

When δ is taken as the square of the Euclidean distance, Meena Mahajan *et al.* in [MNV12] prove the NP-hardness of the k-means problem, making it NP-complete. Their proof focuses on a reduction from PLANAR-3SAT. Within this research, we opt for a traditional clustering algorithm: Lloyd's algorithm.

3.2 Order type clustering

We present a randomized version of Lloyd's algorithm as described in [Cor+22]. As mentioned earlier, Lloyd's procedure is iterative.

Input. A set S of vectors in $\mathbb{R}^2$ and the number k of desired clusters.

Output. A sequence of clusters $S_1, \ldots, S_k$ along with their corresponding k centers $c_1, \ldots, c_k$.

Procedure

1. (Initialize centers) Choose k different centers $c_1, \ldots, c_k$ at random and independently from S.

2. (Points Assignment) For each point $x \in S$, compute the nearest center c_ℓ and assign x to cluster S_ℓ. This step creates the clusters based on the nearest center.

3. If the assignment of elements to their clusters remains unchanged in Step 2, the algorithm terminates and returns the clusters $S_1, \ldots, S_k$ along with their respective centers $c_1, \ldots, c_k$.

4. (Compute centers) After assignment, compute the center of S_ℓ and then return to step 2.

By the convergence criteria, Lloyd's procedure always terminates, as each iteration (except for the last one) decreases the value of

$$\mu := \sum_{\ell=1}^{k} \sum_{x \in S_\ell} \delta(x, c_\ell).$$

and there are only $O(k^n)$ choices of k clusters. Therefore, there comes a point where μ will no longer decrease, prompting the algorithm to terminate. Lloyd's algorithm exemplifies a common approach in machine learning techniques:

First, define a sequence of parameters τ and associate it with some hypothesis $\mathscr{H}_\tau$. Second, consider a value $\mu(\mathscr{E}, \tau)$ that measures how poorly the hypothesis $\mathscr{H}_\tau$ fits to some *training* data $\mathscr{E}$. Finally, apply an optimization procedure to find τ^* that minimizes (locally) $\mu(\mathscr{E}, \tau)$. In the k-means problem (in the plane), our parameters consist of the selection of k centers. Thus, τ is a vector of dimension $2k$, $\mathscr{H}_\tau$ represents the hypothesis that x is grouped with the cluster having its nearest center, and $\mu(\mathscr{E}, \tau)$ is represented by our value μ (refer to Equation 3.6).

In this work, we consider a variant of Lloyd's algorithm where the Euclidean distance is replaced by our combinatorial δ-premetric defined in Definition 1.4.3. Roughly speaking, when applying Lloyd's algorithm, the expectation is to find groups where points within the same group are more similar to each other than to points in different groups. If similarity is measured in a combinatorial manner, clusters with combinatorially similar points could be obtained. This is referred to as *order type clustering*. As mentioned, Lloyd's algorithm is guaranteed to terminate after a finite number of iterations since there are only up to k^n possible cluster configurations. However, an algorithm with exponential running time is not practical. Therefore, in practice and throughout this investigation, we will terminate the procedure after a certain number of iterations. Since Lloyd's algorithm finds only locally optimal clustering, a near-optimal solution will suffice for our purposes. It is worth noting that a brute force computation of δ yields with an $O(n^4)$ running time algorithm; however, we will improve upon this later.

3.3 Implementation

To start, our input consists of a set S of points in the plane in general position and a desired number k of clusters. We begin by computing our δ-premetric. A brute-force approach involves exploring every pair of points $\{u, v\}$ and determining if the line passing through u and v will cross the line segment defined by each pair of points $\{p, q\}$. This procedure has a time complexity of $O(n^4)$ to compute all distances. However, let us analyze the δ-distance between the points p and q. First, we illustrate in Theorem 3.3.1 how to extract those points r that form one pair (r, x) contributing to the δ-distance for a fixed point $x \in S$. We recall that the *symmetric* point of a point y with respect to a point x is the antipodal point of the circle with center x and radius $\| y - x \|$. Actually, the symmetric point of y with respect to x can be computed as

$$2x - y.$$

Theorem 3.3.1. *Let S be a finite set of points in general position in $\mathbb{R}^2$, and consider the points $p, q, x \in S$. Let ℓ_{px} and ℓ_{qx} be the lines passing through p and x, and q and x respectively.*

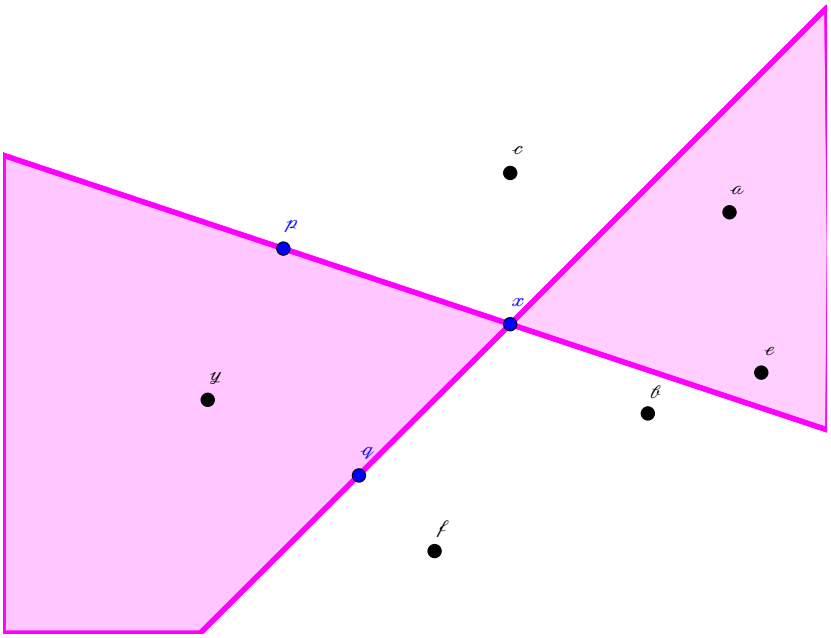


Figure 3.1: Way to count the contribution of x to $\delta(p, q)$.

Consider the cell $\mathscr{A}$ that contains the segment of line with endpoints p and q, and let $\mathscr{B}$ be the symmetric points of $\mathscr{A}$ with respect to x. Then, the point x contributes

$$|S \cap \mathscr{A}| + |S \cap \mathscr{B}| \tag{3.4}$$

to $\delta(p, q)$.

Figure 3.1 provides a visual representation of Theorem 3.3.1.

Proof. At each point $y \in \mathscr{A}$, the line ℓ_{yx} is guaranteed to intersect the line segment defined by points p and q. Let $y \in \mathscr{A}$. For our purposes, let us assume that the convex hull of the points p, q, x, y forms a quadrilateral with vertices p, q, x, y. In this case, one diagonal of the quadrilateral coincides with the line segment between p and q, while the other diagonal corresponds to the line segment defined by ℓ_{yx}. These two diagonals intersect at one point. If the convex hull is not a quadrilateral but a triangle, then replacing y with any point on ℓ_{yx} will form the quadrilateral, ensuring the existence of the crossing. Since a point $z \in \mathscr{B}$ corresponds to its symmetric point y in $\mathscr{A}$ and they both lie on the common line ℓ_{zx}, a crossing is guaranteed. Therefore, for a point $r \in (S \cap \mathscr{A}) \cup (S \cap \mathscr{B})$, the pair (r, x) contributes to $\delta(p, q)$. □

Theorem 3.3.1 outlines a procedure for computing $\delta(p, q)$. We need to identify the points within regions $\mathscr{A}$ and $\mathscr{B}$. To accomplish this, our primary tool will be *polar sorting*. Polar sorting involves sorting a set of points in the plane based on their angles relative to a given point as the center. We recall that sorting can be performed in $O(n \log n)$ time complexity.

Theorem 3.3.2. *Let S be a finite set of points in general position in $\mathbb{R}^2$. Let p and q denote two arbitrary points in S.*

For each point $x \in S$, define the set

$$S_x := S \cup \{\text{symmetric points of } S \setminus \{x\} \text{ with respect to } x\}.$$

Consider the polar sorting array $\mathscr{X}$ of S_x with x as the center. Let τ denote the orientation of the three points x, p, q (in that order). Finally, let i and j represent the indices where p and q are located in $\mathscr{A}$, respectively. Consider α_x to be one of the following values.

1. *If τ represents a left turn and $i < j$, then α_x equals the number of points between p and q in the array $\mathscr{A}$.*
2. *If τ is a left and $i > j$, α equals to the number of points after p and before q in the array $\mathscr{A}$.*
3. *If τ is a right and $j < i$, α equals to the number of points between q and p in the array $\mathscr{A}$.*
4. *If τ is a right and $j > i$, α equals to the number of points after q and before p in the array $\mathscr{A}$.*

Iterating step 1 to 4 over all points $x \in S$, then

$$\delta(p, q) = \sum_{x \in S} \left\lfloor \frac{\alpha_x}{2} \right\rfloor. \tag{3.5}$$

Proof. Let $S, p, q, x, \mathscr{X}$ and S_x, be as defined in the theorem. We first note that points 3 and 4 are symmetric to points 1 and 2 because a right turn τ in the sequence x, p, q corresponds to a left turn in the sequence x, q, p. Therefore, we only need to demonstrate the relationship for points 1 and 2. Hence, τ represents a left turn (counterclockwise). Let i, j be the indexes where p and q are placed in $\mathscr{X}$ respectively. Firstly, consider the case where $i < j$. This situation resembles Figure 3.2. In Figure 3.2, the red points represent the symmetric points with respect to x. When sorting, points with angles greater than p will appear after index i in the array $\mathscr{X}$, and points with angles smaller than q will appear before index j in $\mathscr{X}$. Consequently, the points of S within the defined region $\mathscr{A}$ (as described in Theorem 3.3.1) are accounted for. The process of taking symmetric points involves flipping the angle by 180°. Therefore, points from $\mathscr{B} \cap S$ (the region $\mathscr{B}$ defined in Theorem 3.3.1) will appear in the region $\mathscr{A}$ between indices i and j, as stated earlier. Thus, we have counted all the points that contribute to $\delta(p, q)$ with respect to x.

Point 3 can be considered the "symmetric" of point 1, and therefore, it follows the same proof. Hence, we can accept that point 3 is true as well. Let's prove point 2. Since point 4 is the "symmetric" counterpart of point 2, the theorem will be proved after demonstrating point 2.

If $j < i$, then every point occurring after p and before q belongs to $\mathscr{B} \cap S$, and those belonging to $\mathscr{A} \cap S$ are accounted for by their symmetric points in $\mathscr{B}$. However, we are counting twice because when we repeat the last process with another point, for example, with point a as the center, we will once again count the pair (a, x). Thus, we must take $\lfloor \alpha_x/2 \rfloor$.

□

3.3.1 Algorithms

Theorem 3.3.2 presents an algorithm that can be implemented in any programming language. However, to execute this algorithm, we require a subroutine to compute the sorted array $\mathscr{X}$. For

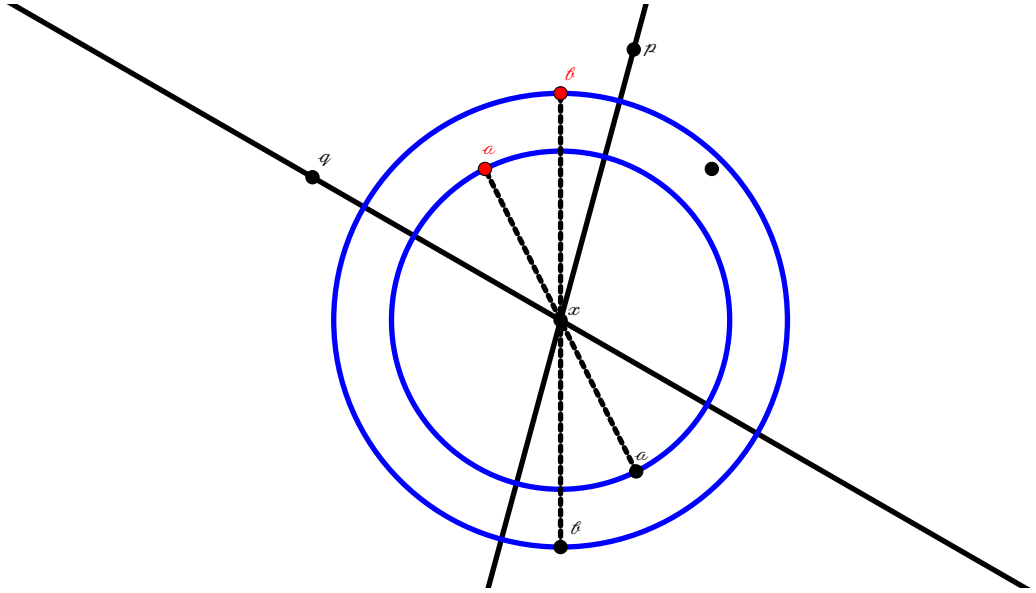


Figure 3.2: Point 1 of Theorem 3.3.2

this purpose, we will assume the existence of a procedure **SORT-AROUND-BY** that operates in $O(n \log n)$ time complexity to sort the points based on their angles around a given center point x. Procedure 1 takes as input the set of n points S and returns n lists of points, each sorted (with the symmetric points included) with respect to a specific point x as the center. In lines 3 and 4, empty sets are initialized to store the sorted points in $\mathscr{A}_x$, and $\mathscr{B}$ stores the symmetric points with respect to x.

Algorithm 1: SORT-POINTS

```
   Data: S
   Result: List of ordered lists
1  Initialize 𝒜 = ∅;
2  for x ∈ S do
3      Initialize 𝒜_x = ∅;
4      Initialize ℬ = ∅;
5      for p ∈ S \ {x} do
6          ℬ = ℬ ∪ {2x − p};
7      𝒜_x = 𝒜_x ∪ ℬ;
8      𝒜_x = SORT-AROUND-BY(𝒜_x);
9      𝒜 = 𝒜 ∪ 𝒜_x;
10 return 𝒜;
```

An asymptotic analysis on the running time of Algorithm 1 is as follows. The running time of line 8 depends on the input of **SORT-AROUND-BY**. In this case, L_x is a set of $2n-1 = O(n)$ points. Therefore, line 8 is $O(n \log n)$. The **For** loop from line 5 to 6 has a time complexity of $O(n)$ since the calculation of $2x - p$ is performed in $O(n)$. Any other line within the **For** loop from line 2 to line 9 will not have a time complexity greater than $O(n \log n)$. Thus, for each point

x in S, the **For** loop will have a time complexity of $O(n \log n)$. Overall, the **SORT-POINTS** algorithm has a time complexity of $O(n^2 \log n)$. Using the **SORT-POINTS** procedure, we can compute $\delta(p, q)$ for points p and q. The input of Algorithm **DELTA** (Algorithm 2) includes the set of points S, two points p and q[1], and $\mathscr{A}$, which refers to the returned set of **SORT-POINTS**. We assume that the calculation of $\mathscr{A}$ has been done as part of the preprocessing when computing $\delta(p, q)$. The return value of Algorithm **DELTA** is the value of $\delta(p, q)$.

Algorithm 2: DELTA

```
   Data: (S, p, q, 𝒜)
   Result: δ(p, q)
 1 Initialize δ = 0.;
 2 for x ∈ S \ {p, q} do
 3     Set M = 𝒜_x;
 4     Find the indexes i, j corresponding to p and q in 𝒜_x respectively;
 5     Compute the turn τ of the points (x, p, q) (in that order);
 6     if τ is a turn to the left then
 7         if i < j then
 8             δ = δ + (j − i − 1);
 9         else
10             δ = δ + (j − 1) + (n − i);
11     if τ is a turn to the right then
12         if i > j then
13             δ = δ + (i − j − 1);
14         else
15             δ = δ + (i − 1) + (n − j);
16 return ⌈δ/2⌉;
```

As mentioned previously, δ takes $\mathscr{A}$ as input to avoid multiple calculations of the same value $\mathscr{A}$ and to allow access to it in constant time. At line 3 of Algorithm 2, we assume that $\mathscr{A}_x$ can be accessed in $O(1)$ time. This can be achieved in various programming languages. For instance, in Python, which is the programming language we used, $\mathscr{A}$ may be represented by a dictionary with x as the key, allowing constant time access to the value associated with the key x. For line 4, several methods to find elements in an array are known. The most common method runs in linear time with respect to the size of the array. However, we can take advantage of the fact that $\mathscr{A}x$ is sorted. Therefore, we can find i and j in $O(\log n)$ time, as presented in Algorithm 3. The remaining part of the **For** loop runs in $O(1)$ time. It is important to note that the value of $(j - i - 1)$, for example, in line 8, corresponds to the number of elements between i and j in $\mathscr{A}x$. Thus, each iteration of the **For** loop takes $O(\log n)$ time. Finally, Algorithm 2 computes $\delta(p, q)$ in $O(n \log n)$. Therefore, to compute all $\delta(p, q)$ will be $O(n^3 \log n)$. This is much better than $O(n^4)$ time with the force brute approach. To perform a binary search to find an element p in a sorted array $\mathscr{A}$, our decision on whether to move to the left or to the right will be based on our geometric primitive: the orientation.

Algorithm 3 takes as input a sorted array $\mathscr{A}$ of length n containing a point p that we want to find. The output is the index where p is located in $\mathscr{A}$. We denote by $\mathscr{A}[k]$ the element of $\mathscr{A}$ at position k. We omit the time analysis since it follows the classic binary search algorithm.

[1]We may assume that the given points belong to S. Otherwise, we return a distance of 0.

Algorithm 3: BINARY-POLAR-SEARCH

Data: $(\mathscr{A}, p)$
Result: The index where p is placed

```
Set n = |𝒜|;
Set L = 0;                                              // L for left
Set R = n − 1;                                          // R for right

while L ≤ R do
    Set M = ⌊(L+R)/2⌋;                                  // M for middle
    if 𝒜[M] = p then
        return M;
    else
        Compute the turn τ of the points (𝒜[0], p, 𝒜[M]) (in that order);
        if τ is a turn to the right then
            R = M − 1;                                  // p is placed before the middle
        else
            L = M + 1;                                  // p is placed after the middle
```

Recall that Lloyd's procedure assigns points of S to their nearest center and then recomputes the center of S_ℓ, repeating this process until there is no improvement in the value.

$$\mu := \sum_{\ell=1}^{k} \sum_{x \in S_\ell} \delta(x, c_\ell). \tag{3.6}$$

How to select centers in order to improve μ?

Algorithm 4: CENTROID

Data: (S_ℓ)
Result: Centroid of S based on δ

```
Set 𝒜 =SORT-POINTS(S_ℓ); // Order the points with respect to the cluster
Set minDist = ∞;
Set c =NULL;                          // To store the centers at the variable c.
for x ∈ S_ℓ do
    Set distX = 0;
    for p ∈ S_ℓ do
        Set d =DELTA(S_ℓ, p, x, 𝒜);
        distX = distX + d;
    if distX < minDist then
        minDist = distX;
        c = x;                  // We assign x as the new center and store it to c.
return c
```

Naturally, if we minimize the expression

$$\sum_{x \in S_\ell} \delta(x, c_\ell) \tag{3.7}$$

we aim to achieve a small value of μ. Therefore, the *centroid* rule to compute centroids in a cluster S_ℓ will be the point c that minimizes

$$\sum_{x \in S_\ell} \delta(x, c).$$

In a classic clustering algorithm where Euclidean distance is used, the centroid acts as the *center of mass* of the cluster S_ℓ, and it is equivalent to the arithmetic mean. However, the disadvantage of using this point as our centroid is that it may not belong to S_ℓ, which poses a computational issue because our δ-premetric is defined on the global set S. In light of this, we consider equation (3.7) as a combinatorial way to compute centroids[2]. Algorithm 4, **CENTROID**, takes as input the cluster S_ℓ and returns the centroid c computed based on equation (3.7). Line 1 has a time complexity of $O(m \log m)$, where m is the size of the cluster S_ℓ. If we assume that $m = \Omega(n)$ (which is a reasonable assumption if the number of clusters is small compared to n), then line 1 has a time complexity of $O(n \log n)$. The **For** loop from lines 4 to 10 has a time complexity of $O(n^3 \log n)$ due to the time taken by **DELTA** and the two nested loops. Thus, **CENTROID** takes $O(n^3 \log n)$ time to compute the centroid c. We finally are able to present our **CLUSTER-ORDER-TYPE** Algorithm.

Let us analyze Algorithm 5 **CLUSTERING-ORDER-TYPE**. At line 2, we assume the existence of a method called **RANDOM** to randomly select k different points from S. This assumption is reasonable since most programming languages provide libraries for pseudo-random number generation. The **While** loop at line 4 continues until the condition $\mathscr{C} = UC$ at line 19 is satisfied. This condition indicates that no improvement in the centroid has been found, prompting the procedure to terminate. Otherwise, we update $\mathscr{C}$ with the new centers computed at line 17. The calculation of centroids is perhaps the most time-consuming step, as the **For** loop at line 16 has a time complexity of $O(kn^3 \log n)$, assuming that centroids may have a size of $\Omega(n)$. Finally, the **For** loop where the assignment of points to clusters is performed takes a time of $O(kn^2 \log n)$ because a calculation of **DELTA** is done k times in the **For** loop of line 9. The last analysis applies for each iteration f. Therefore, **CLUSTERING-ORDER-TYPE** is an $O(fkn^3 \log n)$ running time algorithm.

In practice (and in our implementation), we can limit the number of iterations to some threshold. In our program, we allow f to iterate no more than 10 times. We observe that in most cases, the procedure stops after around 7 iterations. At the beginning of this section, we assumed the existence of a method **SORT-AROUND-BY** to sort by angle in $O(n \log n)$. In fact, an algorithm has been implemented in a Python library created by Ruy Fabila and Carlos Hidalgo [FH15]. This library, PyDCG (Python's Discrete and Combinatorial Geometry Library), is a collection of implementations of algorithms for Discrete and Combinatorial Geometry.

3.3.2 The code

The code is openly available in GitHub https://github.com/yeipi-mora/type-order-clustering

The main code is present in the clusterordtype.py file. The Algorithms 3, 4, 5, 2 and 1 are implemented in this file. More methods are implemented to compute and store the statistics related with the clusters. The dictionaries used to compare crossings were given by Dr. Ruy Fabila-Monroy.

[2]This approach may also be applicable for general metrics.

Algorithm 5: CLUSTERING-ORDER-TYPE

Data: (S, k)
Result: k clusters of S

Set $\mathscr{A}$ =**SORT-POINTS**(S); // We sort S.
Set $\mathscr{C}$ =**RANDOM**(S, k); // Random election of k points of S.
Set $f = 1$; // Iterating variable. Will also provide the number of iteration of the procedure.
while *TRUE* **do**
  Set $CL = \{CL_1, CL_2, \ldots, CL_k\}$; // CL for store our k clusters in this iteration.
  for $p \in S$ **do**
    Set $D = \infty$; // Variable to store distances
    i =**NULL**; // i for the index of the nearest nenter
    for $c \in \mathscr{C}$ **do**
      Set δ =**DELTA**$(S, p, c, \mathscr{A})$;
      if $\delta < D$ **then**
        $D = \delta$;
        i =index of c;
    $CL_i = CL_i \cup \{p\}$; // We assign p to cluster i.
  Set $UC = \emptyset$; // UC for Updated Centers.
  for $j = 1, \ldots, k$ **do**
    $UC = UC \cup \{$**CENTROID**$(CL_j)\}$; // We compute new centroids.
  $f = f + 1$;
  if $\mathscr{C} = UC$ **then**
    return CL
  else
    $\mathscr{C} = UC$; // We update the centers for next iteration.

3.4 Results

In this section, we present the results obtained from Algorithm 5, **CLUSTERING-ORDER-TYPE**. Figures 2.5, 2.6, 2.7, and 2.8 reveal a common pattern characterized by three arms joined to a central core. This observation leads us to conjecture that an optimal drawing of K_n is formed by these four clusters of points: three arms and one central core. We anticipate that each cluster optimizes the drawing for the number of points it contains. Therefore, our initial approach involves clustering optimal drawings and organizing them into four clusters. Figures 3.3, 3.4, 3.5, and 3.6 depict how optimal drawings are partitioned into four colors, with each color representing a separate cluster. Contrary to our initial intuition, clustering optimal drawings does not seem to result in the splitting of points into three arms and a central core. Instead, it appears that some clusters may encompass a portion of the central core. This observation is particularly evident in the clustering of K_{1000} (Figure 3.5), where two of the four clusters incorporate part of the central core. Similarly, in the clustering of K_{75} (Figure 3.4), the second cluster encloses the central core, albeit extending slightly beyond its boundaries. Yet, in each of the clustering drawings, we observe that at least one arm is delineated as its own cluster.

Now, let us refrain from examining clusters solely based on intuition and instead focus on the statistics they produce. We will use the notation CL_i to represent the ratio between the number of points in cluster i and the total number of points. That is, if we are clustering K_n, then

$$CL_i = \frac{\text{Number of points in cluster } i}{n}.$$

Besides, CR_i represents the ratio of the crossing number that results from our set of points and the optimal crossing number that has been found. If $\text{cr}^*(K_{m_i})$ is our small rectilinear crossing number that has been found so far, where m_i is the number of points in cluster i, then

$$CR_i = \frac{\text{Rectilinear crossings in cluster } i}{\text{cr}^*(K_m)}.$$

The value CL_i provides us with the measure of cluster i relative to the total number of points. As we have conjectured, the clusters might divide the set of points into clusters of n/k points. Therefore, we should expect clusters with CL_i of approximately 25%. In Table 3.1, we summarize the statistics for $n \in \{50, 75, 100, 200, 500, 1000\}$. We highlight in blue those values where CL_i is approximately 25%, corresponding to $n/4$, and in red those clusters where the number of crossings is almost optimal, at least 80%. We observe that cluster 3 in K_{50} achieves both $n/4$ number of points and a rectilinear crossing of approximately 96%, and the same is true for cluster 1 in K_{75}. These "good" clusters correspond to the "arms" in Figures 3.3 and 3.4.

Table 3.1: 4 clusters statistics

	50	75	100	200	500	1000
CL_1	0.34	0.253	0.23	0.145	0.188	0.115
CR_1	0.875	0.874	0.673	0.941	0.717	0.889
CL_2	0.26	0.267	0.34	0.2	0.143	0.218
CR_2	0.688	0.61	0.873	0.653	0.73	0.674
CL_3	0.28	0.28	0.14	0.165	0.646	0.333
CR_3	0.964	0.733	0.973	0.849	0.585	0.881
CL_4	0.12	0.2	0.29	0.49	0.12	0.334
CR_4	0.333	0.826	0.849	0.639	0.55	0.885

Despite obtaining several "good" clusters in the 4-cluster approach, we performed the clustering algorithm for $k = 6$ clusters. See Figures 3.7, 3.8, 3.9, and 3.10 to view the graphics of the clusters. Table 3.2 presents the statistics CL_i and CR_i for the same values $n \in \{50, 75, 100, 1000\}$. There are more clusters that satisfy being close to n/k points and have more than 80% of a small crossing number. For instance, clusters 1, 3, and 5 in K_{1000} may be optimal in their number of points and conform to our intuition of arms in optimal drawings. See Figure 3.10. When clustering optimal drawings into 6 clusters, it becomes much more evident that our notion of a central core is not truly a central core, but rather decomposes into three "shoulders" connecting to the arms.

Table 3.2: 6 clusters statistics

	50	75	100	200	500	1000
CL_1	0.2	0.29	0.2	0.14	0.146	0.144
CR_1	0.456	0.846	0.554	0.923	0.941	0.949
CL_2	0.18	0.07	0.12	0.21	0.192	0.192
CR_2	0.529	0.529	0.75	0.644	0.613	0.62
CL_3	0.18	0.12	0.133	0.115	0.19	0.142
CR_3	0.947	0.781	0.925	0.888	0.616	0.94
CL_4	0.08	0.23	0.227	0.195	0.19	0.195
CR_4	1	0.629	0.66	0.592	0.61	0.615
CL_5	0.08	0.13	0.2	0.195	0.142	0.141
CR_5	1	0.592	0.558	0.598	0.942	0.898
CL_6	0.28	0.16	0.12	0.145	0.14	0.19
CR_6	0.964	0.846	0.75	0.934	0.934	0.622

3.4.1 A kind of recursion

As we stated, our goal is to understand how optimal drawings are constructed. From the statistics, we can glimpse that there are sections of the optimal drawings that are close to being optimal. However, since the coordinates of the points in optimal drawings are of a large scale, any graphical representation will omit details within the clusters that could provide more insight into how optimal drawings are built. To address this issue, we will perform a zoom into each cluster to obtain a better visualization of each one. Let w be the average vector of cluster i. That is, if we denote by $x_1, \ldots, x_r$ the vectors belonging to cluster i, then

$$w = \frac{1}{r} \sum_{j=1}^{r} x_j.$$

Then, we *expand* the points $x_1, \ldots, x_r$ in a perpendicular direction v of w by a factor of α. Expanding means applying the linear transformation:

$$T(x) = w + \alpha(x - w).$$

This transformation expands each point x away from the cluster center w by a factor of α. Because affine transformations, and therefore linear transformations, do not modify the order type, T simply expands our set of points $\{x_1, \ldots, x_r\}$ by a factor of α. In other words, the set $T(\{x_1, \ldots, x_r\})$ will have the same crossing number. Figure 3.14 shows the result after applying T to clusters 1, 2, and 3 of the optimal drawing of K_{50} following a 4-clustering. All

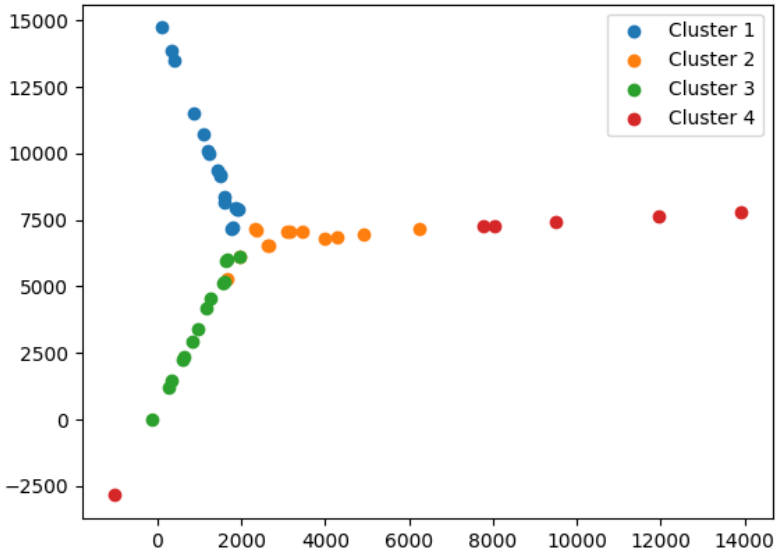


Figure 3.3: 4 clustering of K_{50}

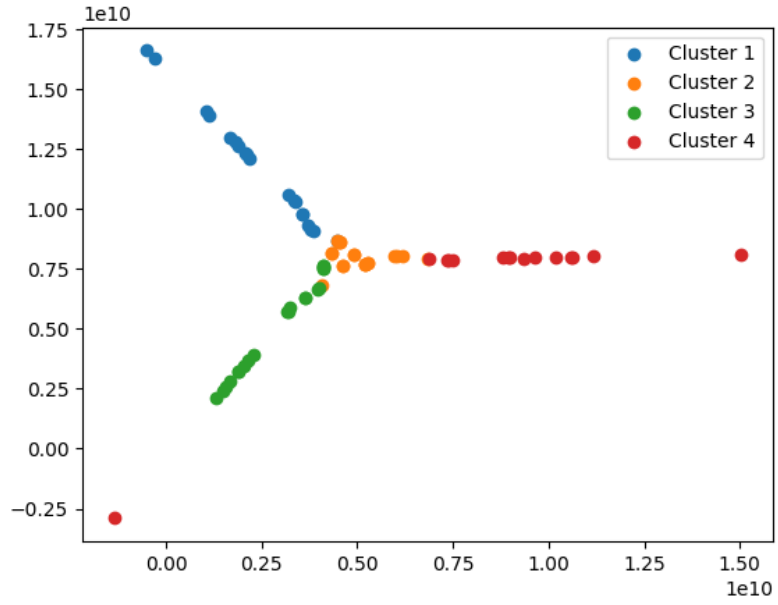


Figure 3.4: 4 clustering of K_{75}

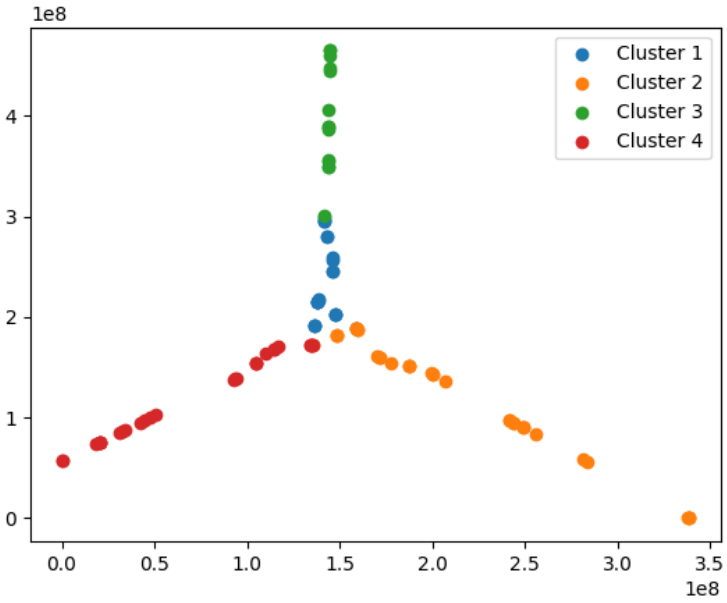


Figure 3.5: 4 clustering of K_{100}

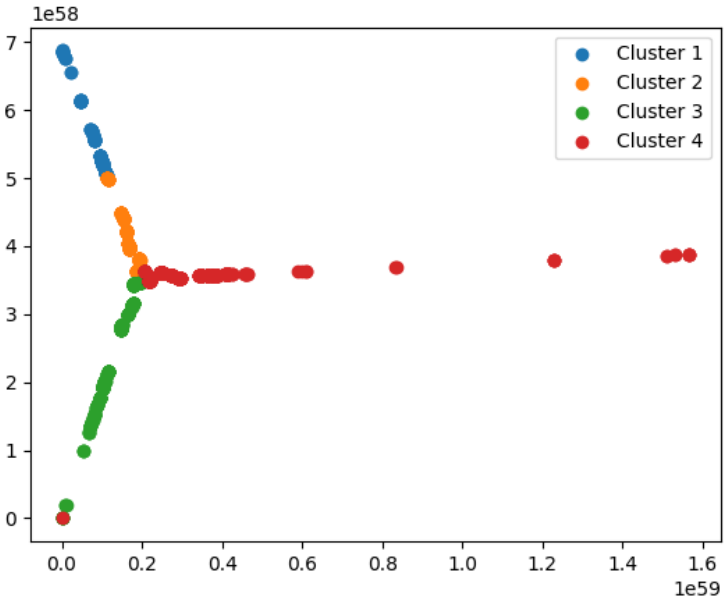


Figure 3.6: 4 clustering of K_{1000}

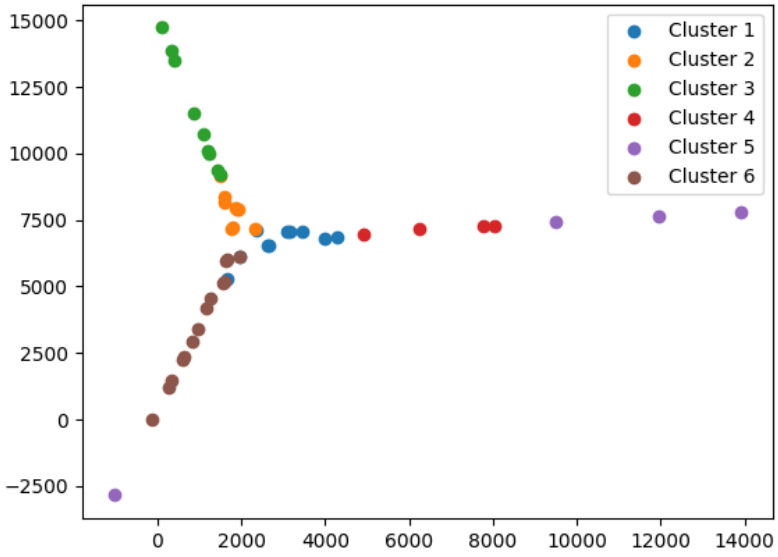


Figure 3.7: 6 clustering of K_{50}

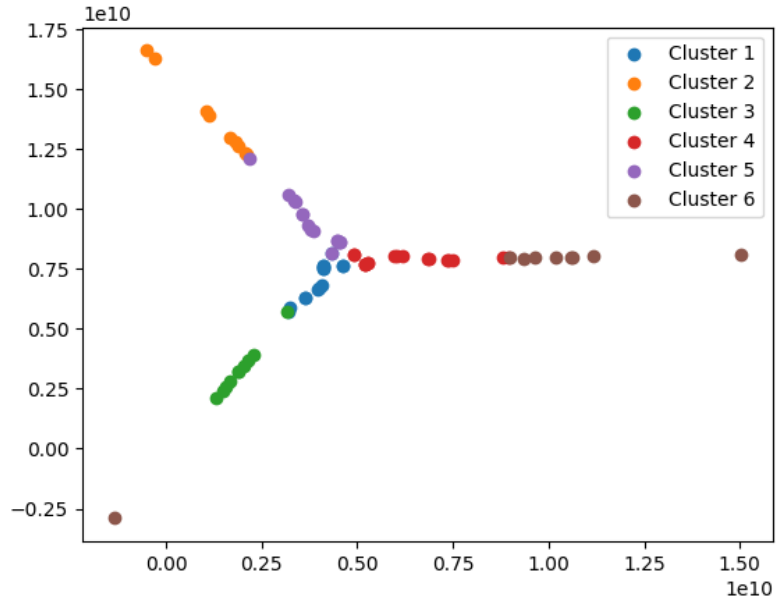


Figure 3.8: 6 clustering of K_{75}

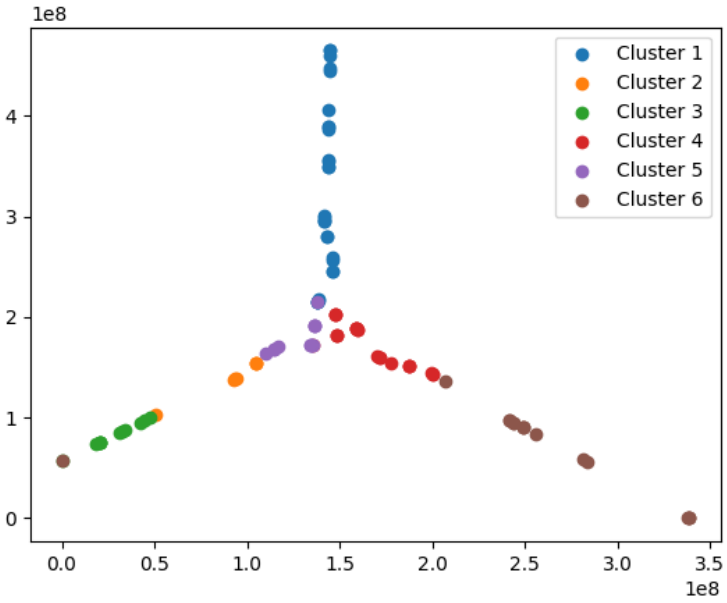


Figure 3.9: 6 clustering of K_{100}

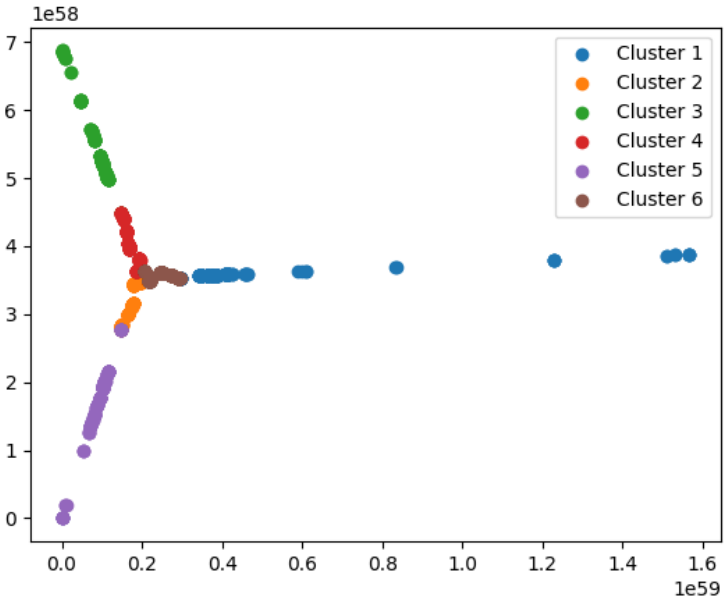


Figure 3.10: 6 clustering of K_{1000}

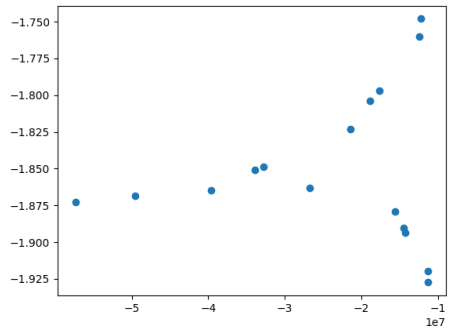


Figure 3.11: 4 clustering: cluster 1 of K_{50}

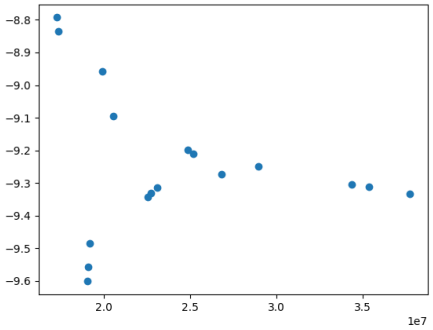


Figure 3.12: 4 clustering: cluster 2 of K_{50}

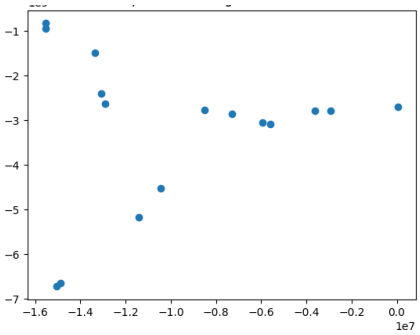


Figure 3.13: 4 clustering: cluster 3 of K_{50}

Figure 3.14: Expansion of clusters $1, 2$ and 3 of optimal drawing of K_{50}

three clusters satisfy Theorem 2.0.3, as the convex hull of each is a triangle. We also observe a similar pattern to the original optimal drawing of K_{50}: arms with shoulders connected by a central core. Additionally, from Table 3.1, clusters 1 and 3 are both close to optimal in terms of the crossing number, and cluster 3 also contains almost $n/4$ points.

We showcase some expansions to certain clusters of optimal drawings of K_n in Figures 3.14, 3.19, and 3.23. Figure 3.16 is remarkably similar to the original optimal drawing, even though the crossing number of Cluster 2 is approximately 65% of the optimal crossing number. See Table 3.2. Similarly, this occurs with the expansions in K_{500} with a 4-cluster approach, as presented in Figure 3.23; the statistics of all clusters are far from being close to the optimal.

We can observe several clusters that preserve the same kind of pattern, which might indicate a recursive construction of optimal drawings with few crossings. Moreover, the recursion seems to be independent of the number of clusters performed, i.e., independent of k. Although the recursion is not present in every cluster, we can conclude by presenting a comparison between the performed clusters of K_{500} and K_{1000} in Figures 3.26 through 3.41.

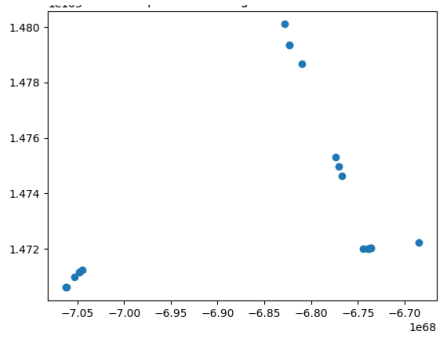


Figure 3.15: 6 clustering: cluster 1 of K_{200}

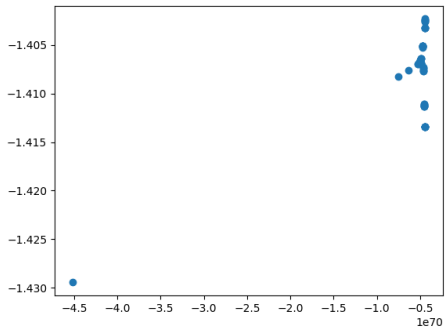


Figure 3.16: 6 clustering: cluster 2 of K_{200}

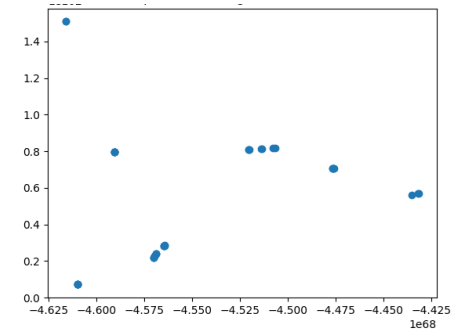


Figure 3.17: 6 clustering: cluster 3 of K_{200}

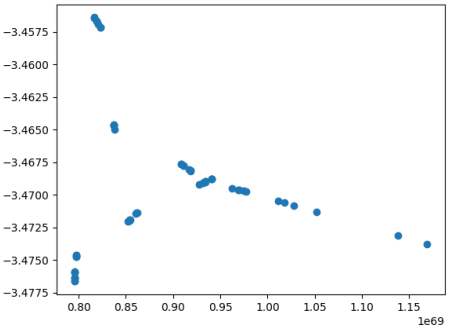


Figure 3.18: 6 clustering: cluster 5 of K_{200}

Figure 3.19: Expansion of clusters $1, 2, 3$ and 5 of optimal drawing of K_{200}

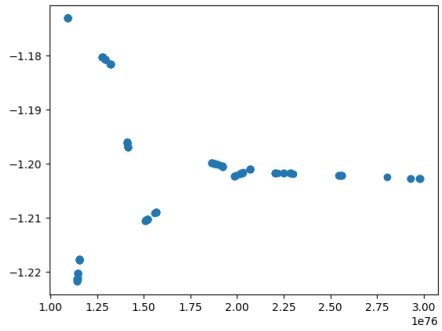


Figure 3.20: 4 clustering: cluster 2 of K_{500}

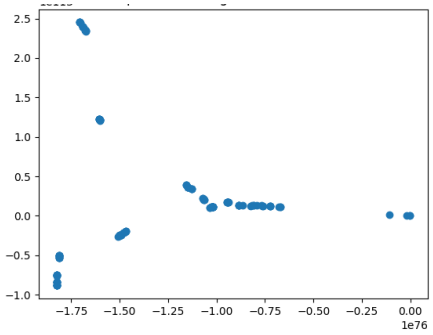


Figure 3.21: 4 clustering: cluster 3 of K_{500}

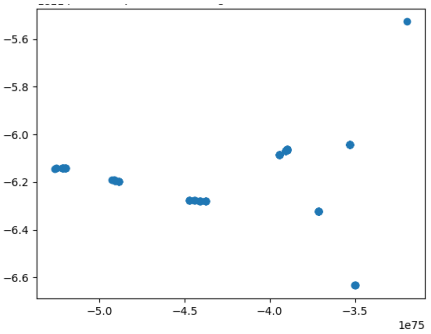


Figure 3.22: 4 clustering: cluster 4 of K_{500}

Figure 3.23: Expansion of clusters $1, 2$ and 4 of optimal drawing of K_{500}

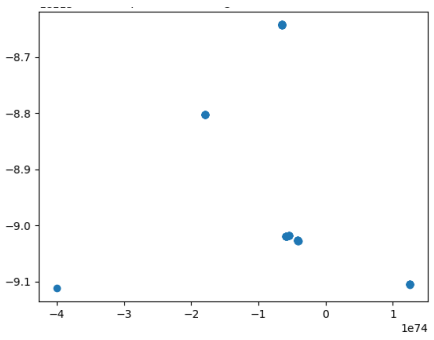


Figure 3.24: 6 clustering: cluster 1 of K_{500}

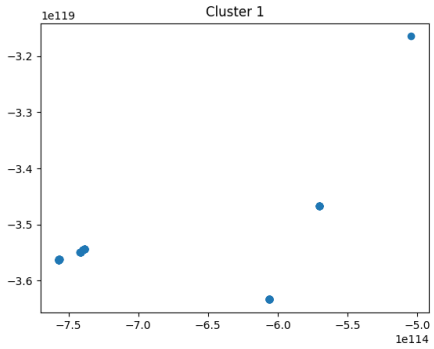


Figure 3.25: 6 clustering: cluster 1 of K_{1000}

Figure 3.26: Comparison of the clusters 1 between K_{500} and K_{1000}

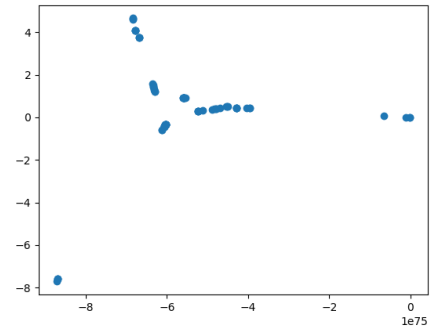


Figure 3.27: 6 clustering: cluster 2 of K_{500}

Figure 3.28: 6 clustering: cluster 5 of K_{1000}

Figure 3.29: Comparison of the clusters 2 and 5 between K_{500} and K_{1000}

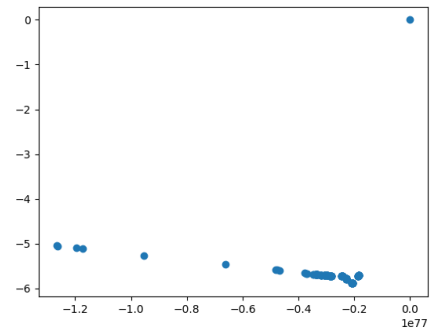


Figure 3.30: 6 clustering: cluster 3 of K_{500}

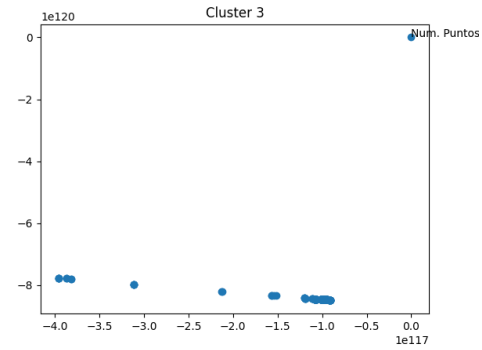


Figure 3.31: 6 clustering: cluster 3 of K_{1000}

Figure 3.32: Comparison of the clusters 3 between K_{500} and K_{1000}

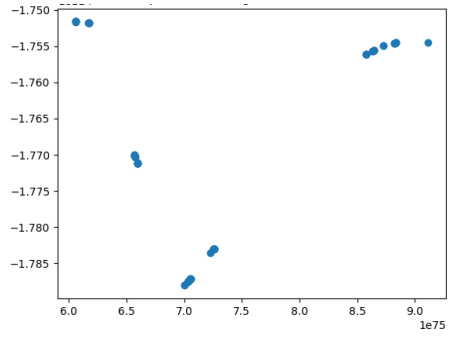


Figure 3.33: 6 clustering: cluster 4 of K_{500}

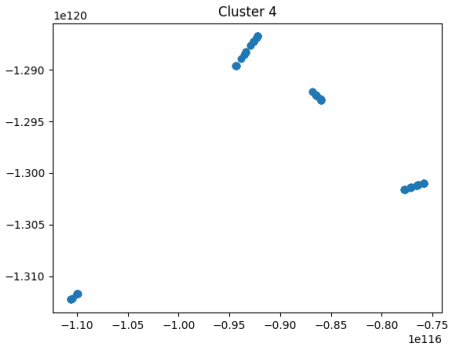


Figure 3.34: 6 clustering: cluster 4 of K_{1000}

Figure 3.35: Comparison of the clusters 4 between K_{500} and K_{1000}

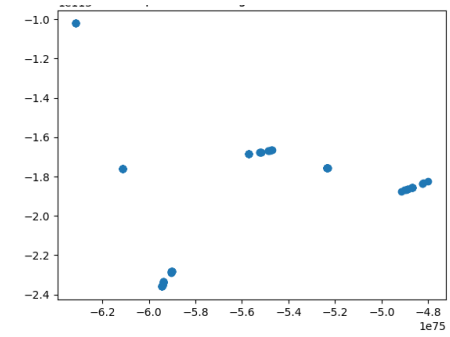


Figure 3.36: 6 clustering: cluster 5 of K_{500}

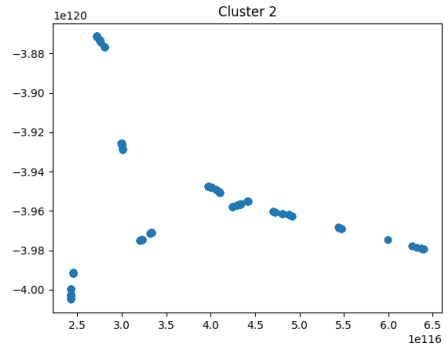


Figure 3.37: 6 clustering: cluster 2 of K_{1000}

Figure 3.38: Comparison of the clusters 5 and 2 between K_{500} and K_{1000}

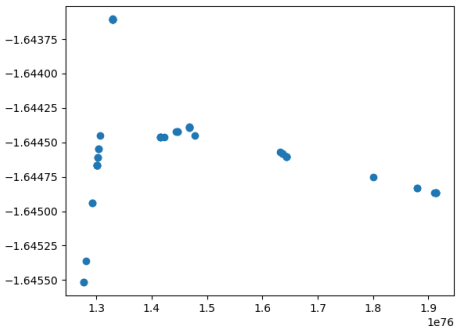


Figure 3.39: 6 clustering: cluster 6 of K_{500}

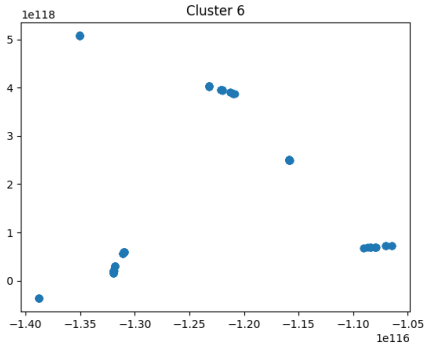


Figure 3.40: 6 clustering: cluster 6 of K_{1000}

Figure 3.41: Comparison of the clusters 6 between K_{500} and K_{1000}

Bibliography

[Kur30] Casimir Kuratowski. "Sur le problème des courbes gauches en Topologie". In: *Fundamenta Mathematicae* 15 (1930), pp. 271–283. URL: https://api.semanticscholar.org/CorpusID:117944571.

[Zar55] Casimir Zarankiewicz. "On a problem of P. Turan concerning graphs". In: *Fundamenta Mathematicae* 41 (1955), pp. 137–145. URL: https://api.semanticscholar.org/CorpusID:118616841.

[HH63] Frank Harary and Anthony Hill. "On the Number of Crossings in a Complete Graph". In: *Proceedings of the Edinburgh Mathematical Society* 13 (1963), pp. 333–338.

[BK64] Jaroslav Blazek and Milan Koman. "A minimal problem concerning complete plane graphs". In: *Theory of graphs and its applications, Czech. Acad. of Sci* (1964), pp. 113–117.

[Guy68] R.K. Guy. *The Decline and Fall of Zarankiewicz's Theorem*. Research paper - University of Calgary, Department of Mathematics. University of Calgary, Department of Mathematics, 1968.

[Jen71] H.F Jensen. "An upper bound for the rectilinear crossing number of the complete graph". In: *Journal of Combinatorial Theory, Series B* 10.3 (1971), pp. 212–216. ISSN: 0095-8956. DOI: https://doi.org/10.1016/0095-8956(71)90045-1. URL: https://www.sciencedirect.com/science/article/pii/0095895671900451.

[Sin71] David Singer. *The rectilinear crossing number of certain graphs*. 1971.

[Guy72] Richard K. Guy. "Crossing numbers of graphs". In: (1972). Ed. by Y. Alavi, D. R. Lick, and A. T. White, pp. 111–124.

[HT74] John Hopcroft and Robert Tarjan. "Efficient Planarity Testing". In: *J. ACM* 21.4 (Oct. 1974), pp. 549–568. ISSN: 0004-5411. URL: https://doi.org/10.1145/321850.321852.

[Tur77] Paul Turán. "A note of welcome". In: *J. Graph Theory* 1 (1977), pp. 7–9. URL: https://api.semanticscholar.org/CorpusID:41065502.

[GJ83] M. R. Garey and David S. Johnson. "Crossing Number is NP-Complete". In: *Siam Journal on Algebraic and Discrete Methods* 4 (1983), pp. 312–316.

[GP83] Jacob E. Goodman and Richard Pollack. "Multidimensional Sorting". In: *SIAM Journal on Computing* 12.3 (1983), pp. 484–507. DOI: 10.1137/0212032. URL: https://doi.org/10.1137/0212032.

[GPS89] Jacob E. Goodman, Richard Pollack, and Bernd Sturmfels. "Coordinate representation of order types requires exponential storage". In: (1989).

[GT01] J.L. Gross and T.W. Tucker. *Topological Graph Theory*. Dover books on mathematics. Dover Publications, 2001. ISBN: 9780486417417.

[Coh+06] Johanne Cohen et al. "Optimal Linear Arrangement of Interval Graphs". In: *Mathematical Foundations of Computer Science 2006*. Ed. by Rastislav Královič and Paweł Urzyczyn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[ÁF07] Bernardo M. Ábrego and Silvia Fernández-Merchant. "Geometric drawings of Kn with few crossings". In: *Journal of Combinatorial Theory, Series A* 114.2 (2007), pp. 373–379. ISSN: 0097-3165. DOI: https://doi.org/10.1016/j.jcta.2006.05.003. URL: https://www.sciencedirect.com/science/article/pii/S009731650600077X.

[Aic+07] Oswin Aichholzer et al. "New Lower Bounds for the Number of (≤ k)-Edges and the Rectilinear Crossing Number of Kn". In: *Discrete & Computational Geometry* 38.1 (July 2007), pp. 1–14. ISSN: 1432-0444. DOI: 10.1007/s00454-007-1325-8. URL: https://doi.org/10.1007/s00454-007-1325-8.

[MNV12] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. "The planar k-means problem is NP-hard". In: *Theoretical Computer Science* 442 (2012). Special Issue on the Workshop on Algorithms and Computation (WALCOM 2009), pp. 13–21. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2010.05.034. URL: https://www.sciencedirect.com/science/article/pii/S0304397510003269.

[Mat13] J. Matousek. *Lectures on Discrete Geometry*. Graduate Texts in Mathematics. Springer New York, 2013. ISBN: 9781461300397.

[FL14] Ruy Fabila-Monroy and Jorge López. *Computational search of small point sets with small rectilinear crossing number*. 2014. arXiv: 1403.1288 [math.CO].

[DY15] Xu Dongkuan and Tian Yingjie. "A comprehensive survey of clustering algorithms". In: *Annals of Data Science* 2 (2015), pp. 165–193. DOI: https://doi.org/10.1007/s40745-015-0040-1.

[FH15] Ruy Fabila-Monroy and Carlos Hidalgo. *PyDCG*. http://rfabila.github.io/PyDCG/. 2015.

[Hid15] Carlos Hidalgo-Toscano. "Un algoritmo para recorrer las celdas de un arreglo de rectas." MA thesis. Departamento de Matemáticas, Cinvestav, 2015.

[Die17] R. Diestel. *Graph Theory: 5th edition*. Springer Graduate Texts in Mathematics. Springer-Verlag, © Reinhard Diestel, 2017. ISBN: 9783961340057.

[DF17] Frank Duque and Ruy Fabila-Monroy. *Updating the Number of Crossings in Rectilinear Drawings of the Complete Graph*. 2017. arXiv: 1609.00867 [cs.CG].

[Aic+20] Oswin Aichholzer et al. *An Ongoing Project to Improve the Rectilinear and the Pseudolinear Crossing Constants*. 2020. arXiv: 1907.07796 [math.CO].

[Xu+21] Yongjun Xu et al. "Artificial intelligence: A powerful paradigm for scientific research". In: *The Innovation* 2.4 (2021), p. 100179. ISSN: 2666-6758. DOI: https://doi.org/10.1016/j.xinn.2021.100179.

[Cor+22] T.H. Cormen et al. *Introduction to Algorithms, fourth edition*. MIT Press, 2022. ISBN: 9780262367509.