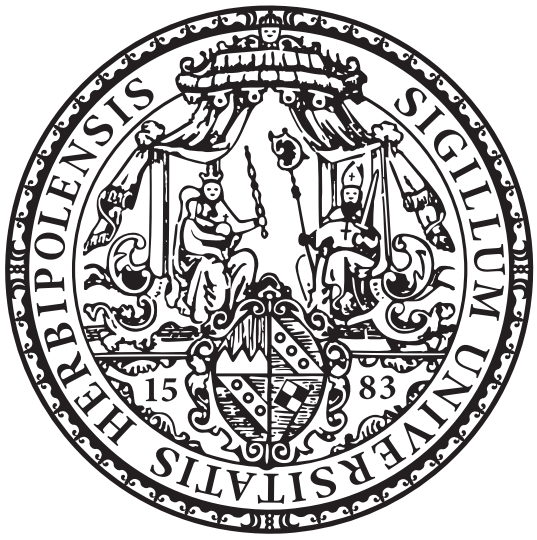
Bachelor Thesis

On the Geometry of Chord Diagrams

Max Kusmitschov

Date of Submission:	February 5, 2026
Revised:	April 30, 2026
Advisors:	Prof. Dr. Alexander Wolff
	Tim Hegemann, M. Sc.



Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

Abstract

Chord diagrams are a popular method of visualizing relationships between data points. A chord diagram consists of nodes represented by circular arcs, that partition a circle, and chords represented by curves that connect the nodes, visualizing their relation to each other. In a chord diagram with chords drawn as straight-line segments, two chords cross exactly once if their endpoints alternate on the circle circumference. In practice however, in most chords diagrams, chords are drawn with Bézier curves or similar smooth curves, since they are more aesthetically pleasing. This leads to the problem we aim to tackle in this work. Unlike straight lines, Bézier curves can theoretically cross more than once and thus contribute unnecessary intersections to the diagram.

In this work, we prove that the edges of a chord diagram can be drawn without unnecessary intersections as circular arcs, and we show how to place the control points of quadratic Bézier curves such that they appear to have the same property. Additionally, we experimentally show the advantage of the above mentioned drawing styles over straight-line segments, and finally, we propose some improvements and additional chord drawing styles to explore.

Zusammenfassung

Chord-Diagramme sind eine beliebte Methode zur Visualisierung von Beziehungen zwischen Datenpunkten. Ein Chord-Diagramm besteht aus Knoten, die durch Kreisbögen dargestellt werden, die einen Kreis bilden, und Sehnen, die durch Kurven dargestellt werden, die Knoten verbinden und deren Beziehung zueinander visualisieren. Wenn die Sehnen in einem Chord-Diagramm als Strecken gezeichnet werden, kreuzen sich zwei Sehnen genau einmal, wenn ihre Endpunkte sich auf dem Kreis abwechseln. In der Praxis werden Chord-Diagramme jedoch meist mit Bézier-Kurven oder ähnlichen Kurven gezeichnet, wahrscheinlich weil sie ästhetisch ansprechender sind. Dies führt zu dem Problem, das wir in dieser Arbeit angehen wollen. Im Gegensatz zu Strecken können sich Bézier-Kurven theoretisch mehr als einmal kreuzen, was zu unnötigen Kreuzungen im Diagramm führt.

In dieser Arbeit beweisen wir, dass die Sehnen eines Chord-Diagramms ohne unnötige Kreuzungen als Kreisbögen gezeichnet werden können, und wir zeigen, wie die Kontrollpunkte quadratischer Bézier-Kurven, die dieselbe Eigenschaft zu haben scheinen, platziert werden können. Darüber hinaus zeigen wir experimentell den Vorteil der oben genannten Zeichenstile gegenüber Strecken auf und schlagen schließlich einige Verbesserungen und weitere Chord-Zeichnungsstile vor.

Contents

1 Introduction

Data visualization is the important task of presenting raw data and statistics to an intended target audience with graphics in a digestible way, so that they may easily access the information. Easily-interpretable graphics can help get a general understanding, a quick overview and even some deeper insight on a subject where raw numbers and data would fail. Well-visualized data simplifies the process of identifying clusters, recognizing patterns and trends, finding outliers and unusual groupings.

A chord diagram is a tool of data visualization that focuses on illustrating binary relationships and flows between entities. A chord diagram consists of nodes represented by circular arcs that partition a circle, and edges represented by curves that connect the nodes, visualizing their relation to each other. To give an example of what a chord diagram can be used for: say, we have a group of researchers and we want to show who worked with whom within the group. This is something that a chord diagram could depict in an easy-to-understand manner. The circular arcs (nodes) represent the researchers and the curves connecting pairs of arcs represent the pair of researchers that have worked together. Furthermore, if we wanted, we could also visualize how many times the researchers collaborated by drawing differently sized connections in proportion to the number of times a pair of researchers has collaborated. While chord diagrams do not offer a whole lot of flexibility, their simplicity, as a result, allows to fit a lot of data in a small graphic that remains interpretable despite how dense it is.

Due to this, chord diagrams are prominently used in health, medicine and biology [FSH19, HBS+14, WDW+24], to visualize socio-economic flows such as trade [YSC+25] and migration [AS14, NOF20, SABS14], and more.

To get a better understanding for what and how chord diagrams are used, we will consider some real-world use cases from the above mentioned papers.

First, consider the chord diagram in Fig. 1.1 depicting migration flows around the world [AS14]. In this diagram, each node represents a world region, more specifically, the total migration volume of the corresponding region, immigration plus emigration. Note that migration within the same region counts "double" since it contributes to emigration and immigration at the same time. The size of nodes is proportional to said volume and takes up space on the diagram relative to the total migration volume of all regions combined. Chords, here, represent the migration volume from one region to another, with self-chords within the same node, representing migration within the region. Additionally, since migration is a flow with direction, the chords also have to show direction. In this diagram, the outgoing endpoint of a chord is closer to the corresponding node, than the incoming endpoint is to its corresponding node. This clearly is not a very good method of showing direction, but the consistent use of color and order of chords makes it a little better. The color of chords is determined by the

color of its startpoint, and chord endpoints are grouped for each node so that incoming flow is grouped on one side of the node and outgoing flow on the other side.

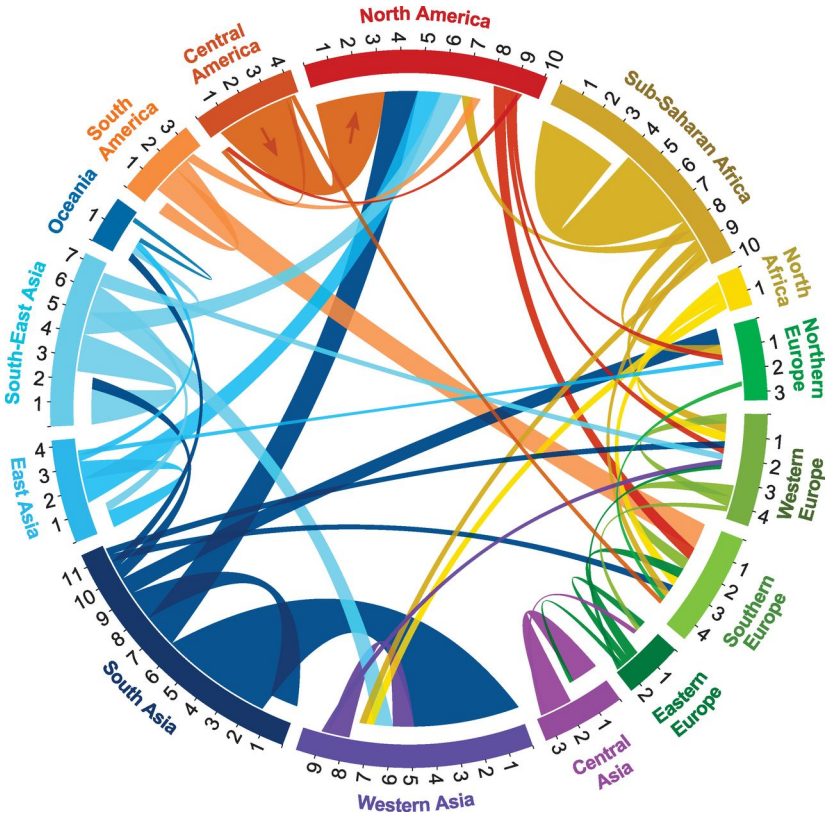


Fig. 1.1: Chord diagram of migration flows between and within world regions during 2005 to 2010. Tick marks show the number of migrants (inflows and outflows) in millions. Only flows containing at least 170,000 migrants are shown. Figure from [AS14].

We now consider another set of chord diagrams. The four diagrams in Figure 1.2 are used to visualize trade volume between countries within a trade network over four different time periods. By using multiple diagrams, the author of the figure can rather easily convey how the market has evolved over time. The diagrams in this figure are very similar in structure to the previous diagram. What sets it apart is that incoming chords have a sort of arrowhead which makes it easier to identify them as incoming and the use of color to group the nodes representing countries by world region.

Observing these diagrams, it is rather easy to get some general insight on the subject. Following individual chords, however, requires attention or is downright impossible for small chords in the case of the diagrams in Fig. 1.2. Admittedly, chord diagrams, especially those with lots of data, are not meant to clearly visualize the exact flow of every chord. Nevertheless, these diagrams are far from perfect and we are confident that they could be improved in terms of readability by reducing visual clutter.

We believe the main contribution to visual clutter stems from intersections of chords.

A simple solution to address this problem is to use straight-line chords: a pair of straight-line chords cross exactly once if their endpoints alternate on the circumference of the diagram circle, and only then. This is the best outcome for any uninterrupted stroke that stays within the circle. However, using straight-line segments is not a very elegant solution because they often lead to small crossings angles, connect at shallow angles with nodes and generally look aesthetically less pleasing than smoothly bent curves.

So the main goal of this work is to prove that drawing styles other than straight lines avoid unnecessary crossings and at the same time offer better readability.

In the remainder of this paper we first cover some terms and properties for our proposed solutions. Next, we present the solutions themselves, of which there are two: a circular-arc drawing style and a quadratic-curve drawing style. Then, we compare those solutions among themselves and with a straight-line approach. Finally, we conclude with a summary, provide possible improvements and discuss completely different solutions.

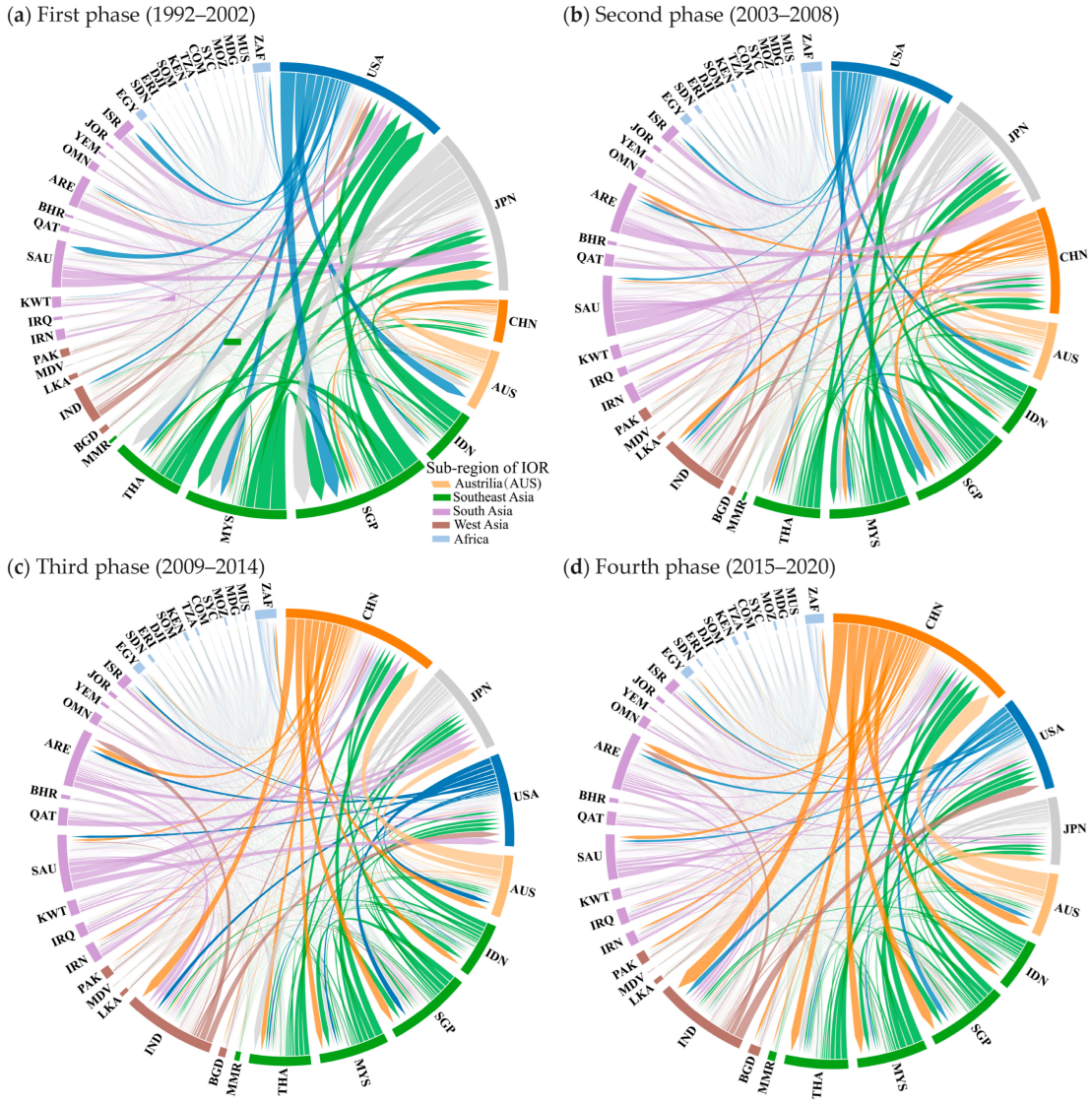


Fig. 1.2: Chord diagram visualization of trade flows of the IOR trade networks over four time periods. Figure from [YSC+25].

2 Background

In this chapter we review some terminology used in this work as well as some knowledge required for following along with the solutions we propose.

2.1 Concepts and Definitions

In this section we cover terminology regarding *graphs*. A graph $G = (V, E)$ is comprised of

- the set of *vertices* V, and
- the set of *edges* $E \subseteq \binom{V}{2} = \{\{u, v\} \subseteq V \mid u \neq v\}$.

We define $n = |V|$ and $m = |E|$ as the numbers of vertices and edges, respectively. The neighborhood of a vertex $v \in V$ is defined as $adj(v) = \{u \in V \mid \{u, v\} \in E\}$, and the degree of vertex is denoted as $\deg(v) = |adj(v)|$. A pair of vertices u, v is *connected* if there is a path from u to v: $u = w_0, w_1, \ldots, w_n = v$ with $\{w_i, w_{i+1}\} \in E$. A graph is connected if every pair of vertices $u, v \in V$ is connected. In this work, vertex and node as well as edge and chord will be used synonymously.

2.2 Bézier Curves

In this section we cover so called Bézier curves. We aim to provide a basic understanding of what they are along with some of their properties that we will make use of later.

Bézier curves, named after their inventor Dr. Pierre Bézier, are parametric curves that were designed to be intuitive for artists and designers. They are widely used in computer graphics and related fields today.

Bézier curves are constructed with *control points* that can be thought of as masses that pull points on the curve with varying strength as the parameter t varies between $0 \leq t \leq 1$. Visually, this can be interpreted as the curve mimicking the shape of the chain of control points (see Fig. 2.1a).

The exact equation for a Bézier curve of degree n (with $n + 1$ control points) is:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i \tag{2.1}$$

with P_i being the i-th control point.

For example, the equation of the quadratic Bézier curve characterized by the control points $P_0 = (0, 0)$, $P_1 = (2, 3)$ and $P_2 = (4, 0)$ (see Fig. 2.2) would be:

$$B(t) = \binom{2}{0}(1-t)^{2-0}t^0(0,0) + \binom{2}{1}(1-t)^{2-1}t^1(2,3) + \binom{2}{2}(1-t)^{2-2}t^2(4,0), \quad (2.2)$$

which can be simplified to

$$B(t) = (1-t)^2(0,0) + 2(1-t)t(2,3) + t^2(4,0). \quad (2.3)$$

This can be further divided in x- and y-components:

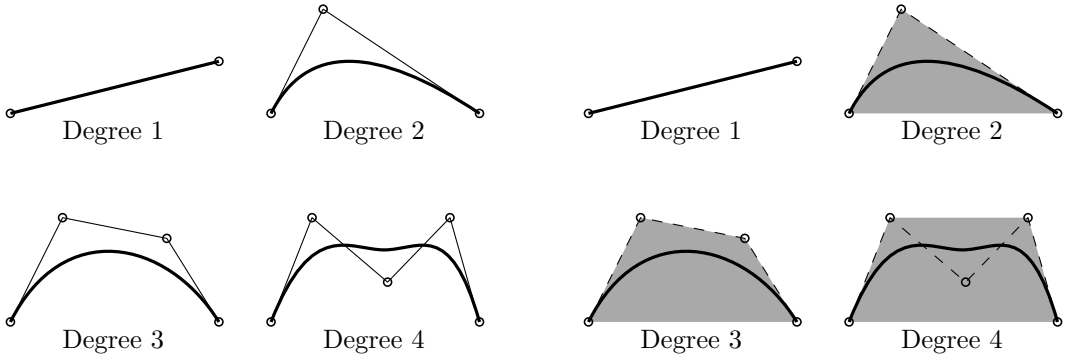
$$x(t) = 2 \cdot 2(1-t)t + 4 \cdot t^2 = 4t - 4t^2 + 4t^2 = 4t \quad (2.4)$$

$$y(t) = 3 \cdot 2(1-t)t = -6t^2 + 6t \quad (2.5)$$

Lastly, there are two properties that we will be making use of later.

First, all quadratic Bézier curves represent sections of parabolas [Mar06]. Quadratic Bézier curves are sections of transformed (rotated and stretched) parabolas. Technically, the equation represents the whole parabola by going beyond $0 \leq t \leq 1$ in the positive and negative directions.

Secondly, all Bézier curves lie completely within the polygon spanned by the outermost control points (Convex Hull Property, see Fig. 2.1b) [Sed12]. For quadratic Bézier curves, this polygon is equal to their *control polygon*. A control polygon is the polygon comprised of the control points of a Bézier curve.



(a) Bézier curve examples. (b) Convex Hull Property examples.

Fig. 2.1: Bézier Curves of degrees 1 to 4

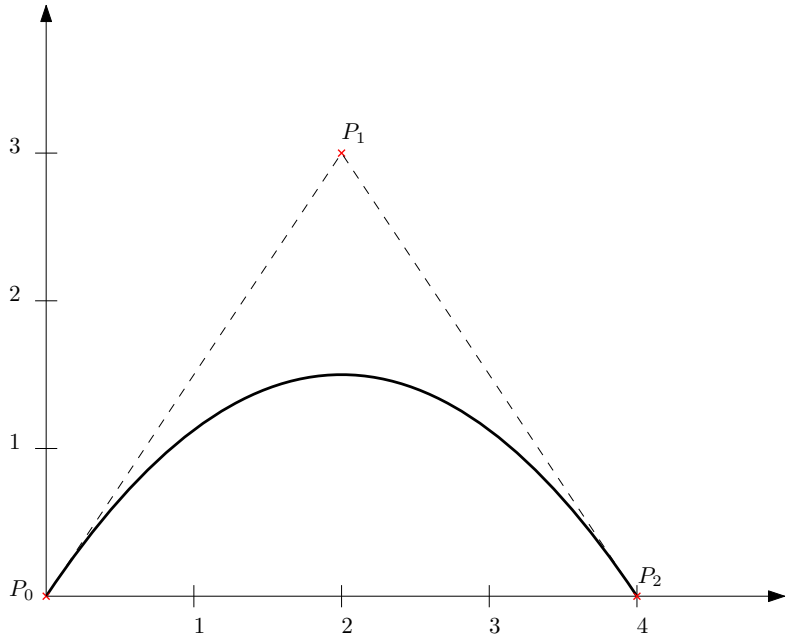


Fig. 2.2: Quadratic Bézier curve defined by control points $P_0 = (0, 0)$, $P_1 = (2, 3)$, and $P_2 = (4, 0)$.

3 Drawing Styles for Chord Diagrams

In this chapter we present the general structure of our chord diagram drawing styles. Then we explain in detail how we order chord-endpoints with an algorithm to avoid crossings between chords of the same node. Finally, we present two solutions that achieve the exact same number of crossings for any given diagram as the straight-line approach.

Definition 3.1 (Chord Diagram)**.** *A* chord diagram *is a drawing of a graph. The vertices (nodes) are represented by circular arcs that partition a circle. The edges (chords) are represented by curves, the endpoints of which lie on the circular arcs of the vertices they connect. The curves have to lie completely within the circle of nodes.*

See Figures 3.1 and 3.2, for examples of our chord diagram drawing styles.

The general structure of chord diagrams in our drawing styles is derived from to the ones used in ChordLink [ADM+19, ADM+22].

Nodes have *weight* which corresponds to their degree and take up space on the circumference in proportion to their weight. For example, four nodes each with degree 3 would have equal weight and thus divide the circumference in four equal parts of size $\pi/2$. We add a little padding between them such that identifying their boundaries is easier. This also increases the distance between neighboring nodes, such that tiny chords between them become bigger and more pronounced.

Our algorithm does not change the order of nodes. We assume the order of nodes on the circumference is fixed and part of the input. Nodes are just placed one by one starting at $0°$.

Chords in our drawing styles have no weight and evenly divide the *node arc* (circular arc of a node). For example, a node with four chords is divided into four equally sized sub-arcs and the endpoints of chords lie on the bisector of their corresponding sub-arc. Which sub-arc corresponds to which chord needs to be determined by an algorithm, since assigning them arbitrarily will lead to unnecessary crossings between chords of the same node.

The idea of the algorithm (Alg. 1) is to divide the connections into *lefts* and *rights* from the perspective of a node and then sort those so that the shortest chord is outermost (on the node arc), the second shortest is second outermost and so on, which should end with the longest being innermost.

In the algorithm this is achieved by, first, finding the point on the circle which is opposite to the center of the node arc (mirror point), and then, the algorithm splits based on whether the node lies in the $(\pi, 2\pi)$ interval (l.7) or in the $(0, \pi)$ interval (l.20).

Let us take a closer look at the first case (lines 7–18), the second case is essentially just mirrored. Nodes that lie between the mirror point and the node, land in *lefts*, while

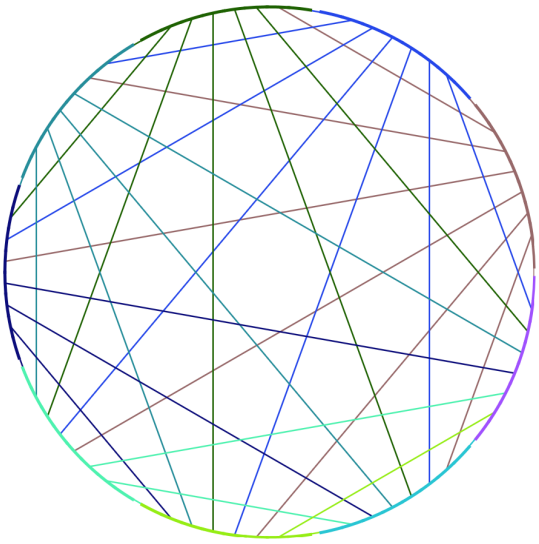
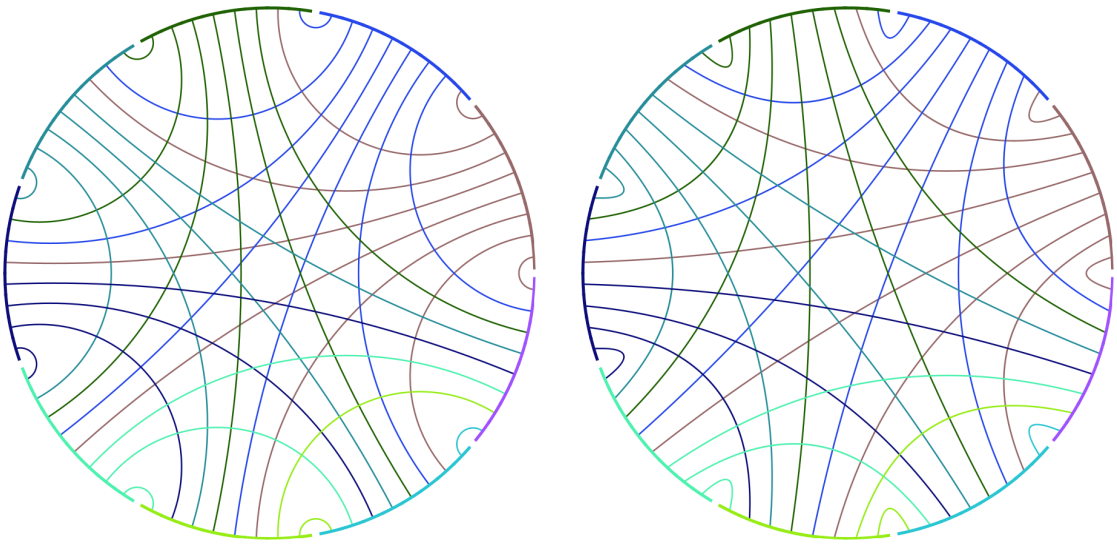


Fig. 3.1: Straight-line chord diagram.



(a) Circular-arc chord diagram.

(b) Quadratic-Curve chord diagram.

Fig. 3.2: Chord Diagram examples of our drawing styles.

the rest go to *rights* (lines 8–12). *Lefts* are sorted by the start angle of nodes (on the circumference) in descending order, meaning closest to farthest relative the current node (l.13). The same applies to *rights*, but before that they need to be split, since nodes that lie in the $(0, mirror)$ interval are actually further than nodes that lie in the $(mirror, 2\pi)$ interval. Although the goal of sorting *rights* in descending order is actually the opposite of *lefts*, the goal is to sort farthest to closest. After sorting *rights* they need to be joined again by concatenating in the same farthest to closest order (lines 14–18). Lastly, *lefts* and *rights* need to be concatenated, and the ordered list of connections needs to be saved for that particular node. See Fig. 3.3 for an example of how the algorithm would determine the order of chords for a node. By observing the figure it becomes apparent the algorithm in reality just orders chords in the order the nodes they are connected to appear when following along the circumference counter-clockwise.

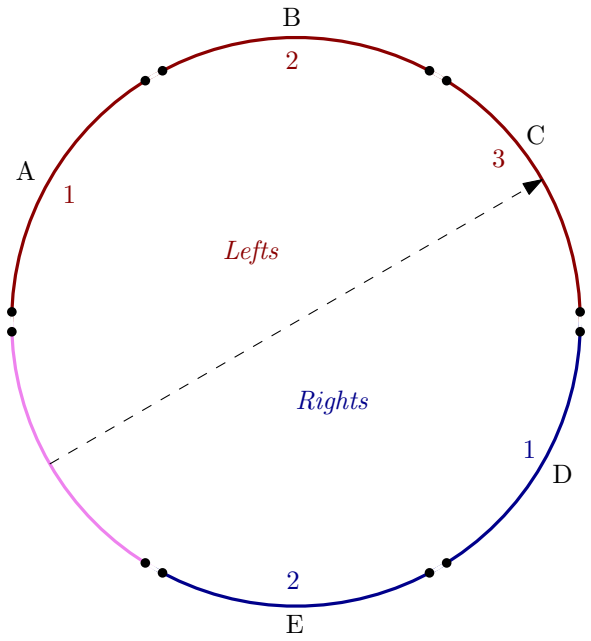


Fig. 3.3: An example of how OrderChords() orders chords for the purple node. Atfer concatenating *lefts* and *rights*, the final order would be: A, B, C, D, E.

3.1 Straight-Line Drawing Style

In this section we define a straight-line drawing style, show why it does not have unnecessary intersections, and present some of the issues of this drawing style.

Definition 3.2 (Straight-line Chord Diagram)**.** *A* straight-line chord diagram *is a chord diagram with chords represented by straight-line segments.*

If we imagine a straight-line chord diagram with one chord already drawn-in that divides the circle in two regions, it is obvious that another chord can cross the existing

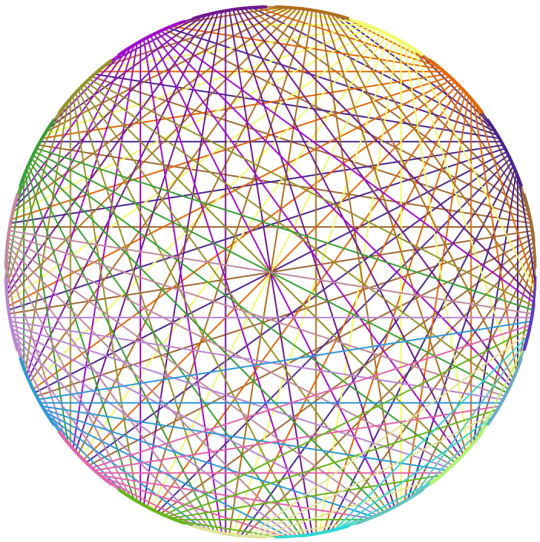


Fig. 3.4: Example of a difficult to read straight-line chord diagram (K_{20}).

chord only if the endpoints each lie in separate regions. Since straight lines can only cross once, a pair of chords in such a chord diagram crosses exactly once and only if its endpoints alternate on the circle circumference.

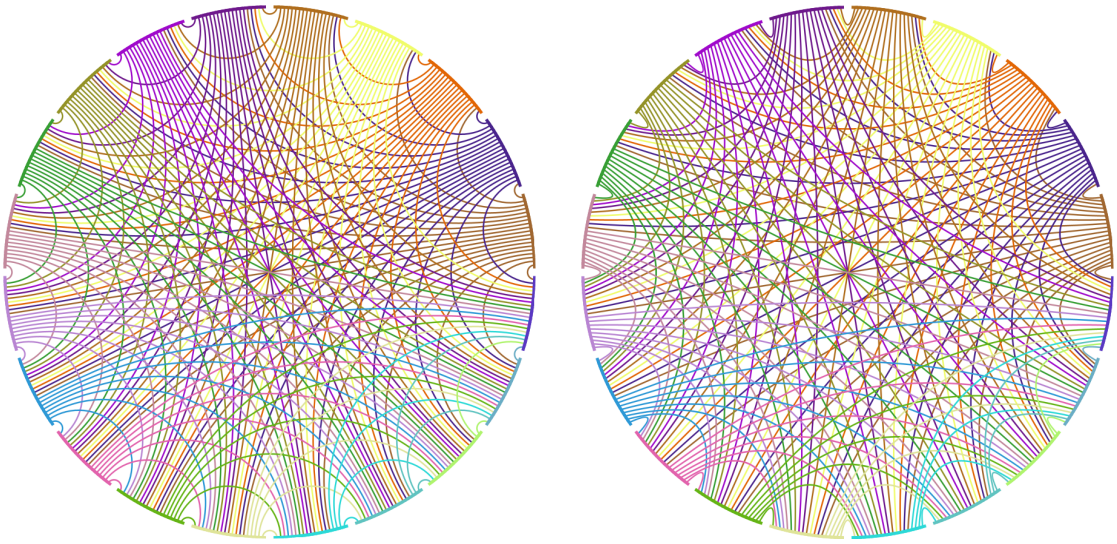
The above mentioned is an advantage of the straight-line drawing style. On the other side, the drawing style has a few significant problems that can be seen in Fig. 3.4. For one, identifying connections between neighboring nodes is very straining and someone with no knowledge of their existence would most likely not notice them at all. Secondly, arguably the most important areas of the graph, the areas near nodes where one would see the connections, are noisy because chords from other nodes are drawn too closely. Finally, such chords and generally pairs of short chords that cross lead to very shallow crossing angles.

In comparison, these problems are essentially non-existent for chord diagrams drawn with circular arcs and quadratic curves of the same graph ($|V| = 20$, $|E| = 190$) as can be seen in Figures 3.5a and 3.5b.

3.2 Circular-Arc Drawing Style

In this section we present our Circular-arc drawing style and prove that diagrams drawn with it do not have unnecessary intersections.

Definition 3.3 (Circular-arc Chord Diagram)**.** *A* circular-arc chord diagram *is a chord diagram with chords represented by circular arcs. The circular arcs are drawn as follows:* (1) *Compute the tangents to the diagram at the end-points of the chord.* (2) *Compute the intersection point* c *of the tangents.* (3) *Draw a circular arc with center* c *that connects the two end-points of the chord. Circular arcs with infinite radius are drawn as straight-line segments connecting the endpoints of the chord.*



(a) Circular-arc drawing of K_{20}.

(b) Quadratic-curve drawing of K_{20}.

Fig. 3.5: K_{20} drawings with our proposed drawing styles.

See Fig. 3.6 to see how circle chords are drawn.

Theorem 3.4. *When transforming a chord diagram with straight-line chords into a circular-arc chord diagram, the crossing-pairs remain exactly the same (no missing crossings, and more importantly, no additional crossings).*

Proof. Looking at the chords in a to-be-transformed straight-line chord diagram pairwise, there are two possibilities: *(1)* the chords cross (the end-points alternate on the circle) or *(2)* they do not cross.

(1) Consider Fig. 3.7 for this case. Say, we have a chord diagram and points a_2, b_2, a_1 and b_1 on the diagram circumference in that order. Then, we draw chord c_a by connecting the endpoints a_1 and a_2. Now, the chord c_a encloses a region A within the diagram. Endpoint b_2 lies inside of A, while endpoint b_1 lies outside of it. Connecting endpoints b_1 and b_2 (drawing chord c_b) requires entering region A (crossing c_a). The only way to connect the endpoints while crossing c_a more than once is to leave and enter region A at least once more, which would bring the total number of crossings to at least three. Circles, however, cross each other at most twice, hence circular chords cannot cross more than twice either. This proves that circular chords with alternating endpoints cross exactly once.
(2) This case has two sub-cases: *(a)* one chord "wraps" the other, that is, the smaller circle circumference section that one chord occupies lies completely within the section of the other (see Fig. 3.8a), or *(b)* the circle sections the chords take up are disjoint (see Fig. 3.8b).
(2a) Consider Fig. 3.8a for this case. Let c_1 be the larger chord circle and c_2 the smaller

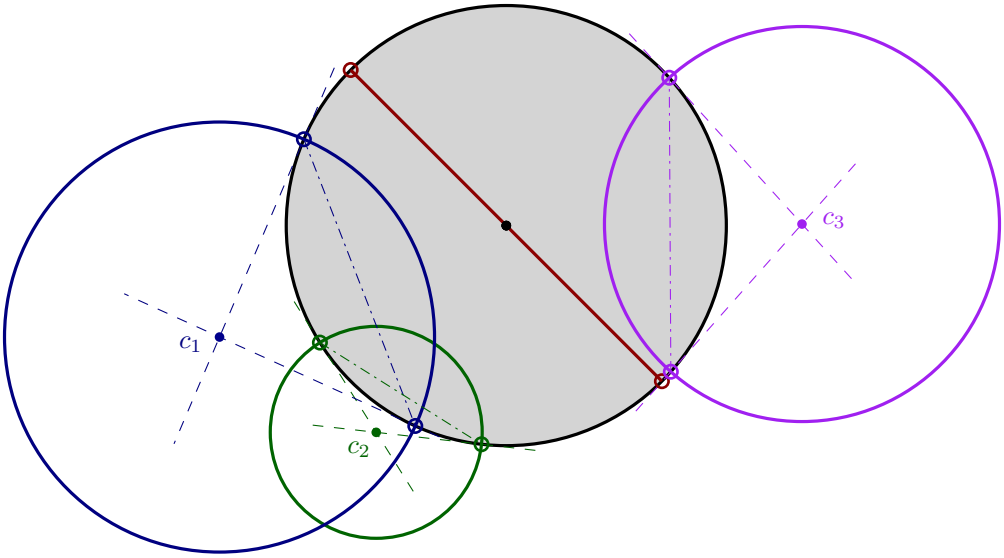


Fig. 3.6: Chord Construction Example. The dash-dot lines show where the straight chords would be.

chord circle. Then let us increase c_2 in size by moving one of the endpoints closer to the nearest endpoint of c_1 so that the distance between the two endpoint pairs is equal (see Fig. 3.9). Since the increased circle c_2' contains c_2 (c_2' touches $c_2 \Rightarrow c_2'$ and c_2 cannot cross), it suffices to prove that c_1 and c_2' do not intersect. To this end, we show that

$$r_1 > r_2 + d_c \tag{3.1}$$

with r_1 and r_2 being the radii of c_1 and c_2' respectively, and d_c being the distance between c_1 and c_2'. First, we find d_c by subtracting the length of the hypotenuse of the triangle $C_0R_2C_2$ from the length of the hypotenuse of the triangle $C_0R_1C_1$ (see Fig. 3.10).

$$d_c = \sqrt{r_1^2 + 1} - \sqrt{r_2^2 + 1} \tag{3.2}$$

Then we plug Eq. 3.2 into Eq. 3.1:

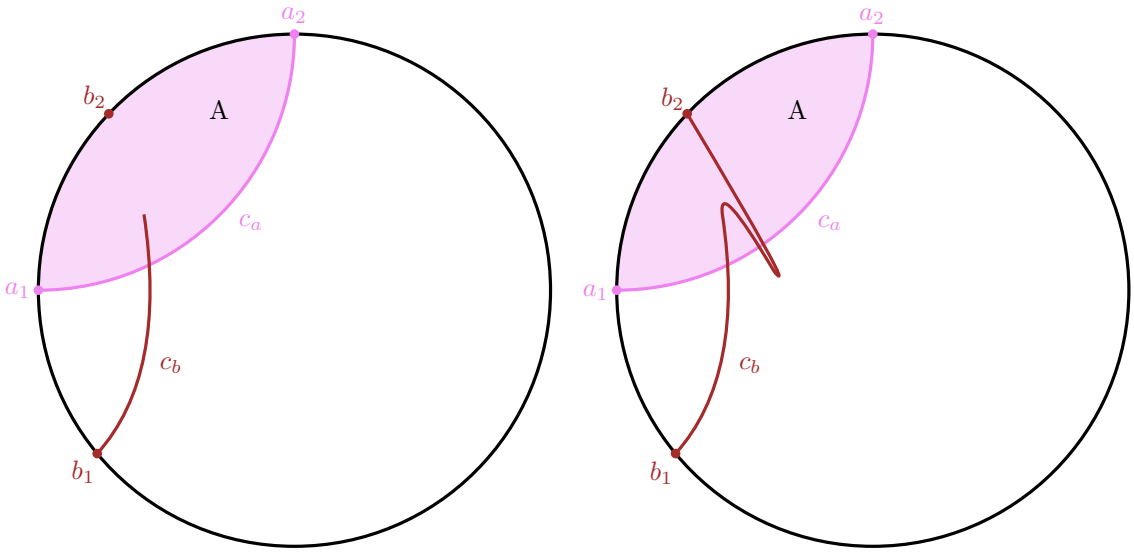
$$r_1 > r_2 + \sqrt{r_1^2 + 1} - \sqrt{r_2^2 + 1} \tag{3.3}$$

and transform it to obtain:

$$r_1 - \sqrt{r_1^2 + 1} > r_2 - \sqrt{r_2^2 + 1} \tag{3.4}$$

Both sides of Eq. 3.4 have the form $t - \sqrt{t^2 + 1}$. Since $r_1 > r_2$, it suffices to prove that $f(t) = t - \sqrt{t^2 + 1}$ is strictly increasing:

$$f'(t) = 1 - \frac{2t}{2\sqrt{t^2 + 1}} = 1 - \frac{t}{\sqrt{t^2 + 1}} = \frac{\sqrt{t^2 + 1} - t}{\sqrt{t^2 + 1}} \tag{3.5}$$



(a) Circular chords with alternating endpoints have to cross at least once.

(b) Chords with alternating endpoints have to cross an odd number of times.

Fig. 3.7: Proof of Theorem 3.4.

For every $t \in \mathbb{R}$, we have $\sqrt{t^2+1} > 0$ and $\sqrt{t^2+1} > t$. Thus $f'(t) > 0$ for every t. Therefore

$$r_1 > r_2 \Rightarrow f(r_1) > f(r_2) \tag{3.6}$$

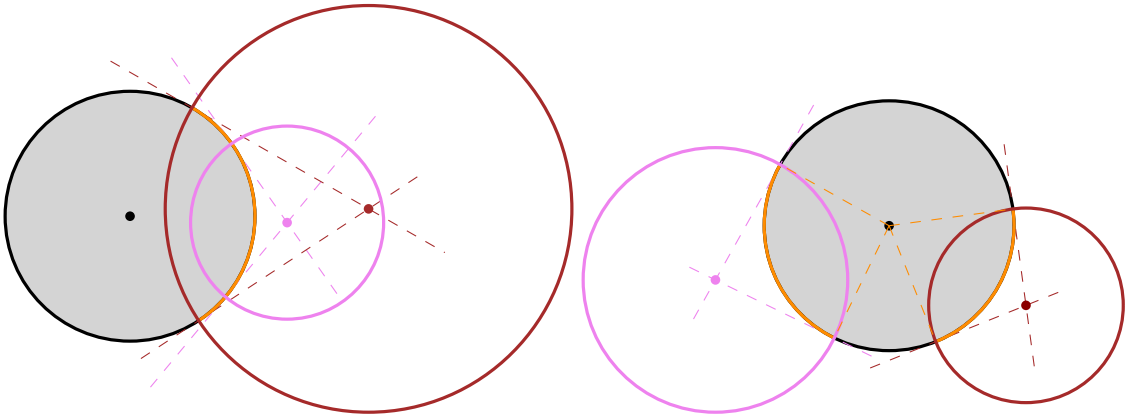
(2b) In this case, we know that the circle sections that the chords contain (marked in orange in Fig. 3.8b) do not overlap. This means that we can draw two radii to the endpoints of each arc, creating two disjoint sectors. The radii of the sectors are tangential to their respective chord-circle. This means that the sectors fully contain their circle, which in turn means that the circles and therefore the chords do not intersect. □

3.3 Quadratic-Curve Drawing Style

In this section we present our quadratic-curve drawing style and attempt to prove that diagrams drawn with it do not have unnecessary intersections.

The first steps of constructing the quadratic-curve drawing style, included reconstructing the chord diagram drawing style used by ChordLink [ADM+19, ADM+22] that already used quadratic Bézier curves for the chords.

Initially, observing chord diagram drawings where the total node weight was quite evenly distributed among all nodes, it seemed like their drawing style was already achieving what we were trying to do. But considering the chord construction of their drawing style it seemed quite unlikely. Without going into too much detail, most quadratic Bézier curves in their drawing style are constructed by placing the outer control points at the



(a) One chord "wraps" the other.

(b) Disjoint circle sections.

Fig. 3.8: Two Possible Layouts.

endpoints of the chord and by placing the middle control point in the middle of the diagram circle (see Fig. 3.11a). In some cases this construction leeds to unnecessary crossings of chords which can be seen in Fig. 3.11b. Looking at the figure it is quite clear that the control polygon of the smaller chord, the shape of which it is mimicking is too tall. So our idea of tackling this problem was to make sure that the height of a control polygon would be proportional to the distance between the nodes of a chord. That way smaller chords like the purple chord in Fig. 3.11b, the endpoints of which could fit within larger chords (cyan), would (hopefully) always be small enough not to cross the larger chord.

Definition 3.5 (Quadratic-curve Chord Diagram)**.** *A* quadratic-curve chord diagram *is a chord diagram with chords in the form of quadratic Bézier curves. The three control points of each quadratic curve are assigned as follows: (1) The endpoints of the curve correspond to the endpoints of the chord as is. (2) The middle control point lies on the bisector of the (smaller) circle sector between the endpoints with distance:*

$$d = r \cdot \left(1 - \sqrt{\frac{\textit{distance between endpoints in radian}}{\pi}}\right) \tag{3.7}$$

from the center of the diagram.

In simple words, *(2)* means: the further the endpoints, the closer the control point to the center, the taller the curve, and vice versa.

See Fig. 3.12 to see how quadratic chords are constructed.

Theorem 3.6. *When transforming a chord diagram with straight-line chords into a quadratic-curve chord diagram, the crossing-pairs remain exactly the same.*

Proof. Looking at the chords in a to-be-transformed straight chord diagram pairwise, there are two possibilities: *(1)* the chords cross (the end-points alternate on the circle)

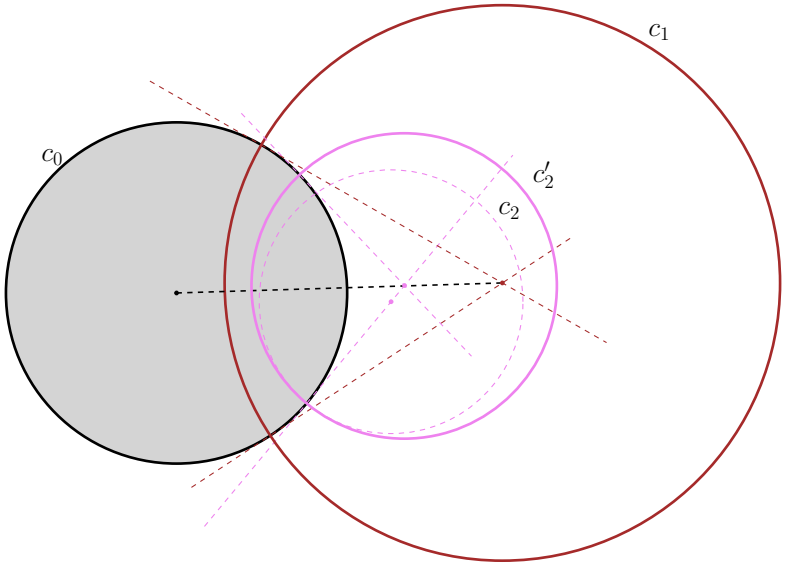


Fig. 3.9: Increasing the size of the smaller circle c_2 to align the centers of c_1 and c_2'.

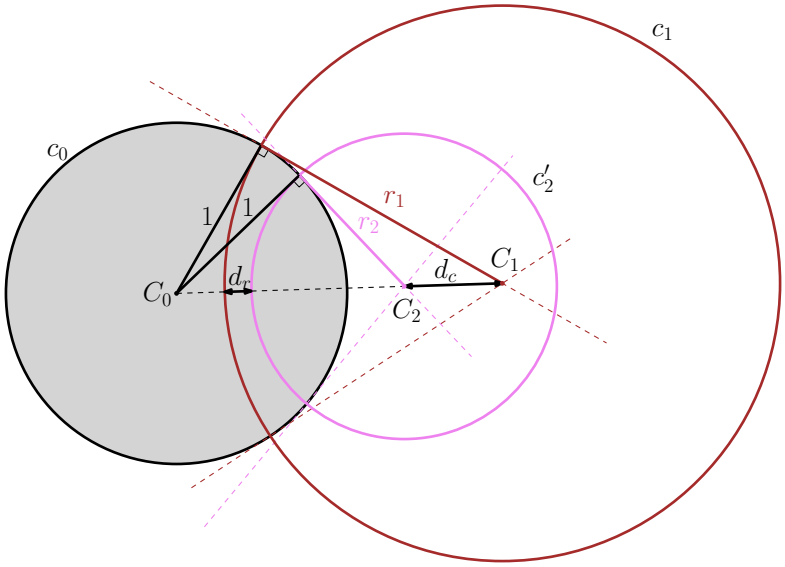
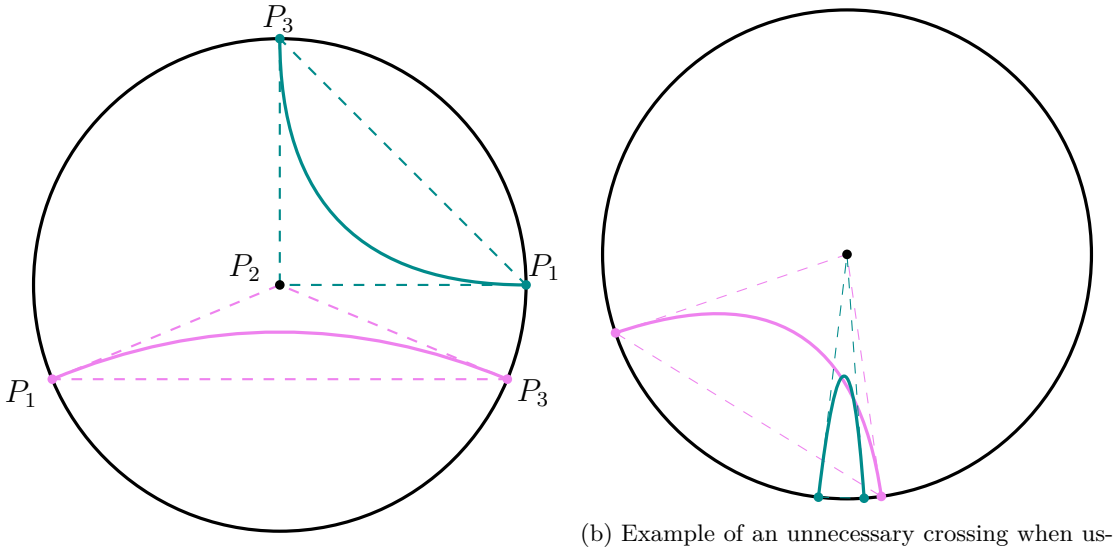


Fig. 3.10: Distance between circular arcs.



(a) Chord construction example in ChordLink.

(b) Example of an unnecessary crossing when using the ChordLink construction method.

Fig. 3.11: ChordLink examples.

or *(2)* they do not cross.
(1) This is an assumption we have to make. It seems to be almost obvious, but proving it turned out to be quite difficult.
(2) This case has two sub-cases: *(a)* one chord "wraps" the other, meaning the smaller circle circumference section one chord takes up completely lies within the section of the other (see Fig. 3.13a), *(b)* or the circle sections the chords occupy are disjoint (see Fig. 3.13b).
(2a) Consider Fig. 3.13a for this case. Let q_1 be the larger chord and q_2 the smaller chord. Then let us increase q_2 in size by moving one of the end-points closer to the nearest endpoint of q_1 so that the distance between both endpoint pairs is equal (see Fig. 3.14a). The now increased chord q_2' fully contains q_2, or rather that is another assumption we have to make.

The assumption is based on an observation that can be seen in Figure 3.14b. The quadratic curves, as we have mentioned in Section 2.2, are parts of parabolas. So if we were to slightly increase the circle and let the parabolas extend further, we would have the exact same curves now with alternating endpoints on the circle. And assuming *1* is true, these curves only cross once, meaning that in the original, smaller circle, these curves only meet at the endpoints. Which would prove that q_2' fully contains q_2.

Now, assuming the above is true, it is enough to prove that q_1 and q_2' do not cross. Which is now fairly easy, because the curves now share a symmetry axis and therefore behave like functions describing parabolas in a shared coordinate system. In such a case, parabolas can only have two intersections (or none). Which in our case can only happen within the circle if the vertex of parabola q_2' were closer to the center than the vertex of q_1 (see Fig. 3.15).

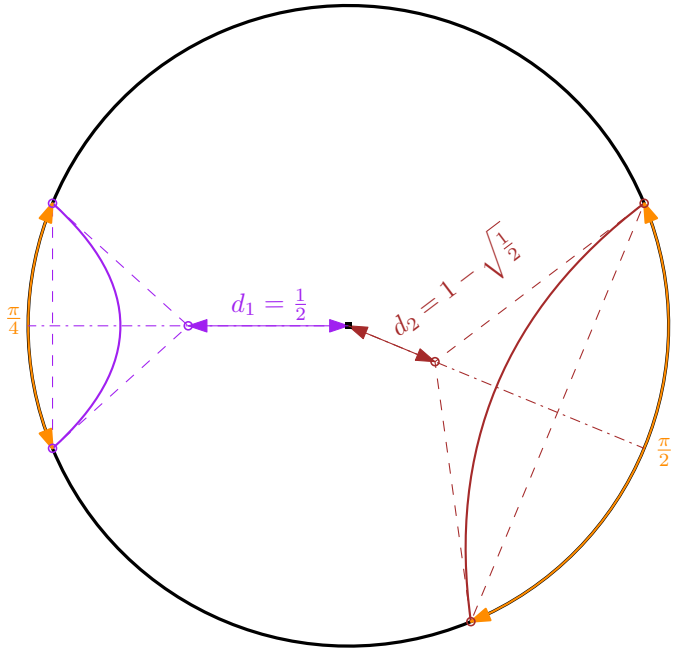


Fig. 3.12: Circular-arc chord construction example.

So we want to prove that:

$$v_1 < v_2 \tag{3.8}$$

Given that the vertices lie on the x-axis, to find v_1 and v_2 we have to find the x-component of the corresponding Bézier equation at $t = 0.5$. The x-component is:

$$x(t) = (1-t^2)A.x + 2(1-t)tB.x + t^2C.x \tag{3.9}$$

Plug $t = 0.5$ into 3.9:

$$v_i = x(0.5) = (1-0.5)^2A_i.x + 2(1-0.5)0.5B_i.x + 0.5^2C_i.x \tag{3.10}$$

$$v_i = x(0.5) = 0.25A_i.x + 0.5B_i.x + 0.25C_i.x \tag{3.11}$$

Insert Eq. 3.11 into Eq. 3.8 :

$$0.25A_1.x + 0.5B_1.x + 0.25C_1.x < 0.25A_2.x + 0.5B_2.x + 0.25C_2.x \tag{3.12}$$

Since the endpoints of q_1 are further apart than the endpoints of q_2' we know that:

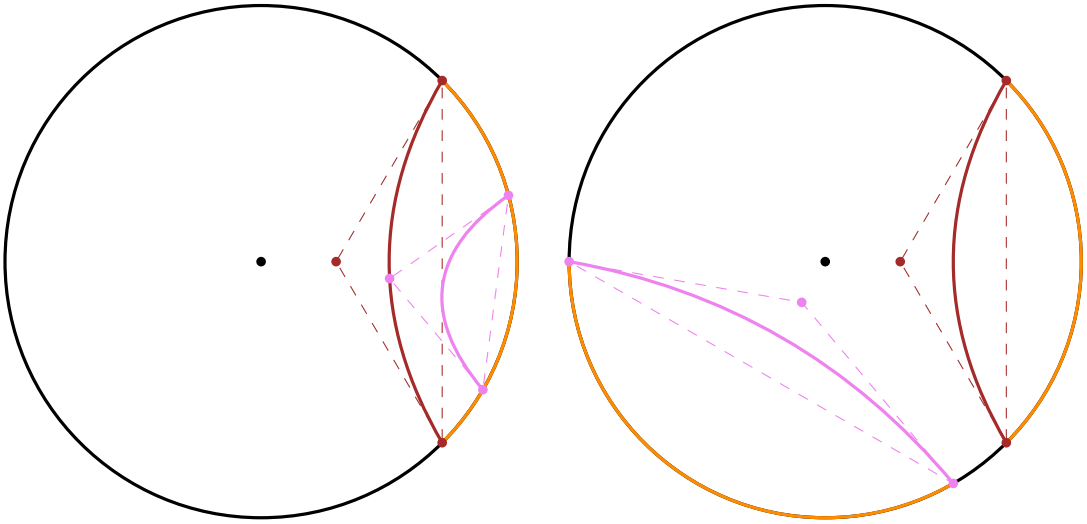
$$0.25A_1.x < 0.25A_2.x \tag{3.13}$$

and

$$0.25C_1.x < 0.25C_2.x \tag{3.14}$$

From our definition of the middle control point in Def. 3.5 additionally follows:

$$0.5B_1.x < 0.5B_2.x \tag{3.15}$$



(a) One chord "wraps" the other. (b) Disjoint circle sections.

Fig. 3.13: Two possible layouts.

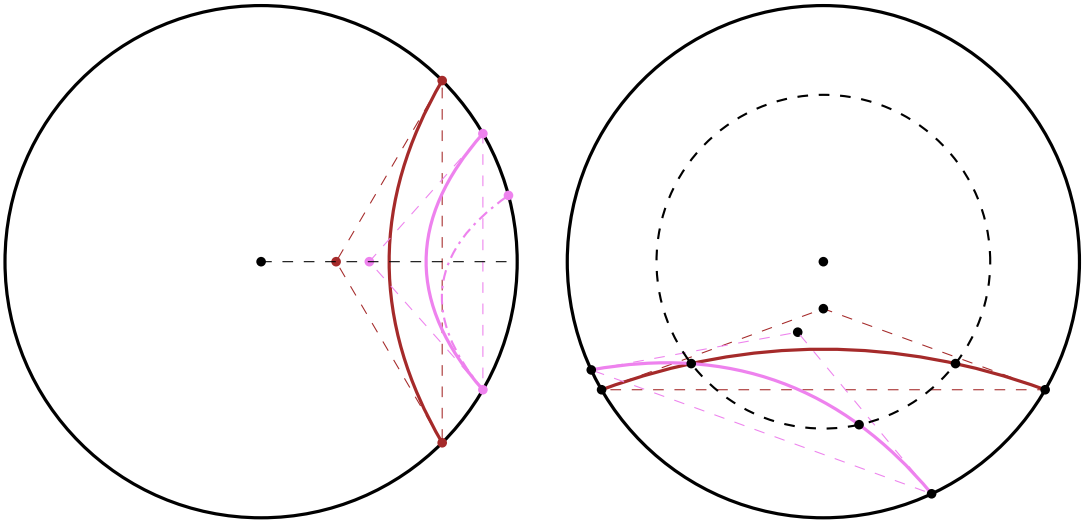
Thus proving Eq. 3.12 and ultimately proving that chords do not cross in this case.
(2b) In this case we know that the circle sections that the chords contain (marked in orange in Fig. 3.13b) do not overlap. This means that we can draw 2 radii to the end-points of each chord, creating 2 disjunct sectors. Our construction ensures that the control polygon of a curve completely lies within such a sector. The Convex Hull property of Bézier curves in turn ensures that quadratic curves lie completely within their control polygon. In sum, disjunct sectors ⇒ disjunct control polygons ⇒ no intersections. □

Finding Intersections Before we move to the next chapter where we compare the intersection angles of the different drawing styles, we first need to talk about finding intersections of quadratic Bézier curves as this is not very straightforward. So in this section we, first, shortly talk about the motivation for an intersection-finding algorithm and then explain said algorithm.

Finding intersections of quadratic Bézier curves analytically is a bit of challenge. Since two quadratic Bézier curves generally can have up to 4 intersections (see Fig. 3.16), finding these intersections exactly would require solving a polynomial of 4th degree.

Quite a few approximation algorithms that address this problem exist already. Among the popular are: subdivision [LR80], internal subdivision [KM83], Bézier clipping [SN90] and implicitization [SP86]. These algorithms, however, were designed for general Bézier curves of varying degrees. We wanted to use one that was specially designed for our construction and took advantage of it. So we made our own algorithm, that makes use of the fact that our drawing style uses exclusively symmetric quadratic Bézier curves, and of their position relative to the circle.

The idea of the algorithm (2) is to search the part of the circle we know the intersection



(a) Increased chord size. (b) Observation of chords that share one endpoint.

Fig. 3.14: Proof 2(a).

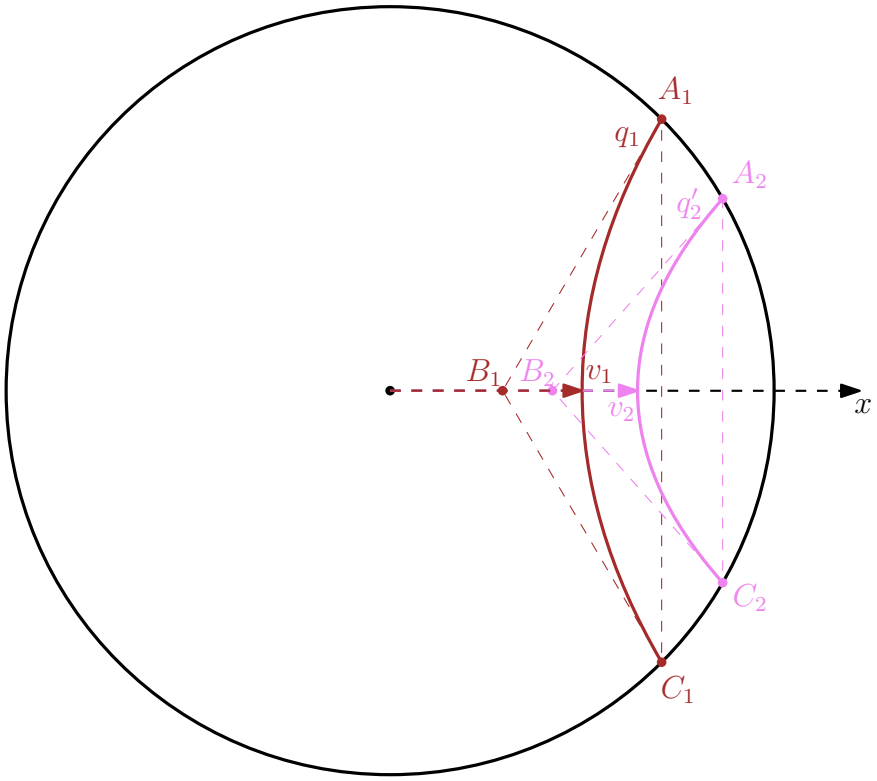


Fig. 3.15: Vertex vectors.

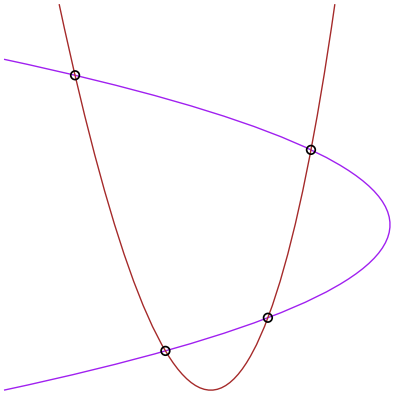


Fig. 3.16: Example of two quadratic Bézier curves with four intersections.

has to be within, with rays from the circle center c. The rays intersect the curves at different distances, except for the intersection point. So finding the intersection comes down to finding the point where both curves have equal distance to c (see Figures 3.17 and 3.18).

This method may seem complex at first, but it is not so without reason. The reason we want to compare the distances to c is because we can rotate the coordinate system (CS) around c without changing the distances. And if we can freely rotate the CS, we can use this to align it with the symmetry axis of the quadratic Bézier curves, so that they can be treated as valid functions. This means that the quadratic curves, which, earlier in Section 2.2, we established are parts of parabolas, can now be represented with the form $f(x) = ax^2 + bx + c$. This makes finding the intersections of the curves with rays very easy and accurate. Which is ultimately why we are doing it this way.

Let us go through the algorithm in detail.

The algorithm receives as input: *(1)* the control points of the curves, *(2)* the initial step size t in radian, which dictates how much the rays will move clockwise or counter-clockwise within the the boundary, *(3)* the step reduction factor q, which is the factor by which t decreases when necessary, and lastly, *(4)* the acceptable difference between two intersection points Δd, which determines when the algorithm ends

Lines 1 to 7 is the initialization part of the algorithm: *(1)* we determine the intersection boundary of the curves, which are the inner endpoints of two curves (see Fig. 3.17), *(2)* place the ray cursor at $c = B + t$, *(3)* set $reverse$ to $false$, which will be necessary to differentiate between different cases, *(4)* initialize the radii difference at $\Delta r = \infty$, *(5)* determine how much the CS has to be rotated counter-clockwise to align the y-axis with the symmetry axis of each curve, *(6)* calculate the new coordinates of the control points relative their new CS, and lastly, *(7)* we determine the quadratic equations of the parabolas with the new control points.

The main part of the algorithm (l.9-31) is a loop that stops only when Δd is satisfied. At the beginning of each loop (l.9-12) the cursor coordinates are translated to the rotated CSs, with which the linear equations of the ray are determined. After (l.13-15), the algorithm finds the intersections $P_{int_{1/2}}$ of the ray with the curves, determines the distances

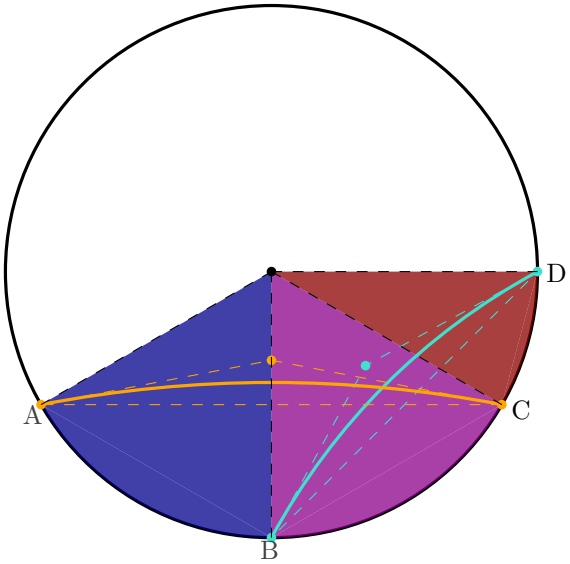


Fig. 3.17: Example of an intersection boundary. The intersection of the curves has to lie in the overlapping area (purple) of the sectors containing the curves (blue+purple and red+purple).

$r_{1/2}$ between $(0, 0)$ and $P_{int_{1/2}}$ and calculates their difference $\Delta r' = r_2 - r_1$. Then the algorithm needs to determine in which scenario it is and act accordingly: *(1)* the ray can be approaching the intersection clockwise (l.22) or counter-clockwise (l.29), in which case the ray keeps moving as it was, *(2)* the ray can shoot past the intersection clockwise (l.27) or counter-clockwise (l.20), but still be closer to the intersection, in which case the cursor needs to move in the opposite direction with reduced step size t, or *(3)* the ray can skip the intersection clockwise (l.31) or counter-clockwise (l.24) and be further from intersection than before, in which case the ray needs move to its previous location and then move forward with reduced step size. The algorithm ends when $\Delta r' \leq \Delta d$ (l.16) and returns the intersection point of the ray with one of the curves translated back to the original CS.

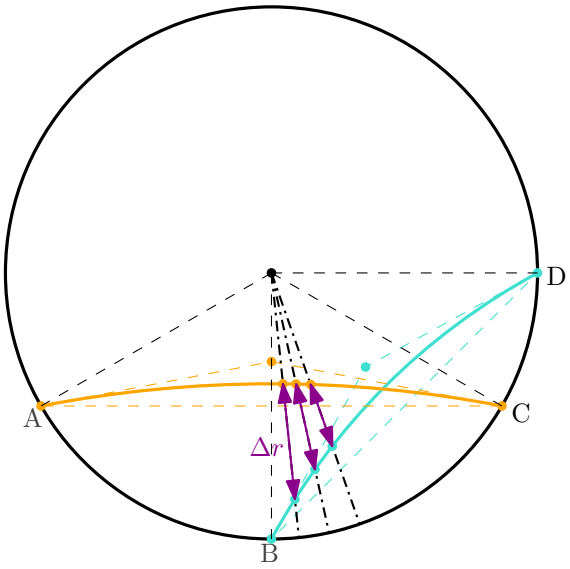


Fig. 3.18: Examples of ray intersections with two curves. At the intersection point of the two curves $\Delta r = 0$.

Algorithm 1: OrderChords(Node[] nodes)

```
1  foreach node ∈ nodes do
2      lefts ← ∅
3      rights ← ∅
4      mirror ← node.arcMiddle + π
5      if mirror > 2π then
6          mirror ← mirror − 2π
7      if mirror < node.startAngle then
8          foreach n ∈ node.connectedNodes do
9              if mirror < n.endAngle ≤ node.startAngle then
10                 add n to lefts
11             else
12                 add n to rights
13         sort lefts in dsc order     // the lists are sorted by startAngle
14         rights_1 ← {n ∈ rights | n.startAngle < node.startAngle}
15         sort rights_1 in dsc order
16         rights_2 ← {n ∈ rights | n.startAngle > node.startAngle}
17         sort rights_2 in dsc order
18         rights ← rights_1 ∪ rights_2
19     else
20         foreach n ∈ node.connectedNodes do
21             if node.endAngle ≤ n.startAngle < mirror then
22                 add n to rights
23             else
24                 add n to lefts
25         sort rights in dsc order
26         lefts_1 ← {n ∈ lefts | n.startAngle < node.startAngle}
27         sort lefts_1 in dsc order
28         lefts_2 ← {n ∈ lefts | n.startAngle > node.startAngle}
29         sort lefts_2 in dsc order
30         lefts ← lefts_1 ∪ lefts_2
31     ordered ← lefts ∪ rights
32     node.connectedNodes ← ordered
```

Algorithm 2: FindIntersection()

Input: arrays Q_1, Q_2 with the control points of the two curves, step size t, step reduction factor q, acceptable distance Δd between two intersection points

Output: accepted intersection Point P_{int}

determine intersection boundary $B < C$
$c = B + t$ `// ray endpoint cursor`
$reverse = false$
$\Delta r = \infty$
determine rotation ϕ_1, ϕ_2 to align coordinate system with symmetry axis of Q_1, Q_2
calculate Q_1', Q_2' with $P_{1,i} = p(P_{1,i}, \phi_1)/P_{2,i} = p(P_{2,i}, \phi_2)$, $p(P, \phi) = \binom{x\cos\phi - y\sin\phi}{y\cos\phi + x\sin\phi}$
determine $f_{1/2} = ax^2 + bx + c$ with Q_1',Q_2'

while true **do**
  $P_c = (\cos c, \sin c)$ `// cursor in` (x, y)`-coordinates`
  $P_{c_1} = p(P_c, \phi_1)$, $P_{c_2} = p(P_c, \phi_2)$ `// cursor for each cs`
  determine $g_{1/2} = mx + b$ with $(0, 0)$ and P_{c_1}/P_{c_2}
  determine $P_{int_{1/2}}$ with $f_1 = g_1/f_2 = g_2$ within the circle
  $r_{1/2} = \sqrt{x_{1/2}^2 + y_{1/2}^2}$ with $P_{int_{1/2}}$ `// distance from` $(0, 0)$ `to crossing point`
  $\Delta r' = r_2 - r_1$
  if $|\Delta r'| \leq \Delta d$ **then**
    return $p(P_{int_1}, 2\pi - \phi_1)$`// return intersection translated back to original CS`
  else if $\Delta r' > 0$ **and** $\Delta r' \leq \Delta r$ **then**
    if $reverse$ **then**
      $t = t/q$, $c = c + t$, $\Delta r = \Delta r'$, $reverse = false$
    else
      $c = c + t$, $\Delta r = \Delta r'$
  else if $\Delta r' > 0$ **and** $\Delta r' > \Delta r$ **then**
    $c = c + t$, $t = t/q$, $c = c - t$
  else if $\Delta r' < 0$ **and** $|\Delta r'| \leq \Delta r$ **then**
    if not $reverse$ **then**
      $t = t/q$, $c = c - t$, $\Delta r = |\Delta r'|$, $reverse = true$
    else
      $c = c - t$, $\Delta r = |\Delta r'|$
  else if $\Delta r' < 0$ **and** $|\Delta r'| > \Delta r$ **then**
    $c = c - t$, $t = t/q$, $c = c + t$

4 Comparison and Discussion

In this chapter we compare and discuss our drawing styles and a straight-line approach, by analyzing how the chord diagrams, drawn by the them, perform intersection-angle-wise on a set of graphs.

4.1 Data

The set of graphs for our analysis was acquired from the online database House of Graphs [CDG23] which, as the database itself puts, contains interesting graphs. A graph on their database is considered "interesting" if at least one user thought it qualified as such.

To avoid unrealistic graphs as much as possible, in the context of chord diagrams, the graphs were filtered by number of vertices and minimum and average vertex degree. The exact query of graphs:

- G is connected
- $5 \leq |V| \leq 20$
- $\forall v \in V : \deg(v) \geq 2$
- $\frac{1}{|V|} \sum_{v \in V} \deg(v) \geq 3$

resulted in about 6000 graphs, but was limited to the first 2000 as a download, as a per user limitation from the side of House of Graphs.

4.2 Results

In this section we present the results and takeaways from comparing pairwise (Circle vs. Straight, Bézier vs. Straight, Circle vs. Bézier) the mean, the median and the minimum intersection angle for each graph of the dataset, as well as analyzing the intersection-angle-distributions for each drawing style. More precisely, for each pair we plot the difference of mean, median and minimum angle (separately on three different scatterplots) for the same graph. This way each scatterplot contains 2000 datapoints that show by how much drawing style x performed better or worse than drawing style y on the same graph for the chosen metric. Additionally, a line-plot contains the intersection-angle-distribution (in 10° bins) over all graphs for each of the styles.

Circle vs. Straight: *Mean(1)* (see Fig. 4.1): The mean angle difference varies greatly (from $-6°$ to $40°$) for graphs with less than 300 crossings, but ultimately favors the circle drawing style. For diagrams with more than 300 crossings the mean angle is very slightly favoring diagrams with straight chords. By looking at one of the diagrams where the circle drawing style is heavily favored in Fig. 4.8, we can see that this comes down to there being many intersections of shorter chords. Over all graphs, the circle drawing style performed better in 1436 (72%) of them in terms of mean angle, and is on average about $3.4°$ better in circle diagrams. *Median(2)* (see Fig. 4.2): Apart from the variance being greater (from $-14°$ to $48°$), the median angle difference plot is very similar to the mean difference plot, with roughly the same trends and groupings. The median advantage can be seen well in Fig. 4.9, where most intersection angles in the circle diagram are about $70°$, while the majority in the straight diagram are $36°$. Over all graphs, the circle drawing style performed better in 1219 (61%) of them in terms of the median angle and is on average about $3.25°$ better in circle diagrams. *Min(3)* (see Fig. 4.3): Similarly, for lower crossing numbers, the minimum angle difference varies quite a lot (from $-21°$ to $31°$), and still mostly favors the circle drawing style, but this time a small number of graphs quite heavily favor the straight drawing style. In Fig. 4.10 can be seen an example that heavily favors the circle drawing style, and in Fig. 4.11 the opposite. What also differs, is that with an increasing number of crossings the minimum angle does not turn in favor of straight lines. Over all graphs, the circle drawing style performed better in 1469 (73%) of them in terms of the minimum angle and is on average about $2.5°$ better in circle diagrams.

Bézier vs. Straight: As can be seen in Figs. 4.4, 4.5, 4.6 and Tables 4.1, 4.2, the Bézier drawing style behaves very similarly in comparison to the straight drawing style as the circle drawing style. The results and takeaways are more or less the same.

Circle vs. Bézier: As could be inferred from the previous pair, the Circle and Bézier drawing styles show very similar results, as can be seen in Tables 4.1, 4.2. The minimum angles are nearly identical and the mean and median angles slightly favor the Bézier drawing style.

By examining the line-plot in Fig. 4.7 that depicts the intersection-angle-distribution over all graphs in 10-degree bins, we can confirm our findings from before. Graphs drawn with the straight drawing style suffer primarily from having significantly more intersection angles proportionally in the $10°$ to $30°$ range (up to three times more in the $10°$-$20°$ bin and up to 33% more in the $20°$-$30°$ bin), which often leads to bad min angles. On the other side, graphs drawn with Circle and Bézier drawing styles have significantly more angles in the $40°$ to $70°$ range which on average leads to better mean and median angles.

The data on its own does not present the most convincing argument. The average advantage in the measured metrics is not hugely in favor of the circle or Bézier drawing style over the straight-line style. What the data does show, however, is that the proposed

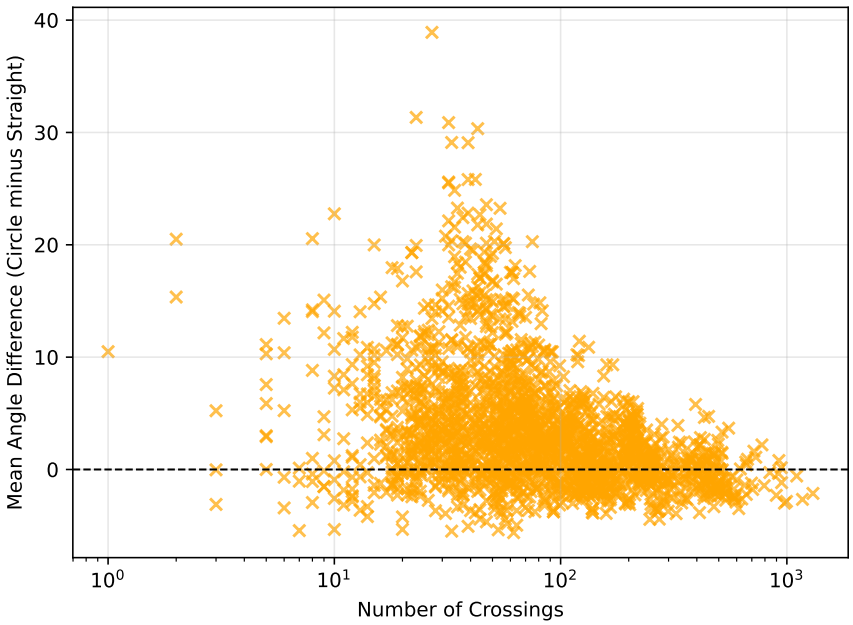


Fig. 4.1: Circle vs. Straight (Mean Angle Difference). Median difference: 2.14°.

drawing styles rarely perform worse, and when they do, it is not a by huge margin. This in combination with the fact, that the proposed drawing styles avoid the pitfalls discussed in Section 3.1, is why we believe our drawing styles to be better than drawing with straight lines.

Pair	Number of Times Mean was better	Number of Times Median was better	Number of Times Min was better
Circle vs. Straight	1436	1219	1469
Bézier vs. Straight	1466	1211	1468
Bézier vs. Circle	1402	1063	931

Tab. 4.1: Number of times (out of 2000) drawing style x performed better than y (x vs. y) in terms of mean, median and minimum angle.

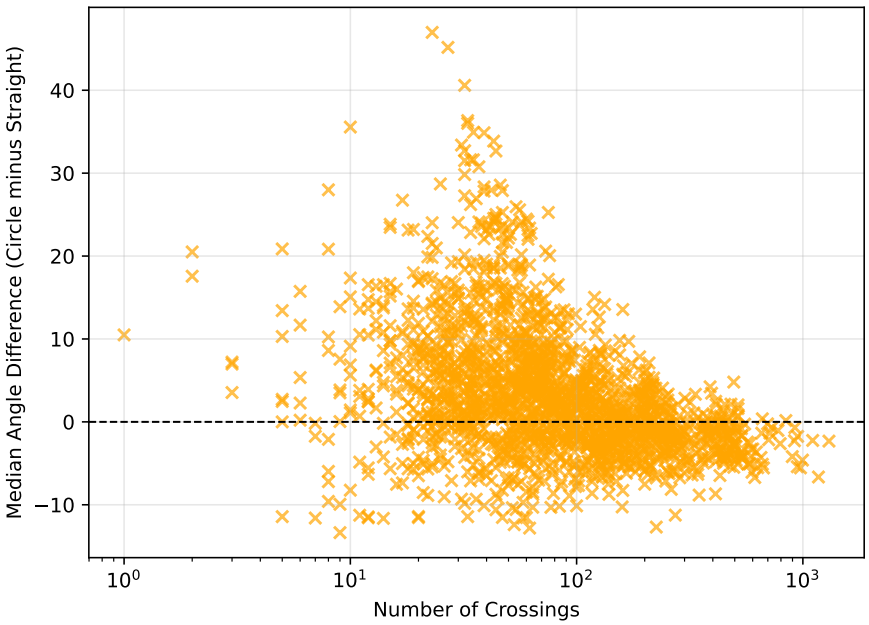


Fig. 4.2: Circle vs. Straight (Median Angle Difference. Median difference: 1.69°.)

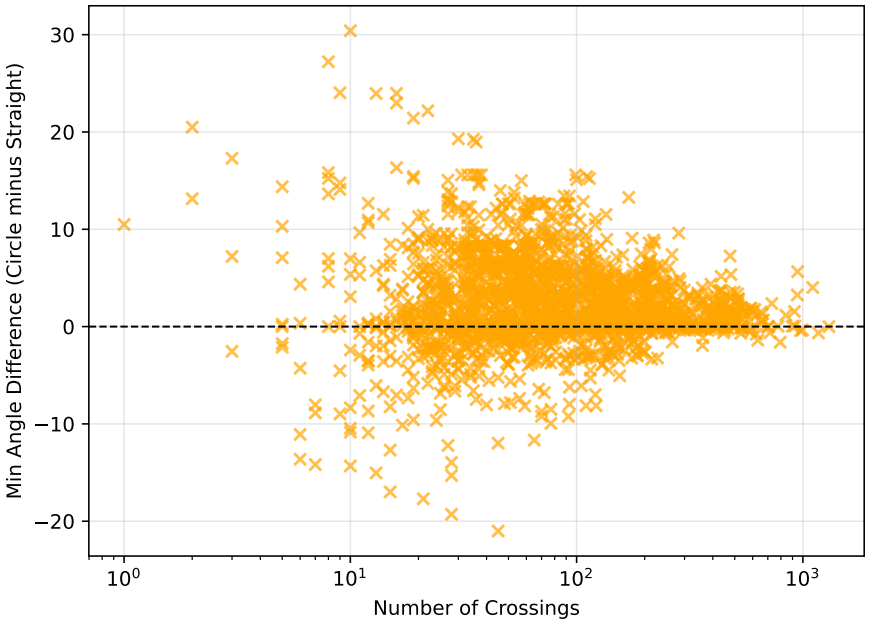


Fig. 4.3: Circle vs. Straight (Min Angle Difference). Median difference: 1.78°.

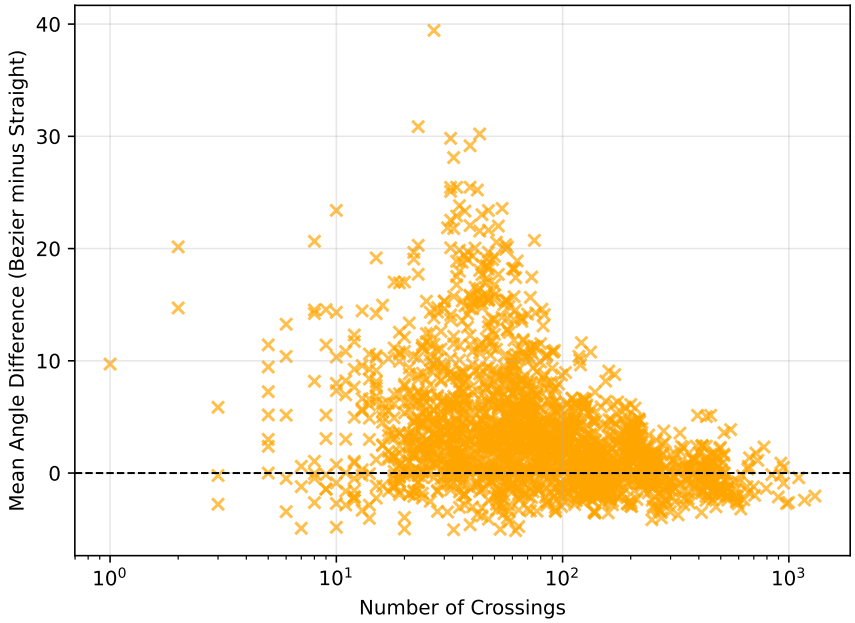


Fig. 4.4: Bézier vs. Straight (Mean Angle Difference). Median difference: 2.25°.

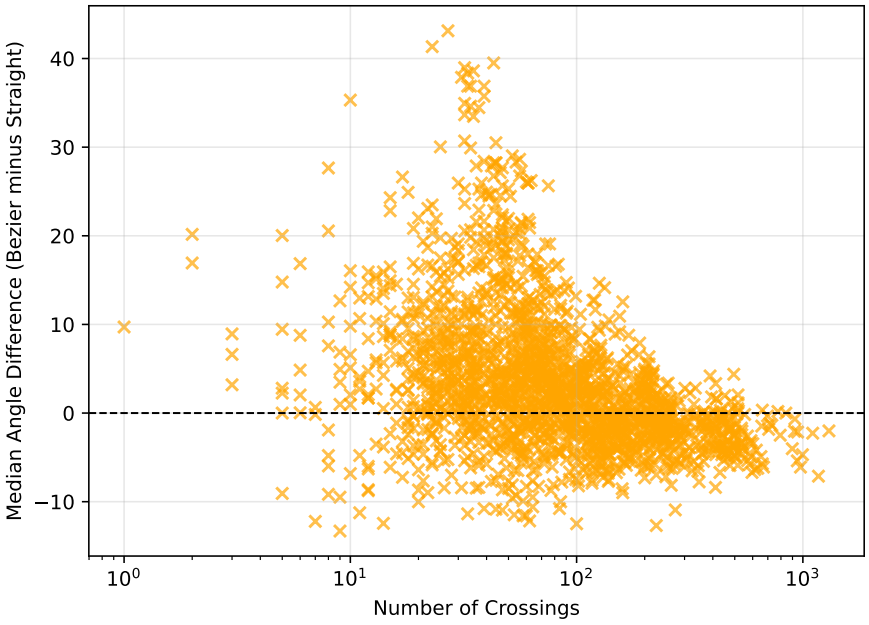


Fig. 4.5: Bézier vs. Straight (Median Angle Difference). Median difference: 1.79°.

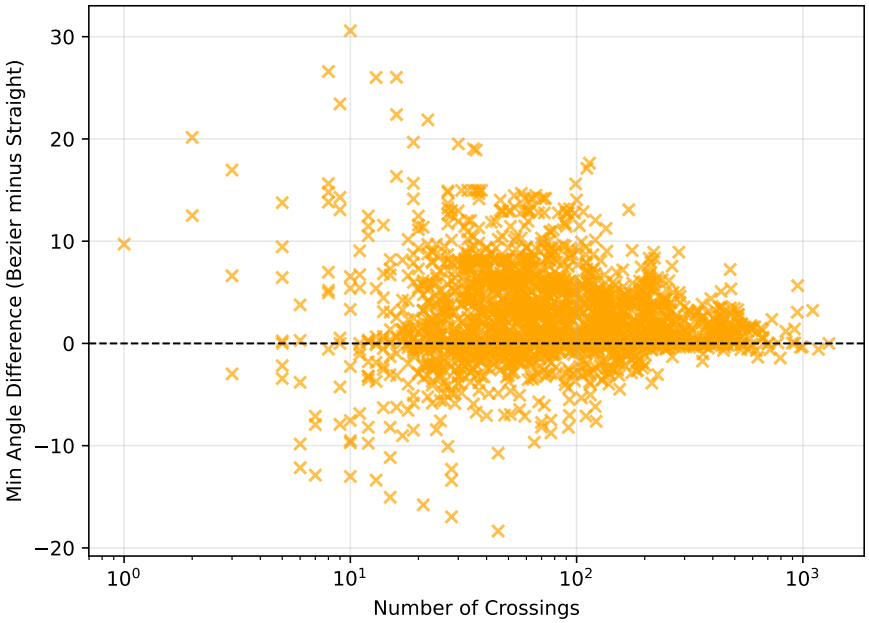


Fig. 4.6: Bézier vs. Straight (Min Angle Difference). Median difference: 1.73°.

Pair	Mean Diff in ° (Mean)	Median Diff in ° (Mean)	Min Diff in ° (Mean)
Circle minus Straight	3.42	3.25	2.52
Bézier minus Straight	3.53	3.43	2.55
Bézier minus Circle	0.11	0.18	0.03

Tab. 4.2: Differences in mean, median, and minimum values between pairs on average over all graphs in the dataset.

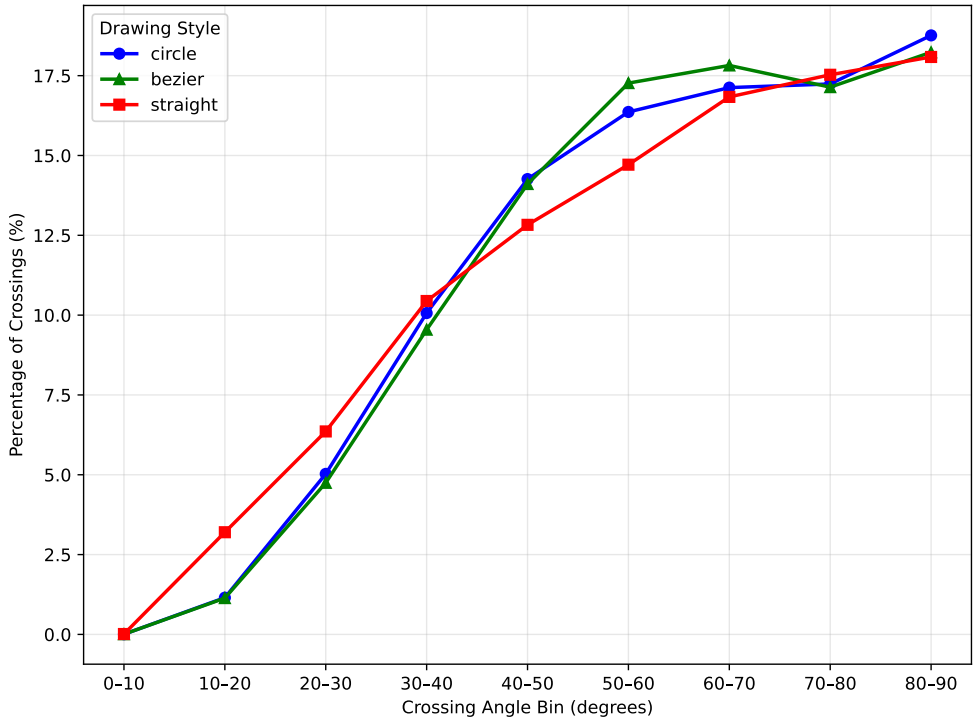
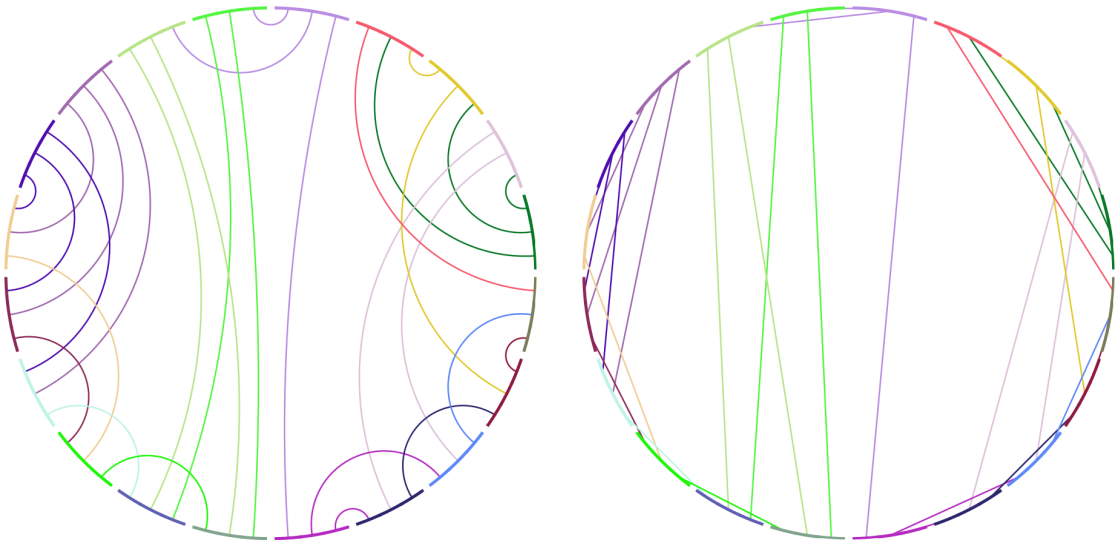


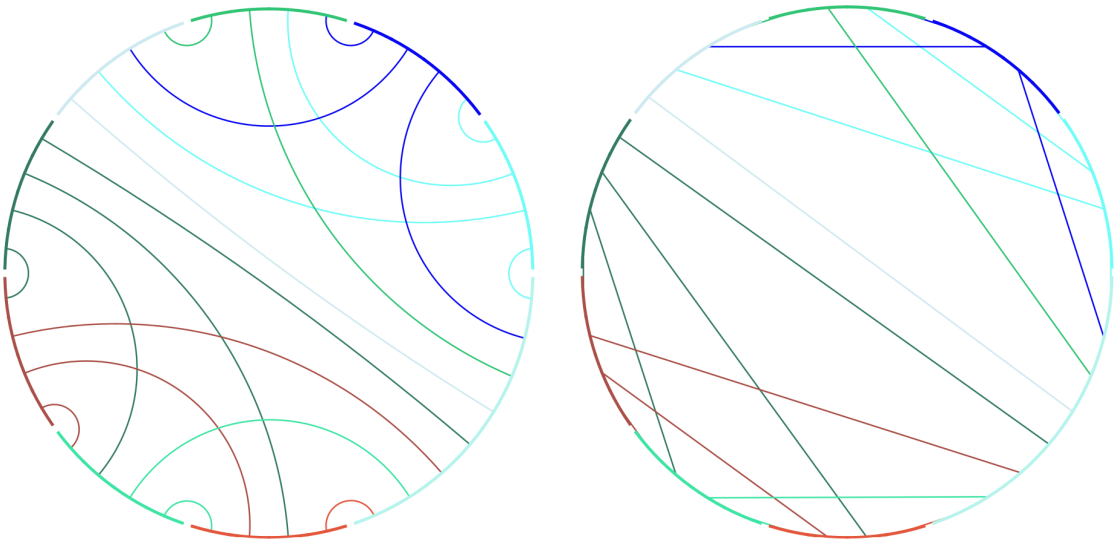
Fig. 4.7: Intersection-Angle-Distribution by Drawing Style. The line plots contains the distribution of intersection angles in 10° bins over all graphs for each of the drawing styles.



(a) Circular-arc diagram. Mean Angle: 65.72°

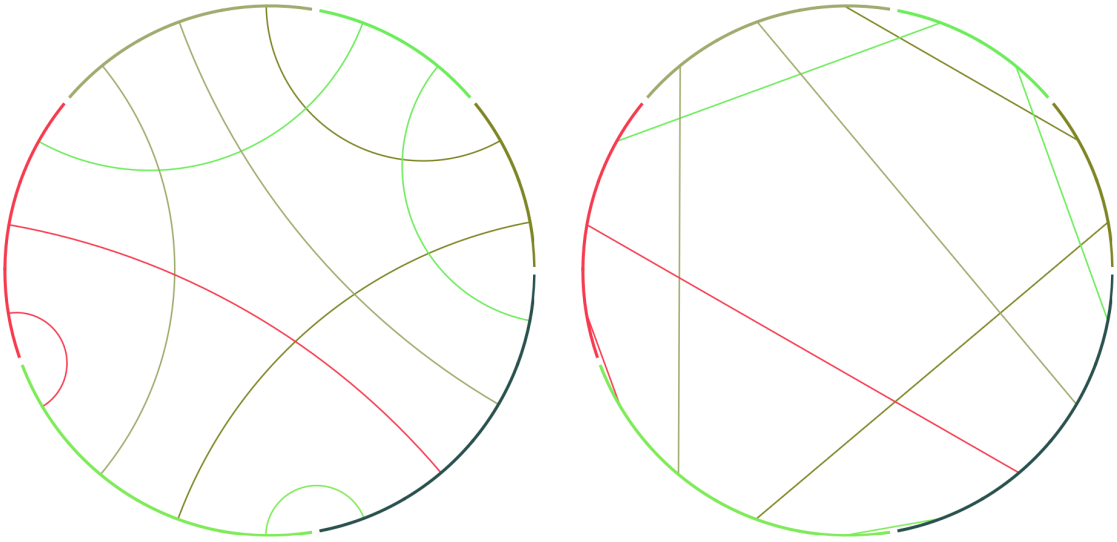
(b) Straight-line diagram. Mean Angle: 34.84°

Fig. 4.8: Graph example where a straight-line diagram performs much worse than a circular-arc diagram in terms of mean angle.



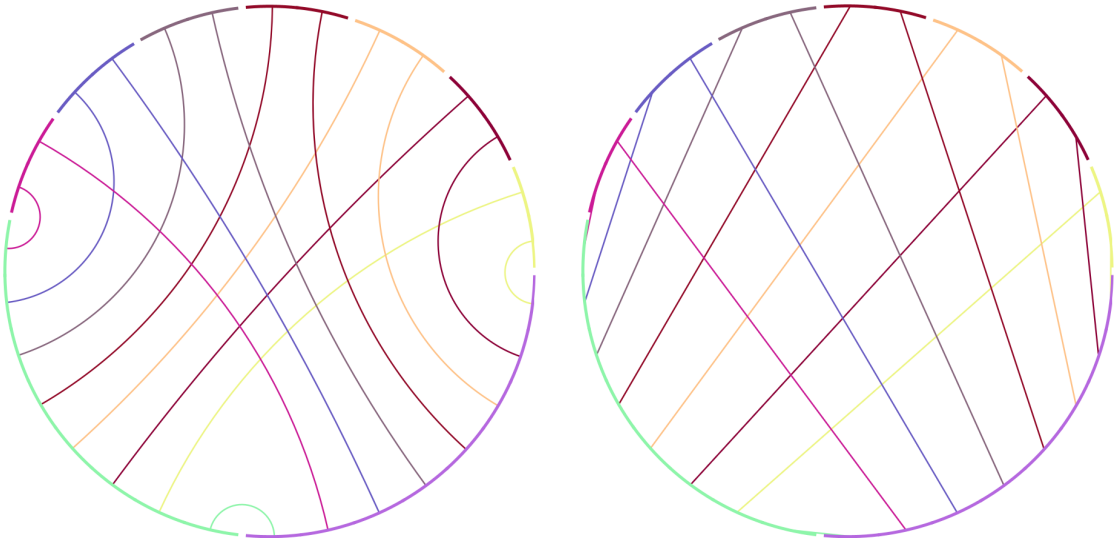
(a) Circular-arc diagram. Median Angle: 71.92° (b) Straight-line diagram. Median Angle: 36.36°

Fig. 4.9: Graph example where a straight-line diagram performs much worse than a circular-arc diagram in terms of median angle.



(a) Circular-arc diagram. Min Angle: 67.46° (b) Straight-line diagram. Min Angle: 40.24°

Fig. 4.10: Graph example where a straight-line diagram performs much worse than a circular-arc diagram in terms of min angle.



(a) Circular-arc diagram. Min Angle: 36.87°

(b) Straight-line diagram. Min Angle: 54.57°

Fig. 4.11: Graph example where a straight-line diagram performs much better than a circular-arc diagram in terms of min angle.

5 Conclusions and Future Work

In this work, we presented two drawing styles for drawing chord diagrams; one that uses circular arcs and one that uses quadratic Bézier curves for the chords. We have shown that in practice our drawing styles usually outperform a straight-line approach when it comes to intersection angles: the mean and median crossing angles increased by 3-4°.

The main potential improvement to this work would be filling the gaps in the proof for our quadratic-curve drawing style that we attempted. We are confident that the assumptions we made are correct, and thus proving them should be feasible but possibly difficult.

Another possible direction for improvement would be implementing a crossing reduction algorithm. As mentioned in Chapter 3, we assume the order of nodes is part of the input, which can lead to unnecessary crossings if the order is not optimized. Since circular crossing minimization is $\mathcal{NP}$-hard [MKNF87], calculating the perfect solution would be computationally difficult, but a heuristic such as the one of Baur and Brandes [BB05] could offer significant crossing reduction in reasonable time.

Finally, one could implement additional drawing styles using the more flexible cubic or even higher-degree Bézier curves. One direction would be to implement an algorithm that attempts to maximize the intersection angles of chords (without increasing the number of intersections) - essentially, a drawing style that attempts to draw chords such that their intersection angle is always (close to) 90°. Another direction would be a drawing style that draws chords with the specified angle of entry, that is, the angle that chords form with the bounding circle in their two endpoints. With an angle slider and live chord diagram drawing updates, one could choose a version of the diagram that one finds aesthetically most pleasing.

Bibliography

[ADM+19] Lorenzo Angori, Walter Didimo, Fabrizio Montecchiani, Daniele Pagliuca, and Alessandra Tappini: Chordlink: A new hybrid visualization model. In Daniel Archambault and Csaba D. Tóth (editors): *Graph Drawing and Network Visualization*, pages 276–290. Springer, 2019, 10.1007/978-3-030-35802-$0_2$2.

[ADM+22] Lorenzo Angori, Walter Didimo, Fabrizio Montecchiani, Daniele Pagliuca, and Alessandra Tappini: Hybrid graph visualizations with chordlink: Algorithms, experiments, and applications. *IEEE Transactions on Visualization and Computer Graphics*, 28(2):1288–1300, 2022, 10.1109/TVCG.2020.3016055.

[AS14] Guy J. Abel and Nikola Sander: Quantifying global international migration flows. *Science*, 343(6178):1520–1522, 2014, 10.1126/science.1248676.

[BB05] Michael Baur and Ulrik Brandes: Crossing reduction in circular layouts. In Juraj Hromkovič, Manfred Nagl, and Bernhard Westfechtel (editors): *Graph-Theoretic Concepts in Computer Science*, pages 332–343. Springer, 2005.

[CDG23] Kris Coolsaet, Sven D'hondt, and Jan Goedgebeur: House of Graphs 2.0: A database of interesting graphs and more, 2023. 10.1016/j.dam.2022.10.013. `https://houseofgraphs.org`.

[FSH19] Amy Finnegan, Saumya S. Sao, and Megan J. Huchko: Using a chord diagram to visualize dynamics in contraceptive use: Bringing data into practice. *Global Health: Science and Practice*, 7(4):598–605, 2019, 10.9745/GHSP-D-19-00205.

[HBS+14] D. Heim, J. Budczies, A. Stenzinger, D. Treue, P. Hufnagl, C. Denkert, M. Dietel, and F. Klauschen: Cancer beyond organ and tissue specificity: Next-generation-sequencing gene mutation data reveal complex genetic similarities across major cancers. *International Journal of Cancer*, 135(10):2362–2369, 2014, 10.1002/ijc.28882.

[KM83] P.A. Koparkar and S.P. Mudur: A new class of algorithms for the processing of parametric curves. *Computer-Aided Design*, 15(1):41–45, 1983, 10.1016/S0010-4485(83)80050-5.

[LR80] Jeffrey M. Lane and Richard F. Riesenfeld: A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(1):35–46, 1980, 10.1109/TPAMI.1980.4766968.

[Mar06] Duncan Marsh: *Applied geometry for computer graphics and CAD*. Springer, 2006.

[MKNF87] Sumio Masuda, Toshinobu Kashiwabara, Kazuo Nakajima, and Toshio Fujisawa: On the NP-completeness of a computer network layout problem. In *Proc. IEEE Intl. Symposium on Circuits and Systems*, pages 292–295, 1987.

[NOF20] Lei Ni and Luis Ospina-Forero: Visualising internal migration flows across local authorities in England and Wales. *Environment and Planning A: Economy and Space*, 53, 2020, 10.1177/0308518X20968568.

[SABS14] Nikola Sander, Guy J. Abel, Ramon Bauer, and Johannes Schmidt: Visualising migration flow data with circular plots. Vienna Institute of Demography Working Papers 2/2014, Vienna Institute of Demography (VID), Vienna, 2014. `https://hdl.handle.net/10419/97018`.

[Sed12] Thomas W Sederberg: Computer aided geometric design. Computer Aided Geometric Design Course Notes, 2012. `https://scholarsarchive.byu.edu/facpub/1`.

[SN90] T.W. Sederberg and T. Nishita: Curve intersection using bézier clipping. *Computer-Aided Design*, 22(9):538–549, 1990, 10.1016/0010-4485(90)90039-F.

[SP86] Thomas W Sederberg and Scott R Parry: Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18(1):58–63, 1986, 10.1016/S0010-4485(86)80013-6.

[WDW+24] B. Wyatt, A. P. Davis, T. C. Wiegers, J. Wiegers, S. Abrar, D. Sciaky, F. Barkalow, M. Strong, and C. J. Mattingly: Transforming environmental health datasets from the comparative toxicogenomics database into chord diagrams to visualize molecular mechanisms. *Frontiers in Toxicology*, 6:1437884, 2024, 10.3389/ftox.2024.1437884.

[YSC+25] Lihua Yuan, Changqing Song, Xiaoqiang Chen, Manjun Zhang, and Menghan Yang: Quantitative analysis of trade position shifts of China and the United States in the Indian Ocean rim trade networks using a weighted centrality approach. *Entropy*, 27(3), 2025, 10.3390/e27030262.