Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Master Thesis Computer Science

An ILP for Perfect Smooth Orthogonal Drawings

Henry Förster

October the 21st, 2016

Reviewers

Michael Kaufmann
(Algorithmik)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Klaus-Jörn Lange
(Theoretische Informatik)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Supervisors

Michael Kaufmann
(Algorithmik)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Michael A. Bekos
(Algorithmik)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Förster, Henry:
An ILP for Perfect Smooth Orthogonal Drawings
Master Thesis Computer Science
Eberhard Karls Universität Tübingen
Thesis period: May 2016 - October 2016

Abstract

Perfect smooth orthogonal drawings represent each vertex as a point in the plane and each edge either as a circular arc or a rectilinear straight line segment. The problem of finding such a drawing given an orthogonal shape as an input is discussed in this thesis. By approximating the arcs with sequences of rectilinear edges an orthogonal shape with edge complexity 1 is created. If this shape is drawn planar while also fulfilling constraints on the lengths of arc approximating edges the resulting drawing can be transformed into a perfect smooth orthogonal drawing. An ILP is provided that can solve the problem of drawing such an orthogonal shape with the mentioned edge length constraints. By improving the arc approximation it can be shown that for an exponentially large input parameter a perfect smooth orthogonal drawing is guaranteed to be found if one exists for the given input shape. After evaluating the performance of the presented approach on practically relevant benchmark test sets the applicability of the approach is discussed and open problems based on the results of this thesis are motivated.

Zusammenfassung

Perfekte glatt-orthogonale Zeichnungen stellen jeden Knoten als einen Punkt der Ebene und jede Kante entweder als Kreisbogen oder als rektilineare Strecke dar. Das Problem eine solche Zeichnung für eine gegebene orthogonale Form als Eingabe zu finden wird in dieser Thesis diskutiert. Durch die Approximation von Kreisbögen mit Sequenzen rektilinearer Kanten wird eine orthogonale Form mit Kantenkomplexität 1 erzeugt. Wird diese Form planar gezeichnet wobei sie auch Beschränkungen bezüglich der Längen der kreisbogenapproximierenden Kanten erfüllen muss, kann die resultierende Zeichnung in eine perfekte glatt-orthogonale Zeichnung transformiert werden. Ein ILP, welches das Problem des Zeichnens einer solchen orthogonalen Form mit den erwähnten Kantenlängenbeschränkungen lösen kann, wird zur Verfügung gestellt. Durch das Verbessern der Kreisbogenapproximationen kann gezeigt werden, dass für einen exponentiell großen Eingabeparameter garantiert werden kann, dass eine perfekte glatt-orthogonale Zeichnung gefunden wird, falls eine solche für die gegebene Form existiert. Nachdem die Performanz des vorgestellten Ansatzes auf praxisrelevanten Benchmark-Testsets evaluiert wurde, wird die Anwendbarkeit des Ansatzes diskutiert und offene Probleme basierend auf den Resultaten dieser Thesis werden motiviert.

Acknowledgements

First and foremost I would like to thank my two supervisors, Prof. Michael Kaufmann and Dr. Michael A. Bekos, for their support and advise and all the helpful discussions during my thesis period. Also I am really happy to have learned about algorithms and graph drawing from you and Dr. Till Bruckdorfer.

I also wish to express a big thank you to the entire Algorithms group at the University of Tübingen, including additionally to my supervisors Dr. Patrizio Angelini and Niklas Heinsohn, for the nice atmosphere in which I could write this thesis as well as for proofreading it in incredibly short time.

As my Master studies now come to a close, I also would like to express my gratitude to all the lecturers that taught me something in the past 6 years, to the friends I got to know who made this time so enjoyable and my parents for their enduring support.

Finally, I would like to thank Prof. Michael Kaufmann for the opportunity to join the Algorithms group after my graduation.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

Smog	Smooth orthogonal
LP	Linear Program, Linear Programming
ILP	Integer Linear Program, Integer Linear Programming
TSM	Topology Shape Metrics
ATEL	Average Total Edge Length
ACT	Average Computation Time
AAEL	Average of Average Edge Length
ADT	Average Declination Time
VLSI	Very Large Scale Integration
i.e.	Latin: id est, meaning ”that is (to say)”
e.g.	Latin: exempli gratia, meaning ”for (the sake of an) example”
cf.	Latin: confer, meaning ”compare to” or ”compare with”
w.r.t.	with respect to

Chapter 1

Introduction

Graphs have many useful applications as tools in different disciplines as well as in our everyday lives:

- Metro plans show the stations of the network (i.e. vertices) which are connected with each other on the plan if there is a train available between the 2 stations.
- Business workflow models connect different states or steps with each other to indicate that one state/step follows the other.
- In the VLSI (Very large scale integration) design process, transistors can be abstracted as vertices of a graph with wires (i.e. edges) connecting them.
- The Unified Modeling Language (UML), primarily used in software development, defines several diagram types that are based on graphs.

All those example applications have in common that they involve the drawing of graphs. Not surprisingly, graph drawing is well studied for aesthetically appealing and well readable drawing types.

One of the most commonly used drawing types is orthogonal drawing. Here vertices are connected by sequences of axis-aligned straight line segments with 90° bends between two line segments. Orthogonal drawings are well known for their angular resolution and readability. However, recent research in human perception indicates, that the sharp bends contained in orthogonal drawings can impede readability [HEH09].

Inspired by the work of American artist Mark Lombardi[1], Bekos et al. introduced the concept of smooth orthogonal drawings [BKKS13]. Smooth

[1]A selection of Lombardi's *Narrative Structures* which show connections between actors in political events and other well-known politicians in artistically drawn graphs with circular arcs can be accessed at `http://www.pierogi2000.com/artists/mark-lombardi/`.

orthogonal drawings retain the angular resolution of orthogonal drawings but in order to change the routing along an edge circular arcs are drawn instead of sharp bends.

Previous research has shown that planar graphs with maximum degree four or less can always be drawn with smooth orthogonal drawings when each edge may consist of at least two segments (straight-line or arc) [ABK$^+$14]. However if every edge shall be drawn either as a single circular arc or a single straight-line segment, i.e. if a perfect smooth orthogonal drawing is requested, only for limited classes of graphs such a result exists [BGPR14].

The goal of this thesis is to find approaches for drawing a graph as a perfect smooth orthogonal drawing given an orthogonal input shape. By only focusing on the "smoothing" process it may be easier to find a solution compared to the problem of finding such a drawing when the input is only a graph without an embedding or shape. Further the identified algorithms should be evaluated using commonly used benchmark test sets with respect to runtime and measurable quality of the resulting drawings. Finally the results should be discussed to show open problems which can be studied subsequently.

The following chapters are organized as follows:

First, in Chapter 2 we will discuss fundamental terminology needed for understanding the remainder of this thesis. Also we will briefly introduce results of previous studies.

In Chapter 3 we will present the main idea of our approach and introduce an ILP with feasible solutions corresponding to perfect smooth orthogonal drawings.

Then, in Chapter 4, we will suggest two improved parametrized versions of our approach and show that both of them are guaranteed to find a perfect smooth orthogonal drawing given exponential parameter settings.

Next, we will evaluate the performance of our approaches on benchmark data in Chapter 5 both with respect to a measurable quality of the produced drawings and the time needed for computation.

Finally we discuss our results and motivate open problems based on our results on perfect smooth orthogonal drawings in Chapter 6 before we concluce with a summary of the thesis in Chapter 7.

Chapter 2

Preliminaries

2.1 Graph Theory

In this section we will define some fundamental terms for the following chapters. Most of the terms are presented similar to their presentation in the work of Nishizeki and Rahman [NR04a] and emphasizing the common way they are used in graph theoretic contexts.

2.1.1 Fundamental Terminology

If we want to discuss graphs, of course first we need to define what a graph is:

Definition 1 (Graph)**.** A *graph* G is a 2-tuple (V, E) consisting of

- a finite set of *vertices* or *nodes* V
- a finite set of *edges* $E \subseteq V \times V$

Therefore, a graph at first is only an abstract mathematical concept. Most commonly it is used to describe relations between some entities. Entities then are represented by a vertex $v \in V$. The existence of a relation between two vertices $v_1, v_2 \in V$ is expressed by the existence of the edge $(v_1, v_2) \in E$. More formally, we say:

Definition 2 (Relations between Vertices and Edges)**.** Two vertices $v_1, v_2 \in V$ are called *neighbors* or to be *adjacent* if $(v_1, v_2) \in E$. For the edge $e = (v_1, v_2) \in E$ we say that it is *incident* to both v_1 and v_2. Also we call v_1 and v_2 the *endpoints* of E. Similarly to vertices, we shall say that two edges $e_1 = (v_1, v_2) \in E$ and $e_2 = (w_1, w_2) \in E$ are neighbors if they have a common endpoint.

Very often, algorithmic applications that deal with graphs require a graph to have special properties. One of the most commonly required properties is that the graph shall be *simple*:

Definition 3 (Simple Graphs and Multigraphs)**.** An edge $(v_1, v_2) \in E$ is called a *loop* if $v_1 = v_2$. Two edges $(v_1, v_2) \in E$ and $(w_1, w_2) \in E$ are called *multiple edges* if they share the same endpoints. A graph is called a *simple graph* if it does not contain any loops or multiple edges. Otherwise, it is called a *multigraph.*

The approaches in this thesis also will require a graph to be simple. Another often required property - especially in the context of orthogonal drawings - is the *maximum degree* of a graph:

Definition 4 (Degree)**.** We say that a vertex $v \in V$ has *degree* k or for short $d(v) = k$ if k edges are incident to v. We say that a graph has *maximum degree* k iff there exists a vertex $v \in V$ with $d(v) = k$ and there exists no vertex v' with $d(v') > k$.

As we will motivate in Section 2.2, for our approaches we will require the graphs to have maximum degree four or less. Further, we will require graphs to be (at least 1-)connected.

Definition 5 (Paths and Connectivity)**.** A sequence of vertices $v_1, v_2, ..., v_k$ is called a *path* in G if

- $v_1, v_2, ..., v_k \in V$
- $\forall 1 \leq i \leq k : \forall 1 \leq j \leq k : i \neq j \implies v_i \neq v_j$
- $\forall 1 \leq i < k : (v_i, v_{i+1}) \in E$

A graph is called *k-connected* if for every pair of vertices $v_1, v_2 \in V$ with $v_1 \neq v_2$ there exist k vertex-disjoint paths between v_1 and v_2. If a graph is 1-connected we also call it *connected.* A graph that is not 1-connected is called *disconnected.*

If a graph is exactly 1-connected, we may observe special types of edges, called *bridges.*

Definition 6 (Bridge (As defined for example by Tarjan in [Tar74] but here limited to connected graphs).)**.** An edge $e \in E$ of a connected graph $G = (V, E)$ is called a *bridge* if and only if removing e would disconnect G.

2.1.2 Graph Drawing and Planar Graphs

So far, we only discussed graphs as an abstract concept. However, many applications also demand to draw a graph. *Drawing* here means that the abstract sets of vertices and edges are represented in a way that is visually perceivable by humans. Very often, vertices are drawn as points, circles or squares whereas edges are drawn as lines connecting the vertices.

It is easy to see, that there are infinitely many drawings of a single graph. One of the most important ways to draw a graph is the *planar representation.* The following definitions for *planar representations*, *planar graphs* and *planar embeddings* all are presented in a fashion similar to the way they are introduced by Weiskircher in [Wei01].

Definition 7 (Planar Representation)**.** A *planar representation* or *planar drawing* of a graph $G = (V, E)$ is a drawing $\Gamma(G)$ where

- every vertex $v \in V$ is drawn as a point in the plane, i.e. $\Gamma(v) = (x, y) \in \mathbb{R}^2$.
- no two vertices $v_1, v_2 \in V$ are drawn at the same point, i.e. $\Gamma(v_1) \neq \Gamma(v_2)$ if $v_1 \neq v_2$.
- every edge $e \in E$ is drawn as an open Jordan curve $\Gamma(e)$ connecting the drawings of its two endpoints and fulfilling the following properties:
  - $\forall e' \in E : e \neq e'$: $\Gamma(e)$ and $\Gamma(e')$ have no common points except if they have a common endpoint. In that case, the only common point(s) is/are their common endpoint(s).
  - $\forall v \in V$: if v is not endpoint of $e \in E$, $\Gamma(e)$ does not contain $\Gamma(v)$

The regions of the plane separated by the representations of edges are called *faces* of Γ. Exactly one of the faces is unbounded, this face is called the *outerface* of Γ.

It turns out that not every graph can be drawn as a planar representation. Therefore, we can define the class of *planar graphs.*

Definition 8 (Planar Graph)**.** A *planar graph* is a graph that can be drawn with a planar representation.

For every planar graph, there exist infinitely many planar representations. However, it can be shown that there exist only finitely many *planar embeddings* as equivalence classes of planar representations.

Definition 9 (Planar Embedding)**.** A *planar embedding* defines the cycles of edges bounding the faces of a drawing as well as which face is the outerface. In contrast to a planar representation it does neither define the position of vertices nor open Jordan curves for edges. A planar representation *realizes* a planar embedding if it uses the embedding's cycles to bound its faces and its outerface is the face designated as the outerface by the embedding.

If a drawing is not planar it contains *crossings*.

Definition 10 (Crossing)**.** Let $G = (V, E)$ be a graph and $\Gamma(G)$ a drawing of G where every vertex $v \in V$ is drawn as a point in the plane. If a pair of edges $e, e' \in E$ is drawn such that $\Gamma(e)$ and $\Gamma(e')$ have a common point c which is not a common endpoint, we call c a crossing.

2.2 Orthogonal Drawings

The definition of planar drawings still allows much freedom in the drawing of edges. While not allowing crossings in a drawing already is beneficial for readability, further improvements are possible by constraining the way we allow an edge to be drawn. *Straight-line drawings* (i.e. drawings where every edge is drawn as a straight line segment) tend to have poor angular resolution[1] and therefore often do not appeal aesthetically to humans. *Orthogonal drawings* (as they are also discussed by EIGLSPERGER ET AL. in [EFK01]) expand the idea of straight-line drawings to using a sequence of straight-line segments to draw an edge. However, in the orthogonal model the orientations of those straight-line segments are further restricted.

[1]Angular resolution is defined by the smallest angle between any two edges incident to a common endpoint. It is one of several commonly used aesthetics measurements. For a list of aesthetics measurements, refer to [NR04b].

Definition 11 (Orthogonal Grid Drawing)**.** An *orthogonal grid drawing* $\Gamma_{Orth}(G)$ of a planar graph $G = (V, E)$ is a planar drawing with the following properties:

- Every vertex $v \in V$ is drawn as an integer grid point in the plane, i.e. $\Gamma_{Orth}(v) = (x, y) \in \mathbb{Z}^2$.
- Every edge $e = (s, t) \in E$ is drawn as a series of axis-aligned straight line segments, i.e. $\Gamma_{Orth}(e) = s_1 s_2 ... s_i$ and $\forall k \in \{1, ..., i\} : s_k$ is a straight line segment with integer coordinate endpoints. Further, $\Gamma_{Orth}(e)$ fulfills the following properties:
  - $\forall k \in \{2, ..., i-1\} : s_k$ only intersects with s_{k-1} and s_{k+1} which are perpendicular to s_k. For any pair of intersecting segments, the intersection happens in a single common endpoint of the segments. Common endpoints are called *bends*.
  - s_1 and s_i each intersect at one of their endpoints with their successor or predecessor in the sequence of segments respectively. The other endpoints are s and t respectively.
- At every vertex $v \in V$, only one of its incident edges may be connected to v from the left/right/top/bottom direction. We say that v has a *west/east/north/south port* at which only one edge can be connected to v.

Note that only graphs of maximum degree 4 or less may allow for an orthogonal drawing because each vertex only has 4 ports.

Since this definition does not make any statement about the number of line segments used for any edge, there can be orthogonal drawings with arbitrary edge routings. Therefore, we define another quality parameter, which we call *orthogonal complexity*.

Definition 12 (Orthogonal Complexity k (as defined in [ABK⁺14]))**.** A planar graph $G = (V, E)$ has *orthogonal complexity* k iff it admits for an orthogonal (grid) drawing $\Gamma^k_{Orth}(G)$ where every edge $e \in E$ is drawn with at most k segments, i.e. $|\Gamma^k_{Orth}(e)| \leq k$. For short, we write $G \in OC_k$. Also, we say that $\Gamma^k_{Orth}(G)$ has orthogonal complexity k and call it an OC_k drawing of G.

Note that OC_1 drawings are very restricted straight-line drawings.

Similar to planar embeddings as equivalence classes, for orthogonal drawings we can define equivalent drawings that use the same *orthogonal shape*.

Definition 13 (Orthogonal Shape)**.** An *orthogonal shape* defines

- a planar embedding.
- the angles (which are multiples of 90°) between edges incident to the same vertex.
- the sequence of bends along each edge (including the information whether it is a left bend or a right bend).

Note, that the orthogonal shape does not tell us anything about vertex positions or edge lengths.

Orthogonal drawings are often drawn with the *Topology-Shape-Metrics Approach* (or for short *TSM approach*) as described by Battista et al. [BETT98]:

1. First, a *topology* of the graph (i.e. a planar embedding) is computed. There exist efficient algorithms to compute an embedding. For orthogonal drawings, any planar embedding is good enough (as shown by Tamassia [Tam87]).
2. Next, we compute an orthogonal *shape*. With the min-flow algorithm presented by Tamassia [Tam87] we can compute the shape with the minimum number of bends.
3. Finally, we fix the edge lengths of the computed shape and therefore when fixing a single vertex position also the positions of all other vertices in the *metrics* step. Minimal total edge lengths here can be computed with a flow network as well (for details refer to Eiglsperger et al. [EFK01]).

Finally, it is worth mentioning, that orthogonal drawings have practical applications in both VLSI design (since it is beneficial to route wires rectilinear) and information visualization (as orthogonal drawings generally appeal aesthetically to humans).

2.3 Introduction to Mathematical Optimization

The following introduction to mathematical optimization is based on the work of Vanderbei [Van10].

Mathematical optimization in general is about finding an optimal solution from a set of feasible solutions. There are several different variants of optimization problems that differ in aspects as

- how it is decided whether a solution is *feasible*.
- how it is decided whether a solution is *optimal*.

In this section we will introduce the variants of optimization that are used in this thesis.

2.3.1 Linear Programming (LP)

The linear programming problem is defined as follows:

Definition 14 (Linear Programming Problem)**.** Given a linear *objective function* $z = \sum_{i=1}^{n} c_i x_i$ with constant c_i's and a set of *constraints* where each is of the linear form $\sum_{i=1}^{n} a_i x_i \{\leq, =, \geq\} b$ with constant a_i's and b over the *decision variables* $x_1, x_2, ..., x_n \in \mathbb{R}$: Assign values to $x_1, x_2, ..., x_n$ such that every constraint is satisfied and the value of z is minimal (or maximal depending on the instance).

An instance of the Linear Programming Problem is called a *linear program* (or for short LP). An assignment of values to the decision variables is called a *solution*. If a solution satisfies every constraint it is called *feasible*. If it further minimizes (or maximizes) the objective function we call it *optimal*.

A huge number of mathematical problems can be modeled as linear programs, e.g. game theoretic problems (finding an optimal strategy), regression, network flow problems. Additionally there exist many applications in business management, e.g. resource allocation, portfolio selection, option prizing.[2]

The most famous and often applied algorithm is the *Simplex Method*. The Simplex Method however has an exponential worst-case running time. *Interior Points Methods* on the other hand achieve polynomial running times. Interior Point Methods can compete with the Simplex Method w.r.t. to runtime for smaller problems[3] and outperform it for larger problems.

2.3.2 Integer Linear Programming (ILP)

While Linear Programming already is a powerful tool, we may easily get into a situation where we want the decision variables to be only integers as not

[2]For more details on those applications, refer to [Van10].

[3]The first polynomial time algorithm, the *Ellipsoid Method*, performed worse than the Simplex method in most practical applications.

always in applications it is possible to split the entities represented by the variables in non-integer parts (e.g. when variables correspond to coordinates on a grid). When we allow only integer values for the variables, we call the problem of finding an optimal solution the *Integer Linear Programming Problem* and instances of this problem are called *Integer Linear Programs* (or for short ILP).

Integer Linear Programming is in fact harder than Linear Programming, it is $\mathcal{NP}$-hard since problems known to be $\mathcal{NP}$-complete, e.g. the travelling salesman problem or even 3-SAT, can be reduced to an ILP[4].

Given an ILP, we may now assume that it would be a good idea to just compute the solution for the LP we obtain when we keep all constraints the same and just allow real numbers instead of integers for the values of decision variables (the resulting LP is called the *LP-Relaxation*) and then round the value of all variables. But this does not always yield the optimal solution even when we have only 2 decision variables.

An algorithm to solve ILP is *Branch & Bound* (cf. Algorithm 1): In Branch & Bound, we solve LP relaxations of the ILP and then check, if their solutions happen to be integer. If this is the case, we check if it is the best solution found so far (*bound*). Otherwise, we *branch* by creating two new ILPs with one additional constraint each which are violated by the non-integer solution we computed. For the initial call, z^* and X^* are undefined. z^* as the bound and X^* as the optimal solution should be available with the same value to all iterations currently running. Note that Branch & Bound can be easily parallelized with every recursive function call using a separate thread. Making z^* available to all running iterations is crucial for the algorithm to terminate as early as possible.

2.3.3 Indicator Constraints and the *bigM* Approach

Integer decision variables turn out to increase the expression power of linear programming more than we might expect at first. With a little trick, they can be used to toggle the need to satisfy a constraint

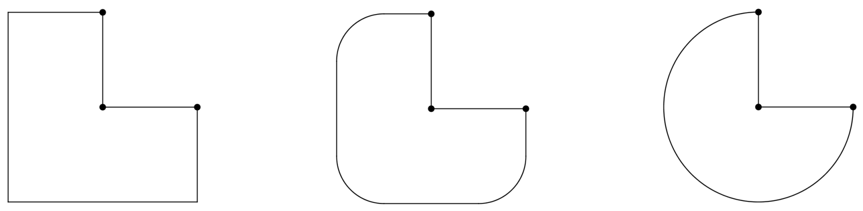
$$a_1x_1 + ... + a_nx_n \geq b$$

on or off as for example discussed in [BLTW15] (along with other approaches on constraints, that can be toggled on or off, or in other words *indicator constraints*): For this to happen we will need to include a new boolean *indicator*

[4]A reduction for the Travelling Salesman Problem can be found in [Van10], 3-SAT is very easy to see: Create a decision variable for every boolean variable in the SAT instance with feasible values drawn from $\{0, 1\}$ and a constraint for every clause of the SAT instance where disjunction is replaced by addition.

Algorithm 1: Branch & Bound Algorithm

```
Input: An ILP $P$, a bound for the objective function $z^*$ (for initial call
       $\pm\infty$ depending on whether the ILP is a minimization or a
       maximization problem), a integer solution $X^*$ (for initial call
       empty) for which the objective function's value is $z^*$
Output: An optimal solution for $P$ if one exists
$P' \leftarrow$ LP-Relaxation of $P$
Solve $P'$
$X \leftarrow$ optimal solution for $P'$
$z \leftarrow$ best objective function value for $P'$
if $P'$ infeasible or $z$ worse than $z^*$ then
    return $X^*$
else
    if $X$ only contains integer values for variables then
        $z^* = z$
        $X^* = X$
        return $X$
    else
        $x_j \leftarrow$ one of the variables violating the integer constraint in $X$
        $v_j \leftarrow$ the value of $x_j$ in $X$
        $X_1 \leftarrow$ Branch & Bound($P$ with $x_j \leq \lfloor v_j \rfloor$ as an additional
        constraint, $X^*$, $z^*$)
        $X_2 \leftarrow$ Branch & Bound($P$ with $x_j \geq \lceil v_j \rceil$ as an additional
        constraint, $X^*$, $z^*$)
        return $OPT(X_1, X_2)$ where $OPT$ is either min or max
```



(a) An orthogonal drawing. **(b)** A smooth orthogonal drawing. **(c)** A smooth orthogonal drawing.

Figure 2.1: Three drawings of the same graph with similar shapes.

variable $i \in \{0,1\}$. Then we can reformulate the constraint to

$$a_1x_1 + ... + a_nx_n + M \cdot (1-i) \geq b$$

where M is a very large positive constant scalar. The idea is that we add $M(1-i)$ on the side that shall be larger. Now if $i = 1$ this equals to adding 0, i.e. the original constraint will need to be satisfied. However, if $i = 0$ then we added M which is very large to the side that should be greater, i.e. we will always satisfy the constraint. This technique is called the *bigM Approach*.

Since - as mentioned before - 3-SAT can be reduced to ILP we are also able to impose logical relations between different indicator variables such that not all indicator variables are set to 1 at the same time.

2.4 Smooth Orthogonal Drawings

Before we conclude this chapter, we still have to define *smooth orthogonal drawings*. Smooth orthogonal drawings are a quite recent research topic and were introduced by Bekos et al. [BKKS13]. Intuitively, we want to retain some of the properties of orthogonal drawings, i.e. the limited number of orientations of straight-line edge segments and the angular resolution defined by the port concept. However, we want to remove the sharp 90° bends along edges by drawing circular arcs instead, or in other words, we want to "smooth" the edges. Sometimes we may even replace more than one bend with a single arc (cf. Figure 2.1).

More formally, we can define:

Definition 15 (Smog Grid Drawing). A *smooth orthogonal* or *Smog grid drawing* $\Gamma_{Smog}(G)$ of a planar graph $G = (V, E)$ is a planar drawing with the following properties:

- Every vertex $v \in V$ is drawn as an integer grid point in the plane, i.e. $\Gamma_{Smog}(v) = (x, y) \in \mathbb{Z}^2$.

- Every edge $e = (s, t) \in E$ is drawn as a series of straight line segments, quarter circle arcs, half circle arcs or three-quarters circle arcs, i.e. $\Gamma_{Smog}(e) = s_1 s_2 ... s_i$ and $\forall k \in \{1, ..., i\} : s_k$ is a straight line segment, a quarter circle, a half circle or a three-quarters circle. Endpoints of segments are always located on integer coordinates. Further, $\Gamma_{Smog}(e)$ fulfils the following properties:
  - $\forall k \in \{2, ..., i-1\} : s_k$ only intersects with s_{k-1} and s_{k+1}. For any pair of intersecting segments, the intersection happens in a single common endpoint of the segments. At this common endpoint the tangents of both segments overlap.
  - s_1 and s_i each intersect at one of their endpoints with their successor and predecessor in the sequence of segments respectively. The other endpoints are s and t respectively. At s and t their tangent's slope is either 0 or $\pm\infty$, i.e. the tangent is horizontal or vertical.
- At every vertex $v \in V$, only one of its incident edges may be connected to v from the left/right/top/bottom direction. We say that v has a *west/east/north/south port* at which only one edge can be connected to v.
- In the scope of this thesis we will also require each circular arc to have an integer radius. This may diverge from definitions of half circle arcs in other publications.

Note that we still require the ports of orthogonal drawings despite two arcs that have a vertex as a common endpoint would not necessarily overlap even if they used the same port.

As we did with orthogonal drawings, we can evaluate the quality of a smooth orthogonal drawing by considering the *smooth complexity.*

Definition 16 (Smooth Complexity k (as defined in [ABK⁺14]))**.** A planar graph $G = (V, E)$ has *smooth complexity* k if it admits for a Smog drawing $\Gamma^k_{Smog}(G)$ where every edge $e \in E$ is drawn with at most k segments, i.e. $|\Gamma^k_{Smog}(e)| \leq k$. For short, we write $G \in SC_k$. Also, we say that $\Gamma^k_{Smog}(G)$ has smooth complexity k and call it a SC_k drawing of G.

In particular, we will deal with SC_1 drawings which are also called *perfect smooth orthogonal drawings.*

Definition 17 (Perfect Smooth Orthogonal Drawing (as defined in [BGPR14])**.** A smooth orthogonal drawing is called *perfect* iff it has smooth complexity 1. We say that an edge has *perfect edge complexity* iff it is drawn with a single segment.

2.5 Related Work

Finally, we will end this chapter with recent results on smooth orthogonal drawings:

In [BKKS13], Bekos et al. showed that if a graph admits for a bend-minimal OC_k drawing that it then also admits for a SC_k drawing with a total number of edge segments not exceeding the total number of edge segments in the bend-minimal OC_k drawing. Also, they could show that SC_1 drawings may require exponential area even when the orthogonal shape of the drawing is not fixed and that there are infinitely many graphs with maximum degree 4 or less that cannot be drawn as a SC_1 drawing.

In [BGPR14] Bekos et al. showed that planar graphs with maximum degree 3 or less can be drawn as SC_1 drawings.

In [ABK+14] Alam et al. showed that every planar graph with maximum degree 4 or less can be drawn as a SC_2 drawing[5]. If the graph even is biconnected and has maximum degree 3 or less, they also upperbounded the required area by $\mathcal{O}(|V|^3)$. Further, they could show that biconnected planar graphs with maximum degree 4 or less can be drawn with SC_1 if they are outerplanar.

[5]Häussner [Hä14] presented an implementation of the constructive proof.

Chapter 3

An ILP for SC_1 Drawings

3.1 General Outline

In this section we want to introduce an approach for SC_1 drawings similar to the TSM framework as described by Battista et al. [BETT98] for orthogonal drawings (cf. Algorithm 2).

```
Algorithm 2: General outline for the Smog TSM approach
  Input: A graph $G = (V, E)$
  Output: A smog drawing $\Gamma_{Smog}(G)$ with $SC_1$ if found
  Compute a planar embedding $\mathcal{E}(G)$
  Compute an orthogonal shape $\mathcal{S}(G)$ which uses $\mathcal{E}(G)$ as its embedding
  Compute positions for the vertices $v \in V$ to obtain a drawing $\Gamma_{Smog}(G)$
  return $\Gamma_{Smog}(G)$
```

As in the TSM approach for orthogonal drawings, we will at first compute a planar embedding $\mathcal{E}(G)$ (the topology) for a given graph $G = (V, E)$, then compute an orthogonal shape $\mathcal{S}(G)$ (i.e. a drawing without fixed edge lengths and vertex positions that implements $\mathcal{E}(G)$) and finally we will assign positions $\Gamma_{Smog}(v)$ on the plane to each vertex $v \in V$ (the metrics) to obtain a smog drawing.

By now, we will use any planar embedding $\mathcal{E}(G)$ and we will compute the orthogonal shape $\mathcal{S}(G)$ with the min-flow algorithm of Tamassia.

Note that not every orthogonal shape $\mathcal{S}(G)$ admits for a Smog drawing $\Gamma_{Smog}(G)$ of SC_1 even if it is bend minimal. To see this, consider the bend minimal shape given in Figure 3.1a[1]. It contains 4 edges with a single bend which therefore should later be drawn as a quarter circle. Intuitively we could

[1]The shape was computed with Tamassia's min-flow algorithm [Tam87] for finding a bend-minimal shape.

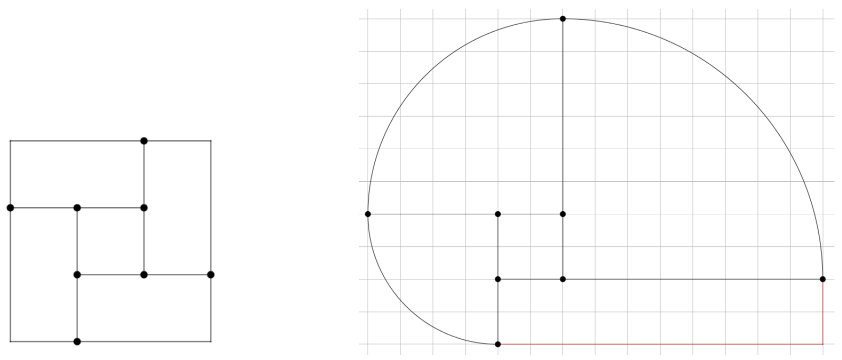

(a) A shape. **(b)** Why it cannot be drawn as SC_1.

Figure 3.1: A bend minimal orthogonal shape that does not admit for a SC_1 drawing.

start for example at the bottom left face and increase its size until we can draw the quarter circle arc there. However, when we then proceed with the other faces with bends in clockwise order, we will observe that every consecutive face has to be drawn larger than the one before. When we finally reach the face with the edge colored red in Figure 3.1b, we can observe that drawing this edge correctly would destroy the quarter circle arc in the bottom left face again.

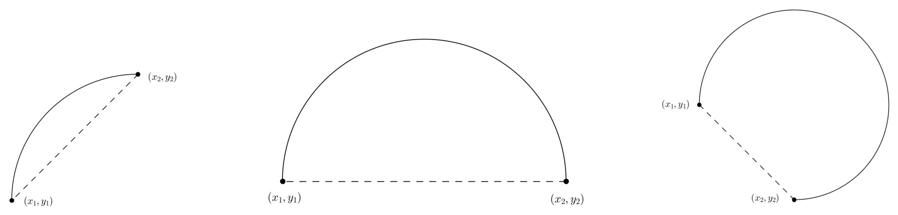
Additionally, assigning edge lengths in the metrics step is not as easy as it is for normal orthogonal drawings. While the goal still is to minimize the sum of the lengths of all edges, the x- and $y-$coordinates of the drawings of vertices $\Gamma(v) = (x, y)$ are not independent from each other in SC_1 drawings. Specifically, circular arcs $\Gamma((v_1, v_2))$ between two vertices v_1 and v_2 impose strict dependencies on the coordinates of $\Gamma(v_1) = (x_1, y_1)$ and $\Gamma(v_2) = (x_2, y_2)$:

- For quarter circle arcs and three-quarters circle arcs: $|x_1 - x_2| = |y_1 - y_2|$, i.e. the two points representing vertices v_1 and v_2 are located on a diagonal.
- For half circle arcs: $|x_1 - x_2| = 0$ or $|y_1 - y_2| = 0$, i.e. the two points representing vertices v_1 and v_2 are located on a straight line. Note that because we also demand integer radii the distance between both points will need to be even (as it is equal to the diameter).

For an illustration, refer to Figure 3.2.

3.2 Metrics

The task in this stage of the TSM approach is the following: Given an orthogonal shape $\mathcal{S}(G)$ of a graph $G = (V, E)$, assign coordinates $\Gamma_{Smog}(v)$ to every



(a) Quarter circle arc. **(b)** Half circle arc. **(c)** Three-quarters circle arc.

Figure 3.2: Possible types of arcs in a SC_1 drawing.

vertex $v \in V$ such that then every edge $e \in E$ can be drawn as a circular arc (quarter circle arc, half circle arc or three-quarters circle arc) or a straight line $\Gamma_{Smog}(e)$. During the calculation we also would like to find a reasonable drawing that uses as few space as possible. Algorithm 3 shows the outline of an ILP approach that assigns edge lengths to all edges in the SC_1 drawing while minimizing the total edge length (which is the usual approximation for minimizing the area if one is restricted to linear operations only).

Algorithm 3: ILP approach for finding a SC_1 drawing based on a given shape $\mathcal{S}(G)$

```
Input: An orthogonal shape $\mathcal{S}(G)$ of a connected planar graph
       $G = (V, E)$
Output: A smog drawing $\Gamma_{Smog}(G)$ with $SC_1$ if exists
Insert dummy vertices and edges to discretely approximate circular arcs
Build an ILP whose constraints preserve planarity and suitable lengths
for dummy edges
Solve the ILP minimizing total edge length
if no solution found then
    return failure
Assign edge lengths to the graph and replace dummy vertices and edges
by circular arcs
return $\Gamma_{Smog}(G)$
```

The basic idea in high-detail is to replace the circular arcs with a sequence of axis-aligned straight line segments that sufficiently approximate the circular arc. Then the constraints of our ILP only need to preserve planarity while also ensuring that there will be enough space to draw the circular arcs. Finally we can solve the ILP and by doing so optimize an useful property, in this case the total edge length.[2] Finally, after we have computed all the edge lengths, we

[2]Another property that can easily be implemented using a linear objective function and some more variables and constraints to store the minimum and the maximum edge lengths is the minimum difference between the shortest and the longest edge length (i.e. the aspect ratio [NR04b]).

can draw the entire graph by fixing the position of a single vertex. Note that if the graph is not connected, this will not work. However, if a graph is not connected, there are two options to still apply the approach:

1. Connect the graph by adding more edges to the orthogonal shape before applying the approach. It is important to only add bridges in order to be sure that the resulting orthogonal shape can be drawn if the original shape could be drawn as well.[3]
2. Apply the approach on each connected component of the graph and putting everything back together afterwards.

Here, no fixed solution for unconnected graphs is presented, since depending on the application one solution may be better than another: Option 1 for example may be beneficial if we want components that are inside the face of another component to still appear inside this component after solving the ILP. On the other hand, option 2 then may yield smaller drawings as the face containing the component in the shape might be drawn with less area.

In the following sections, the steps of the approach will be described more detailed.

3.3 Replacing Circular Arcs - A Simple Approximation

As we will see later, an ILP is sufficient to preserve planarity when assigning edge lengths to a given orthogonal shape. We will exploit this property and replace the bent edges in our given orthogonal shape (that we want to draw as a circular arc later) by simple straight line segments. By doing so, we remove the need for a quadratic constraint for checking intersections of circular arcs with other circular arcs or straight edges. In fact, we will use two sequences of line segments alternating between horizontal and vertical segments, one on either side of the circular arc.

For now, we want to use a *simple approximation*, i.e. we will just use the approximation of circular arcs presented in Figure 3.3. As depicted in the figures, for all types of arcs we will replace the arc by two sectors of squares (depending on the arc type, we will use the same part of the square as the arc uses a part from a circle):

- One of those sequences of straight line segments will be the *outer approximation* (colored blue in Figure 3.3), the sidelength of the corresponding square is as large as the diameter of the circular arc.

[3]By only creating bridges

- The other such sequence will serve as an *inner approximation* (colored red in Figure 3.3), the sidelength of the corresponding square is again defined by the diameter of the circular arc: Intuitively, we can follow the diagonal drawn dashed in Figure 3.3 from the center of the arc until we cross the circular arc. The last point with integer coordinates will be one of the vertices of the square.

- Finally we will connect the two sequences of line segments by another edge (colored green in 3.3) which we will refer to as the *connector edge* (since it connects the inner and the outer approximation). Those edges also contribute to the inner approximation.

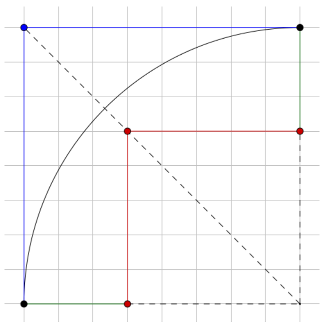
- In a practical implementation, we need to consider the possibility of an existing edge, that is connected to one of the end points of the arc and is routed in the direction of the connector edge (cf. Figure 3.4a). In such a case, the port needed for the connector edge is already occupied, which is bad since we want to create an ILP that preserves planarity in an orthogonal drawing. However, we can solve this issue easily by introducing another dummy edge, which we will later require to have length 0, as depicted in Figure 3.4b (colored orange). Also later it will turn out that we can abuse this *zero length edge* in the ILP for handling it as a special case in our constraints.

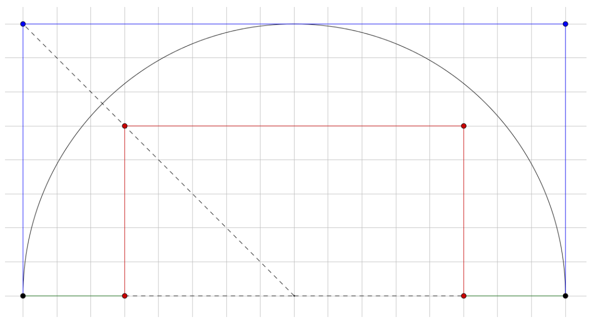
Clearly, after we replaced the bent edges with outer and inner approximation, if the inner approximation stays inside the circular arc in any valid assignment of edge lengths, we will have enough space to draw the arc without intersecting other arcs or straight edges. We will consider an assignment of edge lengths to be viable if the resulting drawing (without arcs but with the dummy edges) is planar with one exception: Faces containing the inner approximation excluding the faces only bounded by dummy edges may only be drawn *weakly planar* at the dummy edges, where a weakly planar drawing is defined as follows:

Definition 18 (Weakly planar drawing)**.** A face f of a straight line drawing $\Gamma(G)$ of a graph $G = (V, E)$ is drawn *weakly planar* at an edge $e \in f$, if

- Either e does not intersect any other edge along the face except those who have a common end point (i.e. f is drawn planar at the edge e)

- or if e intersects another edge $e' \in f$, then either e and e' are parallel (i.e. they overlap) or the intersection point is one of the endpoints of either e or e' (i.e. they touch). Further e' is at the same side of e as f.


(a) Quarter circle arc.


(b) Half circle arc.

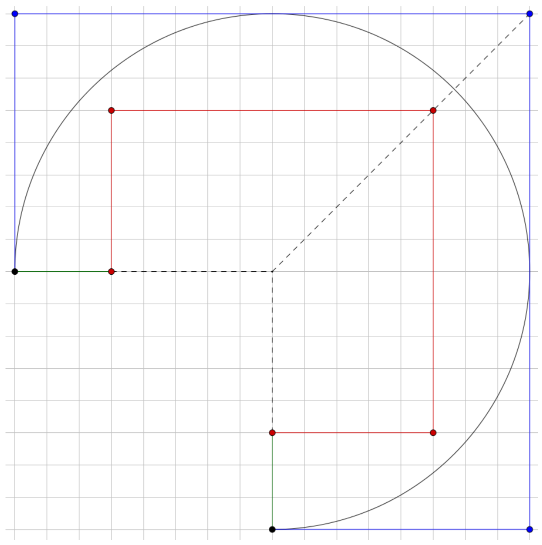

(c) Three quarters circle arc.

Figure 3.3: Simple approximation of the three possible arc types.

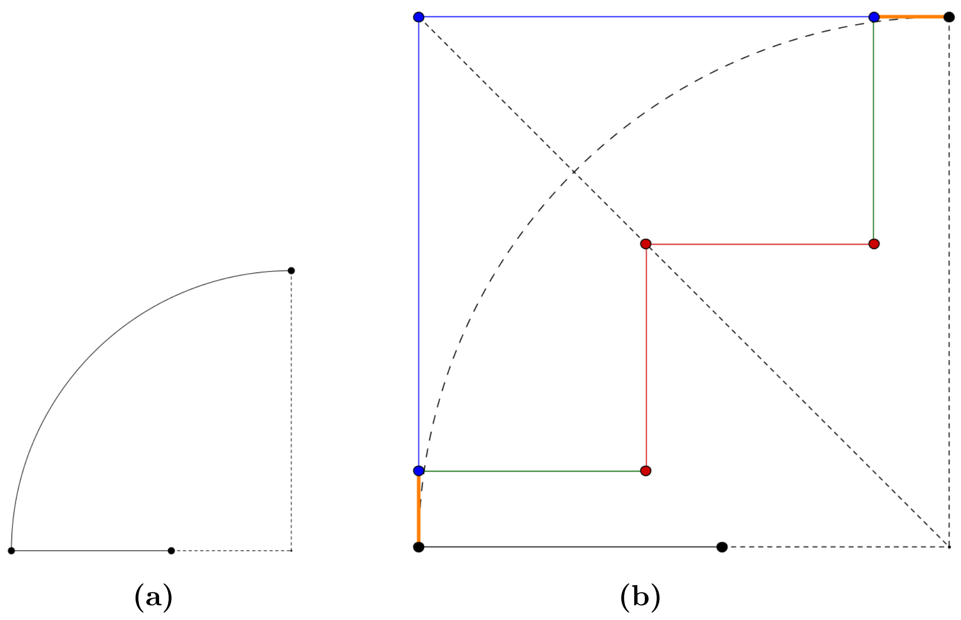


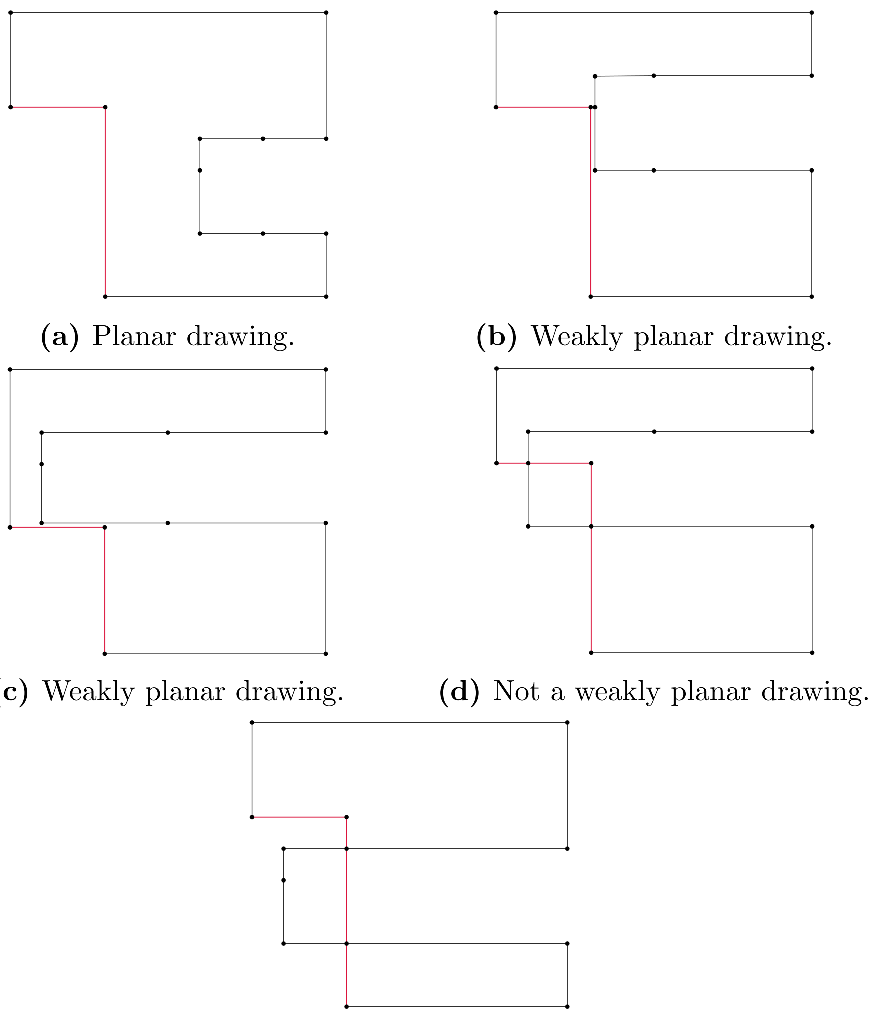
Figure 3.4: (a) An arc with a connected edge that is routed in the direction of the connector edge.
(b) Practical implementation of the arc approximation with dummy edges that will later have length 0 (colored orange).

A *weakly planar drawing* is a drawing, where each face is drawn without intersections or weakly planar.

In order to further emphasize the idea of weakly planar drawings, Figure 3.5 shows an example graph drawn planar, weakly planar and not weakly planar.

The idea behind the weakly planar dummy edges is the following: We know that the inner approximation is located inside the circular arc. Therefore we would waste space (which still is the criterion we would like to optimize) if we would strictly require planarity here. Instead, later in the ILP we will require the inner approximation to at most lay tightly on the edges of the remaining graph inside the circular arc (cf. Figure 3.6). Note that one face can be drawn weakly planar at an edge while the other face may require strict planarity at the same edge, e.g. in the figure you can see that the red edges coincide with the gray area that resembles the rest of the graph while by definition they do not have common points with the blue edges.

To conclude our discussion of the simple approximation, we still need to discuss the edge lengths in relation to the radius of the circular arc that our ILP needs to compute. The edges of the outer approximations are obviously either as long as the radius or the diameter of the circular arc. For the inner approximations however, so far we only said that the vertices on the diagonals from the center of the arc are located on the last integer coordinate before



(a) Planar drawing.

(b) Weakly planar drawing.

(c) Weakly planar drawing.

(d) Not a weakly planar drawing.

(e) Not a weakly planar drawing.

Figure 3.5: Five drawings of the same graph emphasizing the idea of weakly planar drawings. The bounded face may be drawn weakly planar at the red edges. In the 2 weakly planar drawings edges are drawn slightly distinct instead of overlapping for better readability.

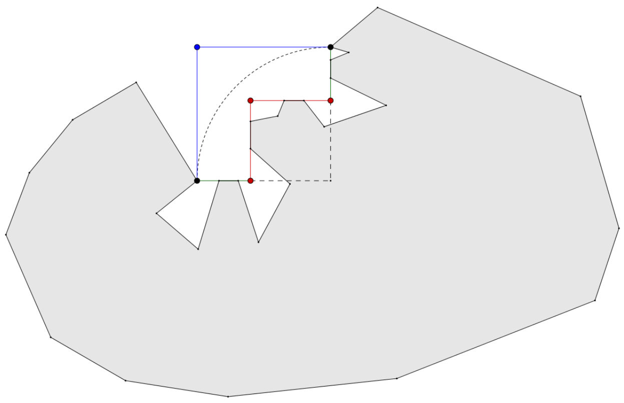


Figure 3.6: A quarter circle arc's inner approximation laying on the rest of the graph (schematic as gray area). Note that the graph is weakly planar at red and green edges (i.e. the inner approximation of the circular arc).

crossing the arc. Let us first formalize the edge lengths of quarter circle arcs:

Lemma 1. *In the simple approximation, quarter circle arcs with radius r are replaced by*

1. *2 edges of length r for the outer approximation.*
2. *2 edges of length $\lfloor \frac{r}{\sqrt{2}} \rfloor$ for the inner approximation.*
3. *2 edges of length $\lceil (1 - \frac{1}{\sqrt{2}})r \rceil$ for the connector edges.*

Proof. As our arc forms a quarter circle and the outer approximation edges span over this arc horizontally or vertically, those outer approximation edges need to have length equal to the radius $l_o = r$. For the inner approximation edges consider the right triangle in Figure 3.7: The blue diagonal crosses the last integer point that we want for our approximation before cutting the circular arc. If we can compute the lengths of the red edges in Figure 3.7, we can round down their lengths and obtain the correct lengths for our inner approximation. Moreover, the two red edges form a square with the blue diagonal of length r as its diagonal. Therefore we can use PYTHAGORAS' theorem to compute the length of the equally sized red edges l_i:

$$l_i^2 + l_i^2 = r^2 \implies l_i = \frac{r}{\sqrt{2}}$$

Finally, as visible in Figure 3.3a, the outer approximation's length is equal to the inner approximation's length plus the connector edge's length l_c:

$$l_o = l_i + l_c \implies l_c = l_o - l_i = r - \frac{r}{\sqrt{2}} = (1 - \frac{1}{\sqrt{2}})r$$

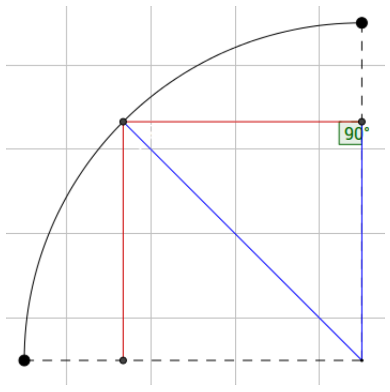


Figure 3.7: The point of the arc cut when moving on the diagonal away from the center and its connection to the inner approximation edges.

Since we will round down the inner approximation's edge length, we will have to round up the connector's edge length. □

Following similar argumentation, we can also prove the following:

Lemma 2. *In the simple approximation, arcs are replaced by approximations with edge lengths dependent on their radius r as depicted in Figure 3.8.*

3.4 Building the ILP

Let $G^* = (V^*, E^*)$ be the graph obtained when replacing all bent edges of the original graph G with the simple approximation. Also, the replacing already defines the orthogonal shape $\mathcal{S}(G^*)$ given an original orthogonal shape $\mathcal{S}(G)$. Note that $\mathcal{S}(G^*)$ is an OC_1 shape, i.e. every edge is either drawn horizontal or vertical and has no bends. The ILP we are about to describe will have an edge length variable for every edge $e \in E^*$ of the graph G^*:

$$l(e) \in \mathbb{N} \tag{3.1}$$

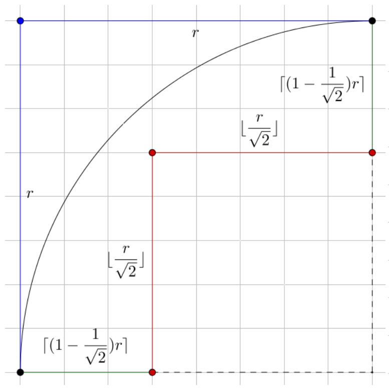
While in general, we will require

$$l(e) \geq 1 \tag{3.2}$$

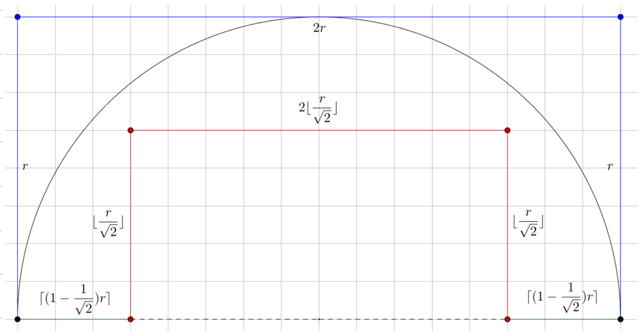
for edges of the inner approximation e_i we will instead demand

$$l(e_i) \geq 0 \tag{3.3}$$

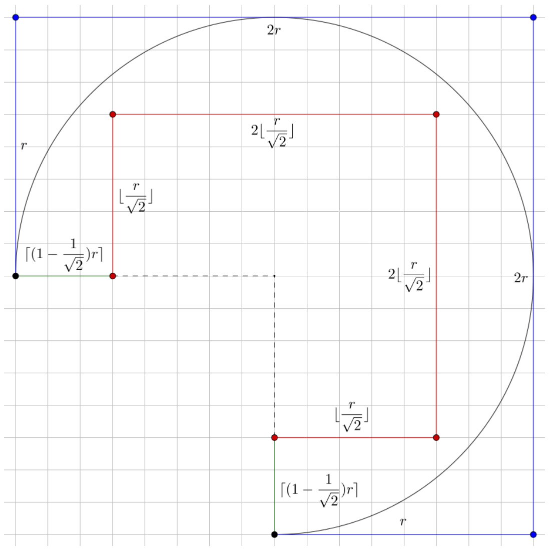
i.e. the inner approximation can collapse to a single point if the arc does not contain any other edges.


r
$\lceil(1-\frac{1}{\sqrt{2}})r\rceil$
$\lfloor\frac{r}{\sqrt{2}}\rfloor$
r
$\lfloor\frac{r}{\sqrt{2}}\rfloor$
$\lceil(1-\frac{1}{\sqrt{2}})r\rceil$

(a) Quarter circle arc.


$2r$
$2\lfloor\frac{r}{\sqrt{2}}\rfloor$
r
r
$\lfloor\frac{r}{\sqrt{2}}\rfloor$
$\lfloor\frac{r}{\sqrt{2}}\rfloor$
$\lceil(1-\frac{1}{\sqrt{2}})r\rceil$
$\lceil(1-\frac{1}{\sqrt{2}})r\rceil$

(b) Half circle arc.


$2r$
$2\lfloor\frac{r}{\sqrt{2}}\rfloor$
r
$\lfloor\frac{r}{\sqrt{2}}\rfloor$
$\lceil(1-\frac{1}{\sqrt{2}})r\rceil$
$2\lfloor\frac{r}{\sqrt{2}}\rfloor$
$2r$
$\lfloor\frac{r}{\sqrt{2}}\rfloor$
$\lceil(1-\frac{1}{\sqrt{2}})r\rceil$
r

(c) Three quarters circle arc.

Figure 3.8: Simple approximation of the three possible arc types with required edge lengths.

Also for each circular arc a which we represent with its simple approximation we store its radius in a variable of the ILP:

$$r(a) \in \mathbb{N}^+ \tag{3.4}$$

which may not be needed for the simple approximation but will prove useful in the more sophisticated approaches in Chapter 4.

Further needed variables will be defined when we need them.[4]

3.4.1 Constraints for Approximation Edge Lengths

Obviously, Lemma 2 has to be encoded in linear constraints of the ILP. This can be done easily since the lengths of all edges depend linearly - aside from floor and ceiling functions - on the radius $r(a)$ of the corresponding arc a. Including floor and ceiling also is trivial since we can replace them by *less or equal* and *greater or equal* in the ILP. Note that by doing so the inner approximation edges may be at most as long as given by the lemma but they may also be shorter (in fact they may even have lenght 0). This is not causing any problem because the computed edge lengths of dummy edges is not part of the solution and we will still check for crossings. Let $e_{f(r)}$ denote an edge with length $f(r)$ as given by Figure 3.8. We will use the following equations to ensure correct relations between the edge lengths:

$$l(e_r) - r(a) = 0 \tag{3.5}$$

$$l(e_{\lceil (1-\frac{1}{\sqrt{2}})r \rceil}) - \left(1 - \frac{1}{\sqrt{2}}\right) r(a) \geq 0 \tag{3.6}$$

$$\frac{1}{\sqrt{2}} r(a) - l(e_{\lfloor \frac{1}{\sqrt{2}} r \rfloor}) \geq 0 \tag{3.7}$$

$$l(e_{2r}) - 2r(a) = 0 \tag{3.8}$$

Let $e^1_{f(r)}$ and $e^2_{f(r)}$ denote the two edges in the approximation of same length $f(r)$. Those two edges will be required to have the same length by the following constraint:

$$l(e^1_{f(r)}) - l(e^2_{f(r)}) = 0 \tag{3.9}$$

The length of the edges $e_{2\lfloor \frac{1}{\sqrt{2}} r \rfloor}$ will be ensured by constraints that demand the closing of faces (cf. Constraints 3.10 and 3.11).

[4]In the practical implementation using Gurobi it proved important to define all the variables first before adding constraints as doing so improved the performance of the ILP solver compared to a previous code version where variables where defined on the fly. Note that the complexity of the approach stays the same since only the loops creating the constraints have to be passed twice - first to create needed variables, then to create the constraints.

3.4.2 Constraints for Planarity

We will add contraints for each face f of G^* that preserve the planarity of f. The idea is, that if we do this for all faces then the entire graph will be planar[5]. First, let us discuss the notation we will use in this section.

Definition 19 (Sets along faces)**.** For every face f of an orthogonal shape $\mathcal{S}(G)$ with fixed orientation with OC_1 of a graph G the following sets can be defined:

- the set of horizontal edges along the face $E_h(f)$.
- the set of vertical edges along the face $E_v(f)$.

Additionally, corresponding to any fixed cyclic order [a] and orientation of the edges along the face (cf. Figure 3.9) we can also define:

- the set of leftward (w.r.t. the given cyclic orientation) edges along the face $E_l(f)$.
- the set of rightward (w.r.t. the given cyclic orientation) edges along the face $E_r(f)$.
- the set of topward (w.r.t. the given cyclic orientation) edges along the face $E_t(f)$.
- the set of bottomward (w.r.t. the given cyclic orientation) edges along the face $E_b(f)$.

[a]The cyclic order is defined by the planar embedding of the graph.

Figure 3.9 also indicates a property, any face f has to fulfil: If we start at any vertex and follow the cyclic order, we will get back to the same vertex after having visited all edges. Therefore, the sum of the lengths of the edges going to the left $(E_l(f))$ is the same as the sum of the edges going to the right $(E_r(f))$. Of course, the same holds for vertical edges. We can formulate this as the first two planarity constraints for our ILP:

$$\sum_{e_r \in E_r(f)} l(e_r) - \sum_{e_l \in E_l(f)} l(e_l) = 0 \tag{3.10}$$

[5]An orthogonal shape contains information about a planar embedding. A planar embedding describes which faces exist in planar drawings realizing the embedding. Therefore if we draw all the faces that are given by the embedding planar the resulting drawing will be planar.

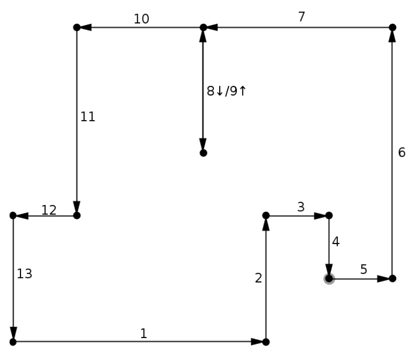


Figure 3.9: A bounded face of an orthogonal shape with OC_1 with a given cyclic order (numbers along the edges) and the resulting orientation of edges. Note that the bridge edge appears twice in the cyclic order with opposite directions.

$$\sum_{e_t \in E_t(f)} l(e_t) - \sum_{e_b \in E_b(f)} l(e_b) = 0 \tag{3.11}$$

Now that we have constraints that make sure our faces are closed, we still need to ensure that the resulting drawing of each face is planar. For the ILP we use the following formulation of planar faces:

Lemma 3. *A face f of an orthogonal drawing $\Gamma_{Orth}(G)$ with OC_1 of a graph G is drawn planar if*

1. *there is no pair of one horizontal $e_h \in E_h(f)$ and one vertical edge $e_v \in E_v(f)$ which do not share a common endpoint but intersect and*
2. *pairs of parallel bridges that do not share a common end point do not overlap.*
3. *pairs of edges that have a common endpoint only intersect/overlap in this single point.*

Proof. We show this by showing that every face that is not drawn planar

1. has an intersection between a horizontal and a vertical edge that do not share a common endpoint or
2. an overlapping pair of bridges.

We do so by distinguishing the different possibilities of crossings:

- A horizontal and a vertical edge cross at a single point. In this case we obviously have Case 1, i.e. a horizontal and a vertical edge intersect.
- Two parallel edges have at least one point in common, i.e. they overlap. In this case, we further need to distinguish:
  - If both edges are not bridges, each of them is connected to at least two other edges via its endpoints, and each endpoint is connected with at least one other edge. Also, either
    - one of the endpoints of either edge is located between the endpoints of the other edge (cf. Figure 3.10a)
    - both of the endpoints of one edge are located between the endpoints of the other edge (cf. Figure 3.10b)

    In either case, we are done if one of the end points located between the two end points of the other edge is connected with an edge of perpendicular orientation (e.g. in Figure 3.10 it should be vertical) as then we also had an intersection of a horizontal and a vertical edge (case 1). If this is not the case, we still know that each edge is connected to another edge at each endpoint with the same orientation (e.g. in Figure 3.10 it would be horizontal), i.e. we would have another pair of parallel overlapping edges. So we could have two sequences of parallel edges that overlap. However, since faces should not be drawn empty, we know that at some point both sequences of edges will be connected to edges of perpendicular orientation, so also in this case we would end up with an intersection of a horizontal and a vertical edge. If we further demand that all edges have non-zero length this perpendicular edge cannot be the connection between the two sequences of parallel edges, i.e. we end up with Case 1.
  - If one edge is a bridge and the other one is not, we can argue similarly to the case where both edges are not bridges. Reason for this is a property of the bridge: Intuitively the bridge will point into the face. But when the face should also be connected, we can only have 2 configurations:
    - the bridge is connected to one parallel and one perpendicular edge (cf. Figure 3.11a)
    - the bridge is connected to two perpendicular edges (cf. Figure 3.11b)

    This means, in either case, even if the bridge is connected to a sequence of parallel bridges, we can follow the bridge to its connection to the rest of the face. At some point either the non-bridge sequence of parallel edges will be connected to a perpendicular edge or the

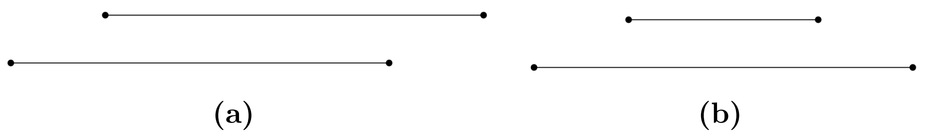


Figure 3.10: Possibilities for overlappings of pairs of parallel edges. Note that the edges are drawn distinct only for better readability.

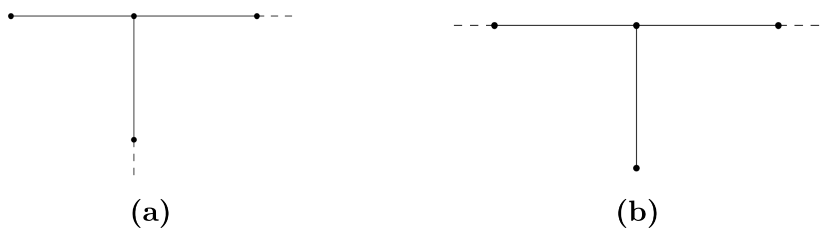


Figure 3.11: Possibilities for connections of bridges to the face. Dashed edges suggest that the border of the face continues from this endpoint.

bridge ends (and then it will be connected to a perpendicular edge). Therefore also in this case we end up with Case 1. Note that this argument does not hold for two bridges. Therefore we need one more case.

– If both edges are bridges by definition we have Case 2.

□

Now that we have an appropriate definition for planarity in OC_1 drawings, we need to encode the two conditions of Lemma 3 in constraints for our ILP. For now we will take care of the first condition: horizontal and vertical edges should not intersect. Again we will establish another description that is more suitable for an ILP:

Lemma 4. *A horizontal edge $e_h \in E_h(f)$ and a vertical edge $e_v \in E_v(f)$ that do not share a common endpoint of a face f of an OC_1 drawing do not intersect if at least one of the following conditions is met:*

1. *The vertical edge is right of the horizontal edge (cf. Figure 3.12a).*
2. *The vertical edge is left of the horizontal edge (cf. Figure 3.12b).*
3. *The horizontal edge is on top of the vertical edge (cf. Figure 3.12c).*

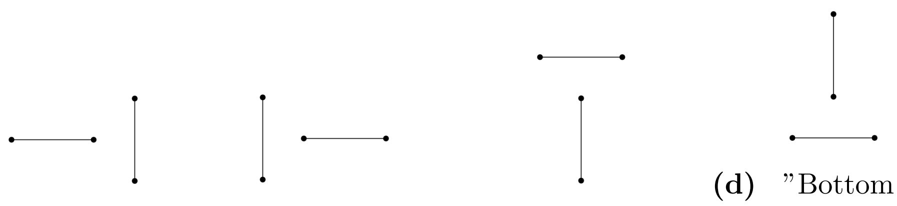

(a) "Right of" case **(b)** "Left of" case **(c)** "Top of" case **(d)** "Bottom of" case

Figure 3.12: The 4 sufficient conditions for non-crossing edges.

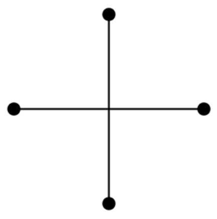


Figure 3.13: If two edges intersect, they necessarily violate all conditions of Lemma 4.

4. The horizontal edge is at the bottom of the vertical edge (cf. Figure 3.12d).

Proof. If there is a crossing between horizontal and vertical edges the vertical edge will be in between the two endpoints of the horizontal edge in horizontal direction. Also, the horizontal edge will be in between the two endpoints of the vertical edge in vertical direction (cf. Figure 3.13). So if there is a crossing, none of the four conditions of the lemma may be true. □

Since Lemma 4 makes statements about required positions of edges relatively to each other, we need to define a *local coordinate system* for each face that only depends on the lengths of the edges. Intuitively we want to place the origin of the coordinate system on a vertex (cf. Figure 3.14). Then, we can use the cyclic order of edges to express the position of each vertex and edge by sums of the lengths of edges. For instance, the coordinates of the yellow vertex in the Figure can be described as $(l(e_1) + l(e_3), l(e_2) - l(e_4))$ where e_i corresponds to the edge with label i in the figure and $l(e)$ is the length of the edge e. More formally, we can define the coordinate system as follows:

Definition 20 (Local coordinate system for a face)**.** Let $e_1, e_2, e_3, ..., e_k$ be the cyclic order of edges along a face f of an OC_1 drawing. We define the source of the local coordinate system to be at the intersection of e_1 and e_k.

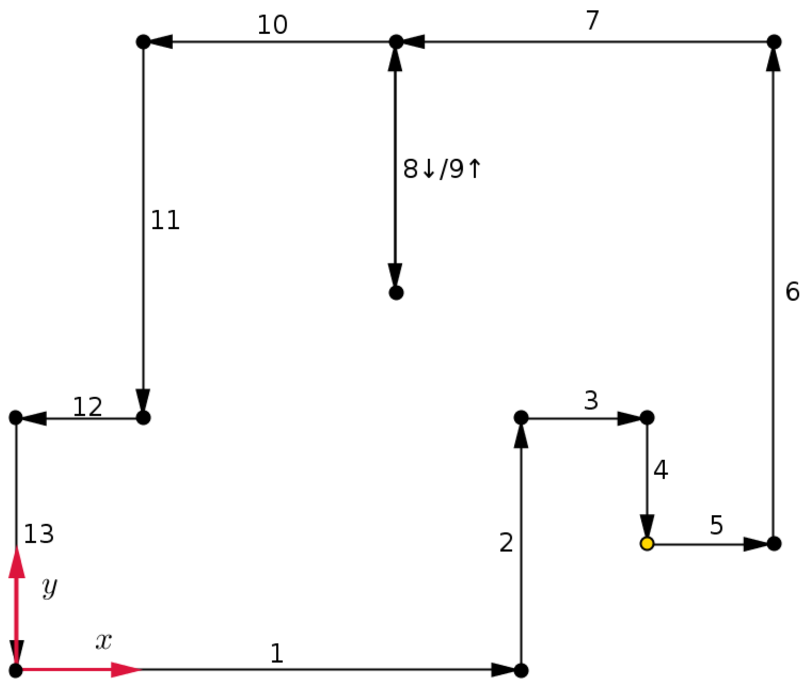


Figure 3.14: Concept of the local coordinate systems for a face. The origin of the coordinate system is placed on a vertex (red). Then the positions of edges and vertices can be expressed depending on the length of the edges along the cyclic order of edges. For instance, the coordinates of the yellow vertex in the Figure can be described as $(l(e_1) + l(e_3), l(e_2) - l(e_4))$ where e_i corresponds to the edge with label i in the figure and $l(e)$ is the length of the edge e.

For any edge e_i with $1 \leq i \leq k$ let $E_i = \{e_1, e_2, ..., e_i\}$ and therefore $E_{i-1} = \{e_1, e_2, ..., e_{i-1}\}$.

Then we define the following for all horizontal edges $e_h = e_i$ with $1 \leq i \leq k$:

- $left(e_h) = \begin{cases} \sum\limits_{e_r \in E_{i-1} \cap E_r} l(e_r) - \sum\limits_{e_l \in E_i \cap E_l} l(e_l) & \text{if } e_h \in E_l(f) \\ \sum\limits_{e_r \in E_{i-1} \cap E_r} l(e_r) - \sum\limits_{e_l \in E_{i-1} \cap E_l} l(e_l) & \text{if } e_h \in E_r(f) \end{cases}$
  which corresponds to the horizontal coordinate of the left endpoint of e_h.

- $right(e_h) = \begin{cases} \sum\limits_{e_r \in E_{i-1} \cap E_r} l(e_r) - \sum\limits_{e_l \in E_{i-1} \cap E_l} l(e_l) & \text{if } e_h \in E_l(f) \\ \sum\limits_{e_r \in E_i \cap E_r} l(e_r) - \sum\limits_{e_l \in E_{i-1} \cap E_l} l(e_l) & \text{if } e_h \in E_r(f) \end{cases}$
  which corresponds to the horizontal coordinate of the right endpoint of e_h.

- $vertical(e_h) = \sum\limits_{e_t \in E_{i-1} \cap E_t} l(e_t) - \sum\limits_{e_b \in E_{i-1} \cap E_b} l(e_b)$
  which corresponds to the vertical coordinate of both endpoints of e_h.

Further we define the following for all vertical edges $e_v = e_i$ with $1 \leq i \leq k$:

- $bottom(e_v) = \begin{cases} \sum\limits_{e_t \in E_{i-1} \cap E_t} l(e_t) - \sum\limits_{e_b \in E_i \cap E_b} l(e_b) & \text{if } e_v \in E_b(f) \\ \sum\limits_{e_t \in E_{i-1} \cap E_t} l(e_t) - \sum\limits_{e_b \in E_{i-1} \cap E_b} l(e_b) & \text{if } e_v \in E_t(f) \end{cases}$
  which corresponds to the vertical coordinate of the bottom endpoint of e_v.

- $top(e_v) = \begin{cases} \sum\limits_{e_t \in E_{i-1} \cap E_t} l(e_t) - \sum\limits_{e_b \in E_{i-1} \cap E_b} l(e_b) & \text{if } e_v \in E_b(f) \\ \sum\limits_{e_t \in E_i \cap E_t} l(e_t) - \sum\limits_{e_b \in E_{i-1} \cap E_b} l(e_b) & \text{if } e_v \in E_t(f) \end{cases}$
  which corresponds to the vertical coordinate of the top endpoint of e_v.

- $horizontal(e_v) = \sum\limits_{e_r \in E_{i-1} \cap E_r} l(e_r) - \sum\limits_{e_l \in E_{i-1} \cap E_l} l(e_l)$
  which corresponds to the horizontal coordinate of both endpoints of e_v.

Note that the defined local positions only depend on sums of lengths of edges - which are the variables of our ILP.

Next, we will formulate Lemma 4 as constraints for the ILP. First of all, we will add new boolean variables for the pairs of edges on a face to keep track of which of the 4 conditions are satisfied:

$$\begin{aligned} leftOf(e_h, e_v, f) &\in \{0, 1\} \\ rightOf(e_h, e_v, f) &\in \{0, 1\} \\ topOf(e_h, e_v, f) &\in \{0, 1\} \\ bottomOf(e_h, e_v, f) &\in \{0, 1\} \end{aligned} \tag{3.12}$$

Semantically, $leftOf(e_h, e_v, f) = 1$ shall mean, that the "left of" case as depicted in Figure 3.12b is satisfied, while $leftOf(e_h, e_v, f) = 0$ shall state, that it might not be the case. The other variables have similar meanings. For short, we shall write $leftOf$ instead of $leftOf(e_h, e_v, f)$ in the following discussion. Also, we will shorten the other variables in a similar fashion. As Lemma 4 states, at least one of the 4 cases needs to be satisfied, so we can add the following constraint for all pairs of horizontal e_h and vertical edges e_v not sharing a common endpoint for each face f:

$$leftOf + rightOf + topOf + bottomOf \geq 1 \tag{3.13}$$

Now we only need to encode the four cases in a linear constraint. The goal here is that if one of the conditions of Lemma 4 is not fulfilled, setting the corresponding boolean variable to 0 will result in a fulfilled equation. On the other hand, to ensure the semantic meaning of the variables, the constraints need to ensure that a variable may only be set to 1 if the corresponding condition of Lemma 4 is fulfilled. We will use the following formulations which use the *bigM* method for turning them into indicator constraints:

$$horizontal(e_v) - right(e_h) - M \cdot rightOf \geq -M + 1 \tag{3.14}$$

$$left(e_h) - horizontal(e_v) - M \cdot leftOf \geq -M + 1 \tag{3.15}$$

$$vertical(e_h) - top(e_v) - M \cdot topOf \geq -M + 1 \tag{3.16}$$

$$bottom(e_v) - vertical(e_h) - M \cdot bottomOf \geq -M + 1 \tag{3.17}$$

where M is a suitably large integer.[6]

[6]Defining how large "suitably large" is, can be tricky. M needs to be larger than any of the differences between coordinates of edges computed. Otherwise we may not find a solution despite it exists (e.g. if $left(e_h) - horizontal(e_v) \ngeq 1$, it is required that $M \geq$

If we look at the constraints above and ignore the summands with M for a while, e.g. for the "left of" case we get the following: $left(e_h) - horizontal(e_v) \geq 1$. This *reduced constraint* tells us simply speaking just that the vertical edge needs to be at least one unit to the left of the left end point of the horizontal edge which is exactly the "left of" case. Now if we look at the entire constraint again, setting $leftOf = 1$ we again end up with the reduced constraint, i.e. if $leftOf = 1$ the "left of" case needs to be satisfied. On the other hand, if we set $leftOf = 0$, we can see that if M is significantly large, the constraint tells us that some difference of coordinates needs to be larger than some very negative number with large absolute value. In fact, if we have chosen a suitable value for M than this negative value will always be smaller than any difference of coordinates, therefore if $leftOf = 0$ the reduced constraint will be ignored and the vertical edge does not need to be left of the horizontal one. The same argumentation holds for the other three constraints.

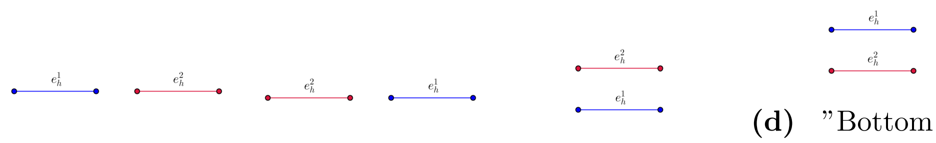
So far we have encoded the first condition of Lemma 3 in the ILP. But we still need to take care of the second condition: no pair of parallel bridges is allowed to overlap. First of all, finding brigdes is easy, they appear twice in the cyclic order of edges along a face (cf. Figure 3.9). Bridge edge pairs which share a vertex of course need to be excluded since they overlap in this point and only in this point. Again we will establish a suitable definition of not overlapping:

Lemma 5. *Two horizontal edges $e_h^1, e_h^2 \in E_h(f)$ do not overlap if at least one of the following conditions is satisfied:*

1. *e_h^2 is right of e_h^1 (cf. Figure 3.15a).*
2. *e_h^2 is left of e_h^1 (cf. Figure 3.15b).*
3. *e_h^2 is on top of e_h^1 (cf. Figure 3.15c).*
4. *e_h^2 is at the bottom of e_h^1 (cf. Figure 3.15d).*

Symmetrically, one may define corresponding conditions for vertical edges $e_v^1, e_v^2 \in E_v(f)$

$horizontal(e_v) - left(e_h) + 1$ because otherwise the "left of" case is not ignored). Although in theory we could set M to arbitrarily high values we cannot do this in a practical context as for performance and memory reasons LP solver software is not using arbitrarily precise data types - if M is too large, the LP solver may round both sides of the constraint to $-M$ and simply accept any solution for the variables. Note, that while modern LP solvers allow for the usage of precision modes one should avoid using those unless values of M that are too large for the normal data types are absolutely required due to the values of other coefficients or variables in the correct solution as more precise data types need more memory and slow down computations.



(a) "Right of" case **(b)** "Left of" case **(c)** "Top of" case **(d)** "Bottom of" case

Figure 3.15: The four sufficient conditions for non-overlapping horizontal edges.

Proof. In order for two horizontal edges to overlap, their endpoints must have equal vertical coordinates. So if either the "top of" or the "bottom of" case (3. or 4.) is satisfied, two edges may not overlap. Further at least one end point of one edge needs to be located in between the two end points of the second edge. This is the case unless the left end point of one edge is to the right of the right end point of the other edge, i.e. if either the "right of" or the "left of" case (1. or 2.) is satisfied, two edges may not overlap. □

Similarly to checking if edges do not cross, we will introduce new variables to the ILP for checking which one of the conditions of Lemma 5 is satisfied. So for a pair of two parallel bridge edges e_b^1, e_b^2 (horizontal or vertical) on a face f we will add the Boolean variables:

$$\begin{aligned} leftOf_b(e_b^1, e_b^2) &\in \{0,1\} \\ rightOf_b(e_b^1, e_b^2) &\in \{0,1\} \\ topOf_b(e_b^1, e_b^2) &\in \{0,1\} \\ bottomOf_b(e_b^1, e_b^2) &\in \{0,1\} \end{aligned} \tag{3.18}$$

Note that this time, the variables are not referenced by the face f since bridge edges are only appear in the cyclic order of edges around one single face. Similarly to the discussion of crossing edges, $leftOf_b(e_b^1, e_b^2) = 1$ shall indicate, that e_b^2 indeed is left of the edge e_b^1. In the further discussion we will use a shortened notation ($leftOf_b, rightOf_b, topOf_b, bottomOf_b$). Again, we know that one of the four cases has to be satisfied, therefore we add the following constraint:

$$leftOf_b + rightOf_b + topOf_b + bottomOf_b \geq 1 \tag{3.19}$$

The remaining constraints depend on whether the bridge edges are horizontal or vertical. First, we will focus on the horizontal case. We will apply the "big M" method as we did before for the constraints that prevent crossings:

$$left(e_b^2) - right(e_b^1) - M \cdot rightOf \geq -M + 1 \tag{3.20}$$

$$left(e_b^1) - right(e_b^2) - M \cdot leftOf \geq -M + 1 \tag{3.21}$$

$$vertical(e_b^2) - vertical(e_b^1) - M \cdot topOf \geq -M + 1 \tag{3.22}$$

$$vertical(e_b^1) - vertical(e_b^2) - M \cdot bottomOf \geq -M + 1 \tag{3.23}$$

Since the "big M" method is applied in a similar fashion as explained before, we will not argue about the correctness here again. For the vertical case we add the following constraints:

$$bottom(e_b^2) - top(e_b^1) - M \cdot topOf \geq -M + 1 \tag{3.24}$$

$$bottom(e_b^1) - top(e_b^2) - M \cdot bottomOf \geq -M + 1 \tag{3.25}$$

$$horizontal(e_b^2) - horizontal(e_b^1) - M \cdot rightOf \geq -M + 1 \tag{3.26}$$

$$horizontal(e_b^1) - horizontal(e_b^2) - M \cdot leftOf \geq -M + 1 \tag{3.27}$$

This concludes the constraints we need for ensuring planarity in a drawing. However, when we augmented the drawing by dummy vertices, we argued that at some edges we would only require weak planarity. Next, we need to discuss how to alter the planarity constraints such that they will ensure weak planarity.

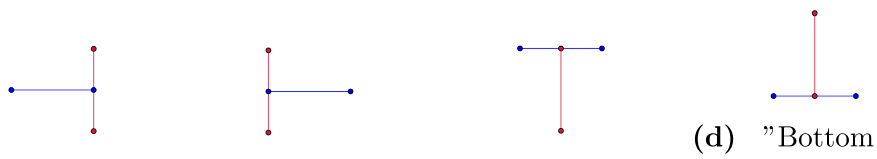
3.4.3 Constraints for Weak Planarity

Let us begin by recalling the concept of weakly planar drawn edges that we introduced earlier. For weakly planar drawn edges e_{wp} we demand that they are either drawn planar or that crossing edges e_{cross} are parallel or the intersection point is one of the end points of either e_{wp} or e_{cross}. It is intuitively clear, that the following lemma holds:

Lemma 6. *A face f of an orthogonal drawing $\Gamma_{Orth}(G)$ with OC_1 of a graph G is drawn weakly planar at an edge $e_{wp} \in f$ if*

1. *if $e_{wp} \in E_h(f)$ there is no vertical edge $e_v \in E_v(f)$ which does not share a common endpoint with e_{wp} but intersects with e_{wp} in a point that is not an endpoint of e_{wp} or e_v*

2. *if $e_{wp} \in E_v(f)$ there is no horizontal edge $e_h \in E_h(f)$ which does not*



(a) "Right of" case **(b)** "Left of" case **(c)** "Top of" case **(d)** "Bottom of" case

Figure 3.16: The four sufficient conditions for weakly planar intersections.

share a common endpoint with e_{wp} but intersects with e_{wp} in a point that is not an endpoint of e_{wp} or e_h

Note that the part with overlapping bridges that can be found in Lemma 3 is omitted here since weakly planar edges by definition may overlap with parallel edges.

If we consider the planarity constraints introduced in the previous section, we can observe that we do not care about parallel edges as long as they are bridges. However, by the way we introduced edges that we want to draw weakly planar in Section 3.3 they are never bridges. Therefore we only need to discuss how Lemma 4 can be adopted to weakly planar edges:

Lemma 7. *A horizontal edge $e_h \in E_h(f)$ and a vertical edge $e_v \in E_v(f)$ that do not share a common endpoint of a face f of an OC_1 drawing do intersect only in one of their endpoints if at least one of the following conditions is met:*

1. *The vertical edge is intersected by the right endpoint of the horizontal edge (cf. Figure 3.16a).*
2. *The vertical edge is intersected by the left endpoint of the horizontal edge (cf. Figure 3.16b).*
3. *The horizontal edge is intersected by the top endpoint of the vertical edge (cf. Figure 3.16c).*
4. *The horizontal edge is intersected by the bottom endpoint of the vertical edge (cf. Figure 3.16d).*

Now we are able to formulate constraints for the ILP that ensure weak planarity between an edge pair $e_h \in E_h(f)$, $e_v \in E_v(f)$ at a face f where either e_h or e_v should be drawn weakly planar at the given face. Again, we will need the boolean variables defined in (3.12). Also, we will need the case distinction introduced in (3.13). Finally, we will adjust the constraint (3.14), (3.15), (3.16), (3.17) to:

$$horizontal(e_v) - right(e_h) - M \cdot rightOf \geq -M \tag{3.28}$$

$$left(e_h) - horizontal(e_v) - M \cdot leftOf \geq -M \tag{3.29}$$

$$vertical(e_h) - top(e_v) - M \cdot topOf \geq -M \tag{3.30}$$

$$bottom(e_v) - vertical(e_h) - M \cdot bottomOf \geq -M \tag{3.31}$$

where again M is a sufficiently large constant.

For an example, let us discuss the differences in the "left of" cases (3.15) and (3.29). If we set $leftOf = 0$ in both cases we will ignore the logic behind the rest of the constraint. However, if $leftOf = 1$, we will evaluate the positions of the edges. In (3.15) we then demand the vertical edge to be at least one unit distance to the left of the left end point of the horizontal edge. Note that this also is covered by (3.29). Further, (3.29) will also be satisfied if

$$left(e_h) - horizontal(e_v) = 0$$

i.e. the left endpoint of the horizontal edge is located at the same horizontal coordinate as the entire vertical edge (cf. Figure 3.16b). Therefore (3.29) will be satisfied, if the face is drawn weakly planar at the edge pair e_h, e_v. The other constraints work similar.

Unfortunately, it is not enough to just use the constraints (3.28), (3.29), (3.30), (3.31) instead of (3.14), (3.15), (3.16), (3.17) if there is at least one weakly planar edge involved. Consider the configuration shown in Figure 3.17 where the face should be drawn weakly planar at the red edge. The two blue edges would fulfill our current constraints for weak planarity. However, both blue edges are on different sides of the red edges. It is easy to see that configurations like this can be valid with the current constraints while they still violate our understanding of the concept of faces. Therefore we need to be more careful and only use those constraints for weak planarity that we need for a given weakly planar drawn edge and use the normal planarity constraints for the other cases.

3.4.4 Choosing the Correct Constraints for Edge Pairs

Edges along a face in an OC_1 drawing can only be vertical and horizontal hence the face can only be on the left, right, top or bottom of the edge (unless it is a bridge which the dummy edges are not). Now depending on the position of a face f w.r.t. an edge $e \in f$, we can say which cases of weakly planar crossings we want to allow at the edge e on the face f:

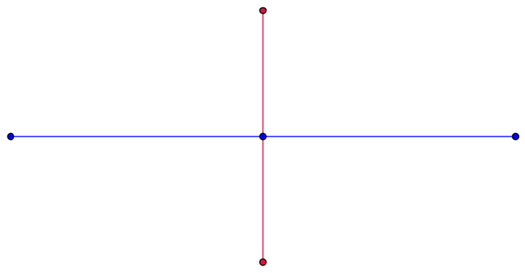


Figure 3.17: An example configuration that may cause problems with the approach so far.

- If the edge is vertical and the face is to the left of the edge, we will allow the "right of" case of weakly planar intersections (cf. Figure 3.16a) which corresponds to the constraint (3.28).[7]
- If the edge is vertical and the face is to the right of the edge, we will allow the "left of" case of weakly planar intersections (cf. Figure 3.16b) which corresponds to the constraint (3.29).
- If the edge is horizontal and the face is at the bottom of the edge, we will allow the "top of" case of weakly planar intersections (cf. Figure 3.16c) which corresponds to the constraint (3.30).
- If the edge is horizontal and the face is on top of the edge, we will allow the "bottom of" case of weakly planar intersections (cf. Figure 3.16d) which corresponds to the constraint (3.31).

However, allowing only weakly planar intersections in one of the four cases is not enough. Before we can explain why, we need to define a classification of vertices w.r.t. faces:

Definition 21 (Concave Vertices and Convex Vertices in OC_1 Drawings)**.** Let $G = (V, E)$ be a graph and $\Gamma_{OC_1}(G)$ be an OC_1 drawing of G. Further let f be a face of G and $v \in V$ a vertex on the face f that is connected to exactly 2 edges e_1 and e_2 along the face f. v is called a *concave vertex* (w.r.t. f) if

- $e_1 \perp e_2$
- the 270 degree angle between e_1 and e_2 is at the same side as f w.r.t.

[7]The case we need to allow is a bit counter-intuitive. If a face is to the left of an edge, than also horizontal edges which we need to check for weak planarity appear on the left of the edge. Then the vertical edge is to the *right of* the vertical edge, thus we need the "right of" case.

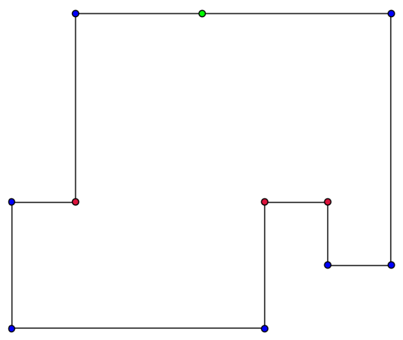


Figure 3.18: An OC_1 drawing with a bounded face containing 7 strictly convex vertices (blue), 1 other convex vertex (green) and 3 concave vertices (red).

to e_1 and e_2.

If v is not concave, we call it a *convex vertex*. If v is a convex vertex, we call it *strictly convex* if $e_1 \perp e_2$ (i.e. the 90 degree angle between e_1 and e_2 is at the same side as f w.r.t. to e_1 and e_2).

For an illustrative example refer to Figure 3.18.

At concave vertices of the face, we may need to allow the configuration depicted in Figure 3.19 in order to not reject valid weakly planar drawings: The face should be drawn weakly planar at the two red edges. Therefore we want to allow the blue edge to be drawn at the same vertical coordinate as the red horizontal edge while the red vertical edge lies between the two end points of the blue one. When we look at the four cases of (weak and normal) planarity for the pair of vertical red and blue edge, we will observe, that only the bottom of case of weak planar intersections is satisfied. We can generalize this:

- If a vertical edge's bottom endpoint is a concave vertex of the face and the face should be drawn weakly planar at the vertical edge, then we will use the constraint for weakly planar intersections for the bottom of case (3.31) instead of the constraint for planarity for the bottom of case (3.17).
- If a vertical edge's top endpoint is a concave vertex of the face and the face should be drawn weakly planar at the vertical edge, then we will use the constraint for weakly planar intersections for the "top of" case (3.30) instead of the constraint for planarity for the "top of" case (3.16).
- If a horizontal edge's left endpoint is a concave vertex of the face and the face should be drawn weakly planar at the horizontal edge, then we will use the constraint for weakly planar intersections for the "left of" case (3.29) instead of the constraint for planarity for the "left of" case (3.15).

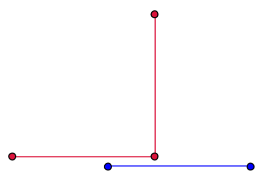
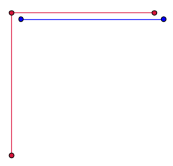


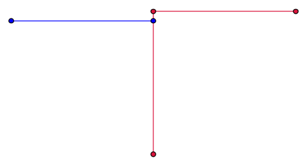
Figure 3.19: A concave vertex on a face with its 2 edges of the cyclic order of the face (colored red). The face should be drawn weakly planar at the two red edges. Therefore the blue edge which also is part of the face can be drawn on the same vertical coordinate as the horizontal red edge (drawn distinct for better readability).

- If a horizontal edge's right endpoint is a concave vertex of the face and the face should be drawn weakly planar at the horizontal edge, then we will use the constraint for weakly planar intersections for the "right of" case (3.28) instead of the constraint for planarity for the "right of" case (3.14).

Next, we need to discuss strictly convex vertices (as the dummy edges we insert never meet in normal convex vertices). Figure 3.20 shows possible configurations between two edges where the face should be drawn weakly planar (red) and another edge of the same face (blue): In Figure 3.20a which shows a valid configuration we can see that between the blue edge and the red vertical edge both the "top of" and the "left of" cases of weakly planar intersections are satisfied. We already stated that the "left of" case shall be possible for edges like the red vertical edge that have the face to the right side. The configurations in Figures 3.20b and 3.20c show us that in contrast to concave vertices we should not allow the "top of" case of weakly planar intersections to be allowed as it is fulfilled in both configurations. However the blue edge in Figure 3.20b also satisfies the forbidden "right of" case of weakly planar intersection whereas the blue edge in Figure 3.20c still "cuts" the red vertical edge. Therefore we have to require planarity in the "top of" case (note that if the "left of" case of weak planar intersections is fulfilled like in Figure 3.20a we will still accept the solution). Again, let us generalise our findings:

- If a vertical edge's bottom endpoint is a strictly convex vertex of the face and the face should be drawn weakly planar at the vertical edge, then we will still use the constraint for planarity for the bottom of case (3.17).
- If a vertical edge's top endpoint is a strictly convex vertex of the face and the face should be drawn weakly planar at the vertical edge, then we will still use the constraint for planarity for the "top of" case (3.16).
- If a horizontal edge's left endpoint is a strictly convex vertex of the face


(a) Valid configuration.


(b) Invalid configuration.

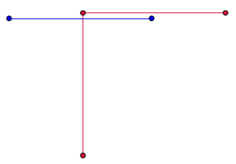

(c) Invalid configuration.

Figure 3.20: A convex vertex on a face with its 2 edges of the cyclic order of the face (colored red). The face should be drawn weakly planar at the two red edges. Therefore the blue edge which also is part of the face can be drawn on the same vertical coordinate as the horizontal red edge (drawn distinct for better readability). However, as the **(b)** and **(c)** already indicates, also horizontal coordinates matter here.

and the face should be drawn weakly planar at the horizontal edge, then we will still use the constraint for planarity for the "left of" case (3.15).

- If a horizontal edge's right endpoint is a strictly convex vertex of the face and the face should be drawn weakly planar at the horizontal edge, then we will still use the constraint for planarity for the "right of" case (3.14).

Note that we can simply set the required weak planarity cases as a property for the dummy edges (instead of the combination of dummy edge and face) since weak planarity for an edge is only required at one of the faces it is part of.

To complete our discussion on constraints for weak planarity we still need to check what happens if both the horizontal edge e_h and the vertical edge e_v are edges at which their face shall be drawn weakly planar. By the way we inserted the dummy edges each of the edges will demand for at most two cases of weak planarity constraints (that both apply in edge pairs with edges that are not drawn weakly planar). If both e_h and e_v demand for the same two cases then those cases will also apply. If the demanded cases differ however, we will only use the weak planarity constraints for all the cases that are demanded by both e_h and e_v. This is absolutely needed since otherwise we could again get a bad configuration as in Figure 3.17: Consider that the red edge may demand for the "left of" weakly planar case while both blue edges demand the "right of" cases instead. Then by joining the demanded cases instead of finding the intersection of them we allowed the sequence of two blue edges to really cross a boundary of the face which is not what we intended to do.

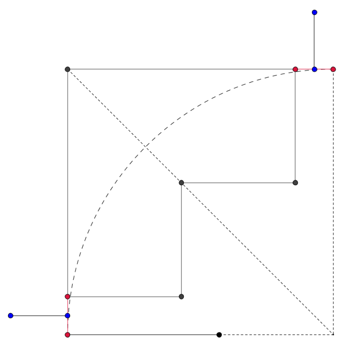


Figure 3.21: Zero length edges of a quarter circle arc (red) and other edges that satisfy weak planarity as demanded by the zero length edges (blue).

Finally we also need to look back at the zero length edges as illustrated in Figure 3.4b. First, we need to consider that the zero length edge will have length zero in the final drawing. Therefore both its endpoints overlap and their indicident edges should be treated as if they had a common endpoint, i.e. they cannot overlap and no constraints should be introduced for edge pairs of neighbored edges.

Second, those edges are really useful: We will let the zero length edge demand for weak planarity such that a perpendicular edge may touch the zero length edge just at the outside of the arc, i.e. at one of the endpoints of the arc (c.f. Figure 3.21). However as a special case we will only allow this when the second edge of the edge pair also is a dummy edge to prevent real edges to coincide with the real vertex. On the other hand, when 2 arcs have a common end point, then at the common vertex the weakly planar face is bounded by two zero length edges (one of each arc): Here the zero length edge really becomes useful. Another edge cannot demand for weakly planar constraints in such a way that it could cut the end point of the arc as the zero length edge is also a dummy edge that demands only for this one particular case of weakly planar constraints that is not critical as it cannot be fulfilled by any edge inside the face.

Now that we know when to use which constraints we have an ILP whose solutions describe OC_1 drawings which can be transformed into an SC_1 drawing.

3.4.5 An Objective Function for Minimizing the Total Edge Length

The only thing remaining to finish our ILP now is defining the objective function we want to minimize. Let Arc_{90} be the set of quarter circle arcs, Arc_{180} the set of half circle arcs and Arc_{270} the set of three-quarters circle arcs. Further let $E_{real} = E^* \cap E$, i.e. the set of edges that were part of the original graph and remain unchanged in the graph where we replaced the bent edges. Then the following objective function sums up the total edge length:

$$\sum_{e \in E_{real}} l(e) + \sum_{a \in Arc_{90}} \frac{\pi}{2} r(a) + \sum_{a \in Arc_{180}} \pi r(a) + \sum_{a \in Arc_{270}} \frac{3\pi}{2} r(a) \qquad (3.32)$$

The rest of the approach is straight-forward. We will first solve the ILP minimizing the objective function 3.32.

If a solution could be found, we will fix the position of a single vertex and draw the entire OC_1 drawing represented by the solution of the ILP. In this OC_1 drawing the inner and outer approximations of arcs reserve enough space to draw the arcs in between them such that no other edge will be intersected. Finally we can remove all dummy edges and vertices and draw the approximated arcs instead.

Chapter 4

More Sophisticated Arc Approximations

4.1 Limitations of the *Simple Approximation*

As we shall see in Chapter 5, the simple approximation already performs quite well on commonly used benchmark test sets of graphs. However it also comes with some limitations that we will try to resolve in this chapter. In particular we will now discuss 3 orthogonal shapes which cannot be drawn with the simple approximation either at all or obviously non-optimal and further investigate what is going on. For the convenience of the reader we will draw them as a SC_1 drawing instead of an orthogonal shape.

Figure 4.1 shows the drawing of a graph where one node is incident to two quarter circle arcs for which the approach failed to draw so far. In the figure, the inner simple approximations of those arcs are also drawn in red. Obviously, in this case due to the way we defined the inner approximations they necessarily need to cross in at least one point. Therefore our ILP is unable to find a feasible solution here. However, it will turn out that this problem can be easily avoided as discussed in Section 4.2.

Next, we will discuss the two drawings in Figure 4.2. Both drawings are of the same orthogonal shape. The drawing in Figure 4.2a has been succesfully computed by the ILP using the simple approximation. Here we can observe that the arc seems quite empty. This results from the definition of the simple approximation which will always favor rather quadratic contents. The other drawing in Figure 4.2b is optimal w.r.t. total edge length instead [1] and its total

[1]The radius of the single arc cannot be smaller as the nodes between the two endpoints of the arc have to be placed vertically above each other as shown in the drawing. If you look closely you may observe that the arc actually cuts one node. However the nodes are only drawn as squares for better readability and if you would draw them as a single point (at the center of the square) there would be no intersection.

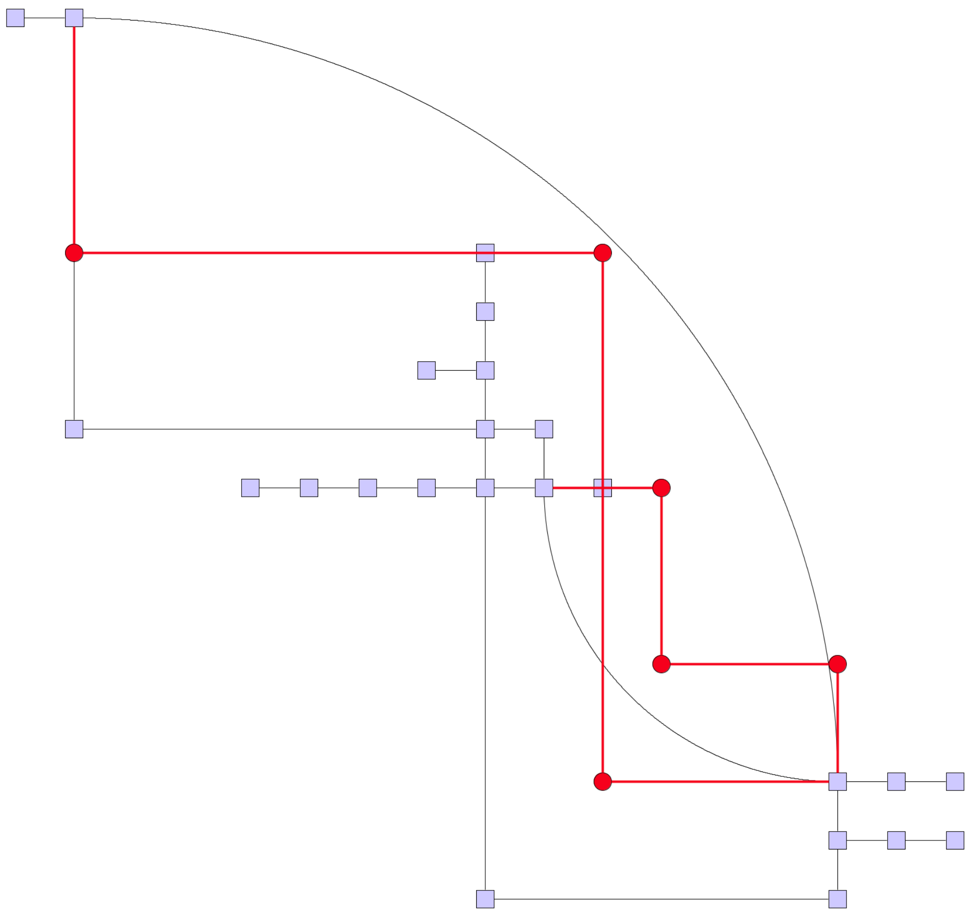
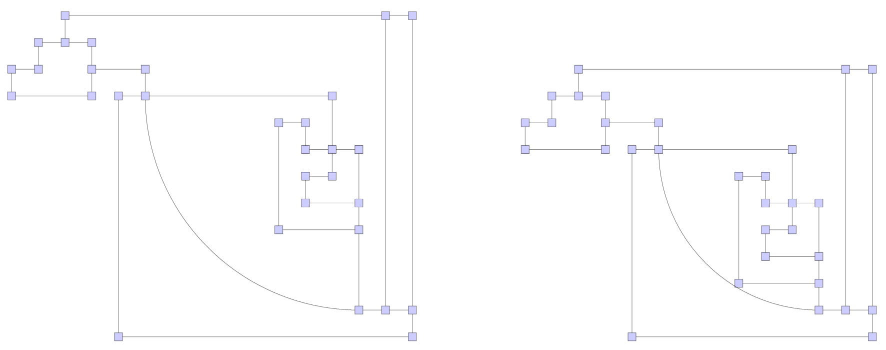


Figure 4.1: A graph with 2 arcs that have a common endpoint. The red edges are the inner simple approximation of the arcs.



(a) Simple approximation ILP result. **(b)** Optimal result.

Figure 4.2: A graph that is drawn too large with the simple approximation.

edge length is about 15 percent smaller compared to the simple approximation result.

Finally, one can even construct orthogonal shapes which are impossible to draw with the simple approximation approach because it forces all the contents of an arc in a square that starts quite a distance from the endpoints of the arcs. Figure 4.3 shows the optimal SC_1 drawing of such a shape. In the figure all the red edges have same length due to the constraints imposed by the arcs. However, one of them (the red edge drawn dashed in the top left corner) connects the interior of the arc with the big arc, i.e. it overlaps completely with the *connector edge*. In the simple approximation model therefore it would need to be drawn way longer than in the optimal drawing (in fact of length $\lceil(1 - \frac{1}{\sqrt{2}})R\rceil$ where R is the radius of the big arc). Unfortunately then also all the other red edges would need to be drawn with this length which leads to the situation that the arc of radius R will contain another arc that has radius $\lceil(1 - \frac{1}{\sqrt{2}})R\rceil$ as well (in fact this large arc is not even contained in the biggest arc since the arcs are "stacked" in this orthogonal shape). In the end, intuitively we will end up in an infinite loop. Our improved approach presented in Section 4.3 will be able to draw the orthogonal shape depicted in Figure 4.3 in a SC_1 layout and will achieve the better result for the previously discussed shape as depicted in Figure 4.2b.

4.2 A New Model for 180° and 270° Circular Arcs

As we have seen in Figure 4.1 so far we cannot draw a shape that has two circular arcs which share a common endpoint if they are routed as shown in the figure. If we could ensure that we do not do any mistakes when we ignored intersections between the approximations of arcs that share a common

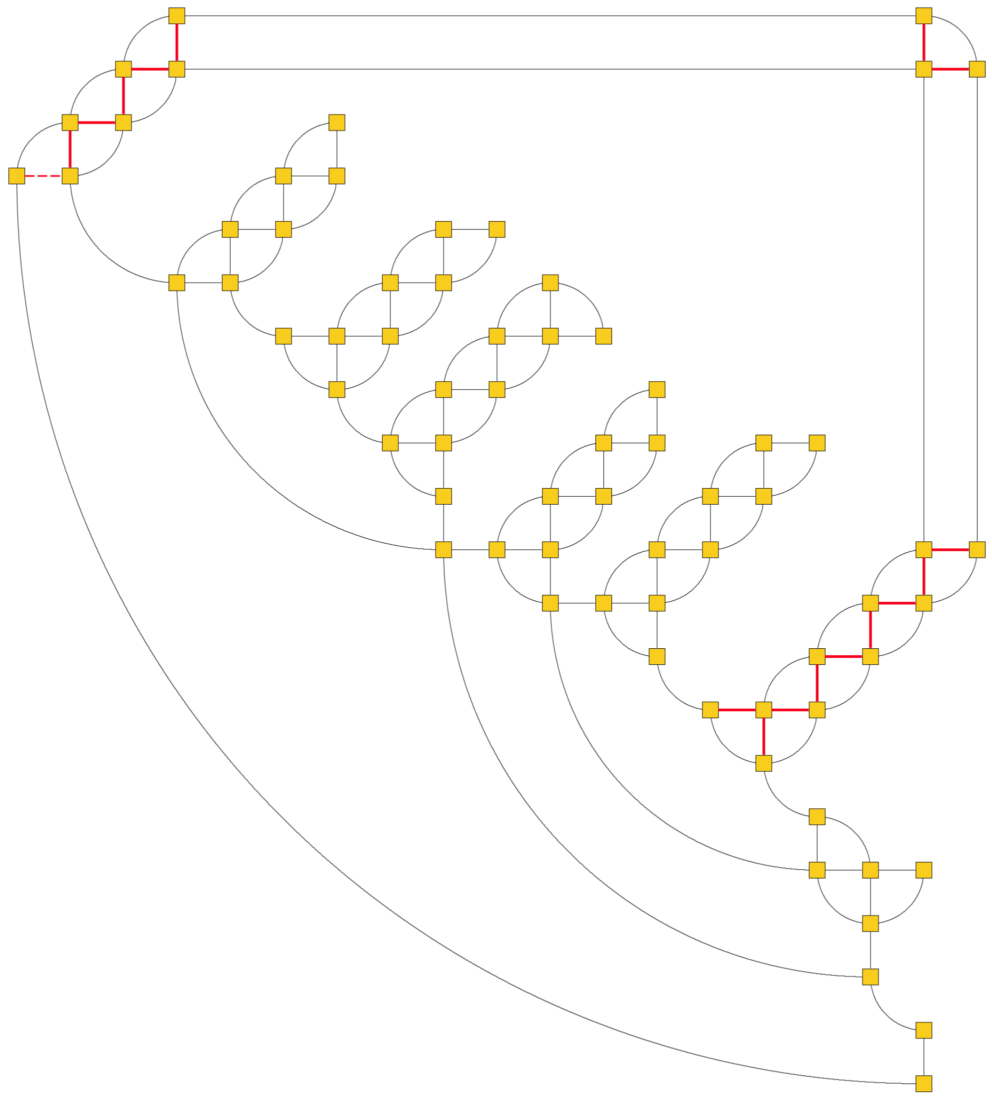


Figure 4.3: A graph that cannot be drawn with the simple approximation model drawn with minimum edge length.

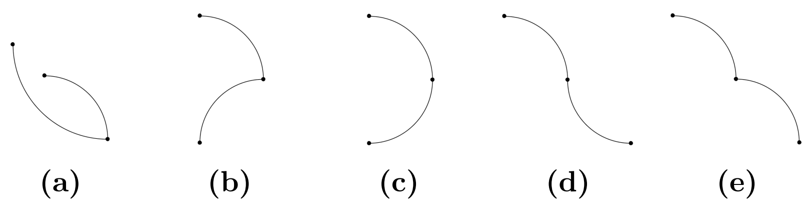


Figure 4.4: Possible configurations between two quarter circle arcs sharing a common endpoint.

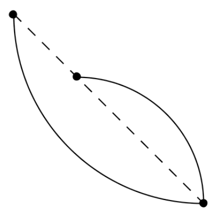


Figure 4.5: The 3 nodes in this configuration need to be aligned on a diagonal.

endpoint then we could draw drawings like in Figure 4.1 even with the simple approximation ILP. Luckily, we can formulate the following:

Lemma 8. *If two quarter circle arcs share a common endpoint they cannot cross except in their common endpoint.*

Proof. First of all let us recall that in smooth orthogonal drawings we require that each port of a node is used by at most one edge. Therefore the only possible configurations for the two quarter circle arcs are those shown in Figure 4.4 (and the same configurations rotated or mirrored). It is easy to see that all the cases except for the one depicted in Figure 4.4a are not interesting since the arcs leave the common node in different directions.

In the one interesting case however we also know that all nodes are located on a diagonal (cf. Figure 4.5). This means, that one arc is completely above the diagonal while the other one is located completely below the diagonal. Since we will draw the nodes of the graph distinct, also in this case the two arcs will not cross except for their common endpoint. □

Unfortunately we cannot extend Lemma 8 to half circle and three-quarters circle arcs. However, we can change the way we model those larger arcs in our ILP to still use Lemma 8 in order to overcome the problems we faced e.g. when drawing the graph in Figure 4.1.

Instead of directly introducing an approximation for half circle and three-quarters circle arcs as shown in Figure 3.3 we can split those arcs in 2 and 3 quarter circle arcs, respectively, which we then connect with a dummy vertex

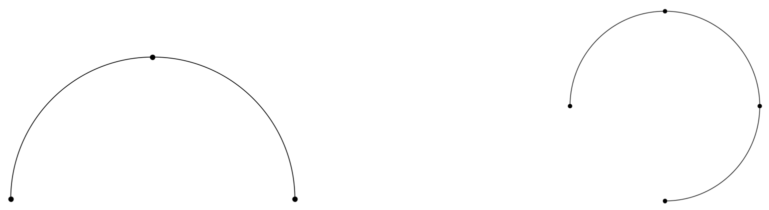

(a) Split half circle arc. **(b)** Split three quarters circle arc.

Figure 4.6: Larger arc types split in 90° arcs.

and require to have the same radius (cf. Figure 4.6). We then can use the approximation for the quarter circle arcs for all the components of the larger arc types and still get the same results as before when we use a common variable for the radius of all 90° arcs that form a larger arc. Further we can now also apply Lemma 8 which enables us to draw drawings like the one in Figure 4.1.

4.3 A Staircase Approximation

Next we will try to improve the simple approximation itself such that less space is reserved for drawing the arc later as we have seen that reserving too much space (as the the simple approximation does) can result in more (cf. Figure 4.3) or less (cf. Figure 4.2) severe issues. Also recall Figure 3.3a: The entire face bounded by the approximation dummy edges will later not contain any other edge and the figure already suggests that we waste quite a lot of space in the drawing that may even be needed to find a feasible drawing.

Inspired by the simple approximation we will introduce the following *staircase approximation* that has a parameter $n_{staircase} \in \mathbb{N}^+$ that is part of the input (cf. Figure 4.7):

- The simple inner approximation edges are split into $n_{staircase}$ equally sized [2] staircase *inner approximation edges* (red in the figure) each.
- Those inner approximation edges are connected with each other and an endpoint of the arc by $n_{staircase}$ *inner connector edges* (green in the figure). Their lengths are determined such that they are as short as possible while the arc is not crossed by the inner approximation edges.
- There are $n_{staircase}$ *outer approximation edges* (blue in the figure). Their lengths are such that the summed length of the first k outer approximation edges is longer than the summed length of the first k inner approx-

[2]If the length of the simple inner approximation edge can be divided by $n_{staircase}$ the new edges will really have equal lengths. Otherwise their lengths may differ by 1.

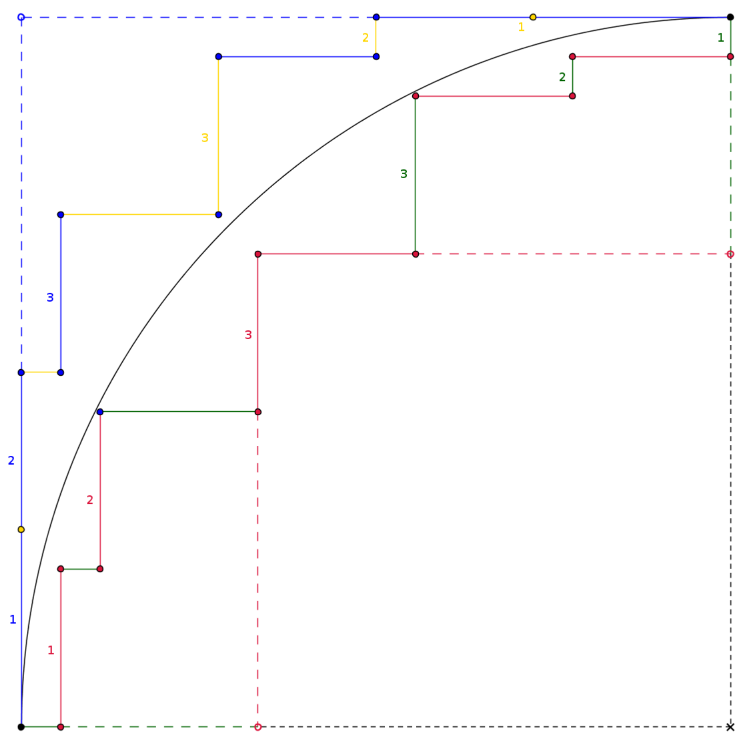


Figure 4.7: A staircase approximation with $n_{staircase} = 3$. Note that at the outer approximation, the yellow nodes represent two nodes connected by an outer connector edge that for the given radius is of length 0. The dashed edges show the old simple approximation edges.

imation edges for all $1 \leq k \leq n_{staircase}$ enumerating the edges from the arc endpoints (as in the figure).

- There are $n_{staircase}$ *outer connector edges* (yellow in the figure). Their lengths are such that the summed length of the first k outer connector edges is shorter than the summed length of the first k inner connector edges for all $1 \leq k \leq n_{staircase}$ enumerating the edges from the arc endpoints (as in the figure).

Note that for $n_{staircase} = 1$ the staircase approximation is identical to the simple approximation.

Now that we have an informal idea of what the staircase approximation should look like, we can also formulate more precisely how long we want the edges to be:

Lemma 9. *Let e_i^k denote the k-th inner approximation edge, e_o^k the k-th outer approximation edge, e_{ic}^k the k-th inner connector edge and e_{oc}^k the k-th outer connector edge with $1 \leq k \leq n_{staircase}$. Further let $l(e) \geq 0$*

denote the length of an edge.

For a given arc with radius R the following equations hold for any k with $1 \leq k \leq n_{staircase}$:

1. $\sum\limits_{i=1}^{k} l(e_i^k) = \lfloor \frac{\alpha}{\sqrt{2}} R \rfloor$
2. $\sum\limits_{i=1}^{k} l(e_{ic}^k) = \lceil (1 - \sqrt{1 - \frac{\alpha^2}{2}}) R \rceil$
3. $\sum\limits_{i=1}^{k} l(e_o^k) = \lceil \frac{\alpha}{\sqrt{2}} R \rceil$
4. $\sum\limits_{i=1}^{k} l(e_{oc}^k) = \lfloor (1 - \sqrt{1 - \frac{\alpha^2}{2}}) R \rfloor$

where $\alpha := \frac{k}{n_{staircase}}$.

Proof. α exactly is the ratio that we wanted to use for the inner approximation edges. Therefore the first equation is correct as in the simple approximation the inner approximation edge's length was $\lfloor \frac{1}{\sqrt{2}} R \rfloor$. Also note that we need to always round down here since rounding up may cause the edge to cut the arc.

For the second equation consider Figure 4.8: Given an α there exists one point on the arc that is $\frac{\alpha}{\sqrt{2}} R$ above the bottom of the arc. If we draw an edge down (red edge) we cut the horizontal edge such that the left part (green) is equal to the sum of the edge lengths of the inner connector edges $\sum_{i=1}^{k} l(e_{ic}^k)$ which we want to compute. Also we have created an right triangle where we know two edge lengths (as given in the figure). Therefore we can compute the blue edge's length $l(e_b)$ with Pythagoras' theorem:

$$R^2 = l(e_b)^2 + \left(\frac{\alpha}{\sqrt{2}} R \right)^2 \implies l(e_b) = \sqrt{1 - \frac{\alpha^2}{2}} R$$

Since the blue and the green edge sum up to R, we can formulate that $\sum_{i=1}^{k} l(e_{ic}^k) = \lceil (1 - \sqrt{1 - \frac{\alpha^2}{2}}) R \rceil$ where the ceiling is necessary to make sure the length is the next integer length that prevents crossings.

Equations 3 and 4 are just a formulation of the fact that we wanted the outer approximation's edge lengths to depend on the lengths of the inner approximation - by replacing floor with ceiling function and vice versa we make the outer approximation edges longer than the inner approximation edges and the outer connector edges shorter than the inner approximation edges. □

Another improvement we will use from now on to improve the quality of our drawings is to also draw the face outside of the arc weakly planar at the

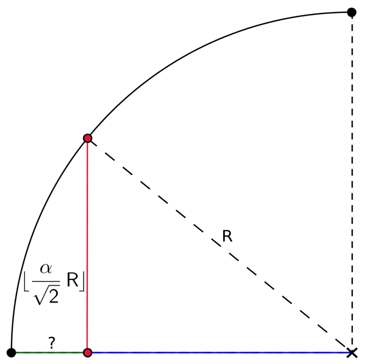


Figure 4.8: Computation for the inner connector edge lengths.

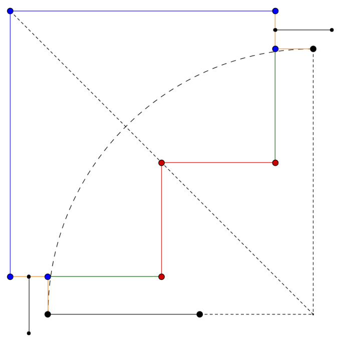


Figure 4.9: Arc approximation with two zero length edges at each endpoint (orange). The only weak planarity case admitted for new zero length edges when paired with other approximation edges is indicated with the additional black edges in the drawing.

outer approximation and outer connector edges. We can do so since the arc is not touching the outer approximation edges except at its endpoints (cf. Figure 4.7) which we still have to avoid from happening. Now we can benefit from treating the zero length edges as a special case: Approximation edges may be drawn passing through the endpoints of arcs (since approximation edges will not appear in the final drawing) while edges also present in the final drawing should not cross the endpoints of arcs. However, since we have only one zero length edge at each endpoint of the arc we can only take care of one type of other edges (horizontal or vertical) that may cross this endpoint. But there is a simple solution for this issue: we can just insert another zero length edge with similar properties but routed perpendicular to the already existing (cf. Figure 4.9).

To conclude our discussion of the staircase approximation, we only need to modify our ILP to the model. First of all, all edges e_o of the outer approximation (including the outer connector edges) will satisfy:

$$l(e_o) \geq 0 \tag{4.1}$$

Further we need to adopt constraints (3.5) to (3.8) to the new model: Let again e_i^k denote the k-th inner approximation edge, e_o^k the k-th outer approximation edge, e_{ic}^k the k-th inner connector edge and e_{oc}^k the k-th outer connector edge with $1 \leq k \leq n_{staircase}$. Then we will add the following constraints to our ILP in order to implement Lemma 9:

$$\frac{\alpha}{\sqrt{2}}R - \sum_{i=1}^{k} l(e_i^k) \geq 0 \tag{4.2}$$

$$\sum_{i=1}^{k} l(e_{ic}^k) - \left(1 - \sqrt{1 - \frac{\alpha^2}{2}}\right) R \geq 0 \tag{4.3}$$

$$\sum_{i=1}^{k} l(e_o^k) - \frac{\alpha}{\sqrt{2}}R \geq 0 \tag{4.4}$$

$$\left(1 - \sqrt{1 - \frac{\alpha^2}{2}}\right) R - \sum_{i=1}^{k} l(e_{oc}^k) \geq 0 \tag{4.5}$$

where $\alpha := \frac{k}{n_{staircase}}$.

The staircase approximation successfully solves the issues we had observed with the simple approximation. In fact, the drawing Figure 4.2b was drawn of the staircase approximation and $n_{staircase} = 25$. Also, even for $n_{staircase} = 2$ the staircase approximation can draw the orthogonal shape depicted in Figure 4.3 (however not as optimal as in the figure). On the other hand, at first glance it may seem difficult to argue about whether the staircase approximation will find a correct solution if a perfect smooth orthogonal drawing for a given orthogonal shape. Therefore in the next section we will introduce another way to approximate the arcs of the input shape that is motivated by properties of any SC_1 drawing.

4.4 A Provably Optimal Approximation

When we approximate arcs with sequences of horizontal and vertical line sequences we may also wonder if we can do so in such a way that it is totally equivalent to drawing the arc. Of course, if this is possible, we also would like to know how complex such an approach is w.r.t. time and space consumption so we can argue about the practical usefulness.

First of all, let us recall, that there are three different kinds of intersections in the final drawing that we want to prevent from happening:

1. Intersections between two straight-line edges are easily prevented by the noncrossing constraints of the ILP.

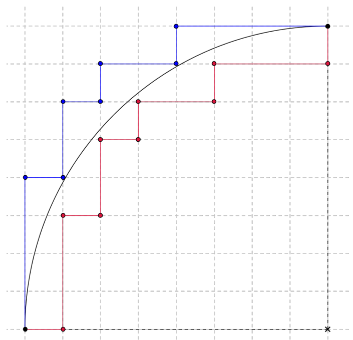


Figure 4.10: Minimally optimal approximation.

2. Intersections between a straight-line edge and an arc need to be prevented by the fact, that the straight-line edge is at most laying on the outer or inner approximation of the arc (i.e. weakly planar case).

3. Intersections between arcs and arcs are prevented such that their approximations are at most intersecting in a weakly planar fashion.

For Cases 2 and 3 we need to discuss how we can replace the arcs in such a way that if we replace the arcs of a given SC_1 drawing we will end up with an OC_1 drawing which is planar or weakly planar at faces where this is allowed.

It is easy to see that we can replace each arc by horizontal and vertical line segments, so that each integer point inside the arc is either inside the face bounded by the inner approximation or overlaps with the inner approximation (cf. Figure 4.10). The same of course also shall be true for the outer approximation. We will call this type of approximation *minimally optimal* as we only reserve as much space as absolutely needed with the minimum number of straight-line edges.

Lemma 10. *The* minimally optimal approximation *of a quarter circle arc with radius R consists of* $\Theta(R)$ *edges.*

Proof. We can split the approximation into two parts along the diagonal of the arc (cf. Figure 4.11) with the same number of edges. Therefore, we will only discuss the half of the arc approximation to the bottom left of the diagonal in the figure. For the inner approximation (red) for each unit of horizontal distance between the bottom left arc endpoint and the last integer point inside the arc on the diagonal we can observe, that we need a horizontal and a vertical edge. As shown before, the last integer point has a horizontal distance of $\lceil(1-\frac{1}{\sqrt{2}})R\rceil$ from the endpoint of the arc. Note that due to parity we have one edge in the bottom left half that reaches into the other half of the arc. This edge later will be assigned length 0. For the outer approximation (blue) we insert

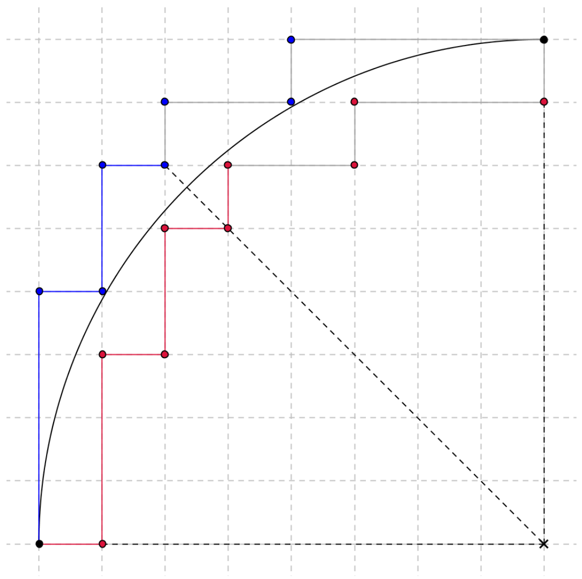


Figure 4.11: One half of the minimally optimal approximation.

2 edges (one vertical and one horizontal) for each unit of horizontal distance between the bottom left arc endpoint and the first integer point outside the arc on the diagonal. It is easy to see, that here we insert $2\lfloor(1-\frac{1}{\sqrt{2}})R\rfloor$ edges. Therefore in total we need $2\lceil(1-\frac{1}{\sqrt{2}})R\rceil + 2\lfloor(1-\frac{1}{\sqrt{2}})R\rfloor = \Theta(R)$ edges. □

The problem we are still facing is that we do not know the radii of arcs in advance. But fortunately, we can oversample instead. Oversampling here means that we compute the approximation and the exact edge lengths for a very large arc of radius R^*. For smaller arcs of radius $R << R^*$ we still use the same approximation and scale the edge lengths by a factor of $\frac{R}{R^*}$.[3]

Lemma 11. *Oversampling, i.e. using the approximation and constraints for a larger radius $R^* >> R$, yields correct results for the radius R.*

Proof. As we have already shown half of the approximation edges (cf. Figure 4.11) when not oversampling have length 1. The lengths of the other edges (for the bottom left part of the approximation) depend on the vertical coordinates of the arc at multiples of 1 unit of distance right of the bottom left arc endpoint. When we now scale an minimally optimal approximation for a radius R^* down to R we have more staircase steps with smaller distances to each other (cf. Figure 4.12; each of the points will contribute 2 edges for the inner approximation). It is easy to see that we have enough edges to emulate

[3]The division by R^* can be done when formulating the ILP, the multiplication with R then yields a linear constraint.

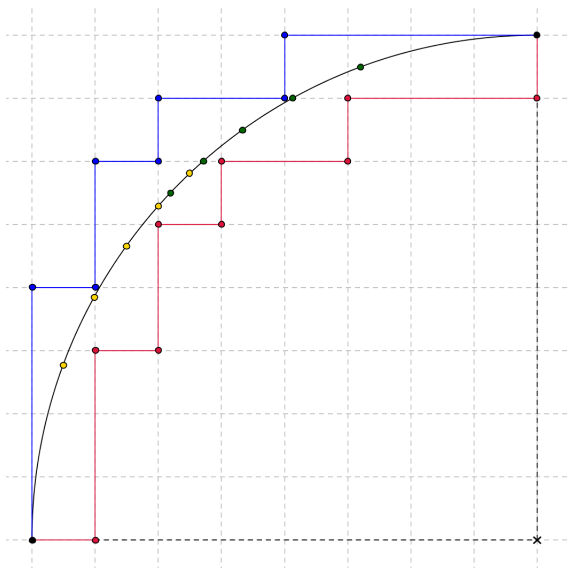


Figure 4.12: Oversampling in the minimally optimal approximation. Yellow points are part of the bottom left half of the approximation, green points are part of the top right half of the approximation.

the correct minimally optimal approximation for radius R (where some of the edges will have length 0 since we have to many).

The other important part is, that the distances between sample points are smaller when using the approximation for R^*. If $R^* >> R$ we can safely assume that the oversampling is so dense that for every vertex v in the minimally optimal approximation for R we can find a vertex in the minimally optimal approximation for R^* whose next integer point in the direction of the center of the arc is where we would draw v. □

Now that we know we can oversample, we would like to compute the largest possible radius R_{max} that is necessary to draw in a minimum edge length SC_1 drawing of a graph with n nodes. Since in Lemma 11 we assumed $R^* >> R$, it would also be useful to overestimate R_{max} so that this assumption is satisfied.

Lemma 12. *The arcs in an SC_1 drawing of a graph G with n nodes and minimum total edge length for a given orthogonal shape have a maximum radius of $R_{max} < 3^n$.*

Proof. We will prove this by induction. First consider a graph with $n = 2$ nodes. Here we can only have a single arc of radius 1 (cf. Figure 4.13a). Clearly, $1 < 3^n = 9$.

Let us assume, that for the previous value of n the entire graph completely filled a square shape of area $3^n \times 3^n$ (cf. Figure 4.13b). Also let us assume that

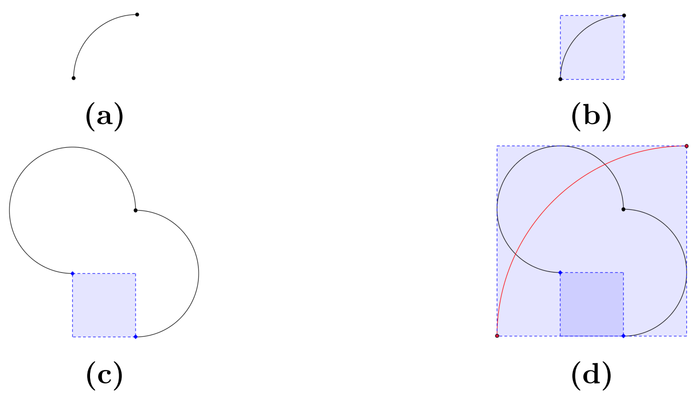


Figure 4.13: Proof for an upper bound of R_{max}.

for any given integer coordinate on the border of the square we can connect to it with an edge from the outside.

Then for $n + 1$ nodes we want to place the $(n + 1)$-th node outside of the square shape for n nodes. This is a safe assumption since if we would place it inside of the square we could have chosen another order of the nodes such that the next node would always be outside of the graph of the previous nodes (cf. the canonical representation introduced in [DFPP90]).

If we want to connect the new node to the square shape and by doing so maximize the total area required, our "best" bet is to place and connect the new node as shown in Figure 4.13c. Placing the new node over one of the sides of the square is worse as we cannot build the half circle arc then to place the vertex 3^n over the old graph whereas if we want to place it on the extension of the diagonal of the square we also cannot force the vertex to be placed further away than a smaller number of units as building two large arcs then becomes difficult.

Finally, we build the bounding square again which now is of size $3^{n+1} \times 3^{n+1}$ and claim that we cannot get a larger arc than the one connecting the endpoints of the diagonal of the square shape (cf. Figure 4.13d). This arc is of radius 3^{n+1}.

□

Now that we limited the number of edges we create when oversampling, we can prevent crossings between arcs and straight-line edges with the *minimally optimal approximation*. However, in order to show that this kind of approximation will always find an SC_1 drawing if such a drawing exists, we also need to consider crossings between two arcs.

In general, two arcs may be arbitrarily close to each other. This makes it a bit harder than preventing crossings between arcs and straight-line edges while also not ignoring correct solutions since if two arcs are drawn very close to each other, their approximations may cut. On the other hand, it is easy to see that if one drawing of minimum edge length exists, there also exist infinitely many other drawings where every edge length (of straight-line edges) and every radius is multiplied with a scalar integer factor $s \geq 1$. If s increases, the minimum distance between any two arcs decreases. If s becomes large enough, the approximations of any two arcs will not intersect anymore and the s times larger drawing is a feasible solution of the ILP using the minimally optimal approximation. We can formulate our findings as follows:

Theorem 1. *There exists an exponential time and exponential space reduction from the problem of finding a SC_1 drawing given an orthogonal shape to ILP.*

Proof. Oversampling the input shape with the minimally optimal approximation according to Lemmata 10 and 12 needs exponential many edges that need to be stored and created hence the exponential time and exponential space consumption. □

The only thing that prevents us from finding the drawing with minimum edge length are arcs that are really close to each other.

Theorem 2. *Let $\Delta < 1$ be the minimum distance between any two arcs in the drawing with minimum edge length in either x or y direction. Then if we scale the grid (in the constraints) by a factor of $\frac{1}{\Delta}$ the minimally optimal approximation ILP will compute the minimum edge length drawing.*

Proof. If we scale the grid by $\frac{1}{\Delta}$ the new minimum distance between any two arcs is 1. If two arcs have a distance of at least 1, their minimally optimal approximations at most overlap (cf. Figure 4.10; all integer coordinates inside respectively outside of the arc are not reserved for arc drawing). □

4.5 Further Considerations

As we have seen in the previous section, if we oversample we can ensure to find a drawing, if one exists. However, in a practical application we cannot hope to be allowed to create exponentially many approximation edges if we want to achieve a reasonable running time[4]. In a practical context, we rather need to subsample, i.e. using an approximation of a smaller radius on a larger radius. It turns out that if we have to subsample, the minimally optimal

[4]While already the creation of all those edges would take exponential, the runtime of the ILP solver would be even worse.

approximation performs worse than a staircase approximation with the same number of edges (cf. 4.14). Therefore, in the practical evaluation we will focus on the staircase approximation.

Also we do not lose the guarantee of finding a solution if one exists:

Corollary 1. *The staircase approximation ILP will find a solution for the SC_1 drawing problem for a given orthogonal shape if a feasible drawing exists.*

Proof. The reason for this is, that also the staircase approximation may reserve the same area as the minimally optimal approximation does. When we recall Figure 4.3, we can observe that the distance between sampling points increases until we reach the diagonal of the arc. Therefore we only have to ensure, that the last connector edge has a length smaller than 1 in an arc of radius R_{max}. We will skip the computation of a feasible $n_{staircase}$ as it is not practically relevant due to exponential runtime and exponential space requirements. □

Although we could guarantee to find solutions in Theorem 1 and Corollary 1 there are two assumptions underlying those, that we have not spoken about yet:

1. Arcs may not cross through points $(x, y) \in \mathbb{Z}^2$ unless the points are those where the tangent of the arc is either horizontal or vertical. This depends on the actual radius R of the arc. Not for all radii, we can find such a point.

   In order to ensure planarity then, we would first need to ensure, that the approximations are drawn as close to the arc as possible. So far we solve the floor and ceiling functions in the edge lengths computations by greater equal constraints and the fact, that reserving more space for the arc is not essential as long as the minimum required space is part of the solution space.

   Then we could use *bigM* formulations and indicator constraints to check if outer and inner approximation overlap (which is the case if there is an integer coordinates point). This would enable us to decide in this specific case to use strict planarity constraints instead.

   Since this case could not be observed in the practical evaluation, it is not implemented in the practical implementation accompanying this thesis as more constraints also mean a worse performance. However, one should know that this case may occur.

2. The second issue for the staircase approximation is parity. In the staircase approximation we assume that the last integer point of the diagonal

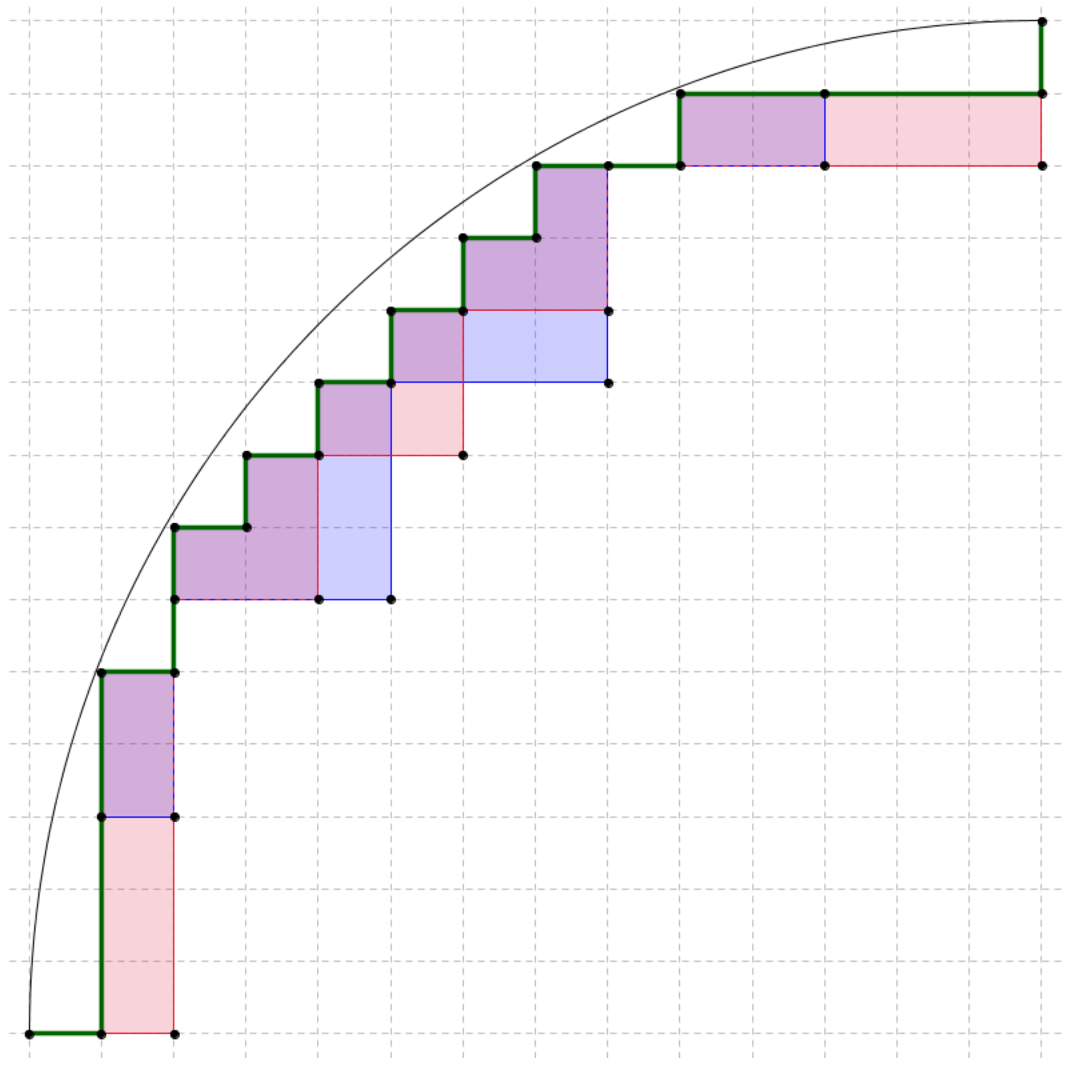


Figure 4.14: Subsampling an arc with the minimally optimal approximation and the staircase approximation with the same number of edges as in the subsampling. The green approximation is the minimally optimal approximation for the radius of the arc. The red area shows which parts are not covered by the subsampling whereas the blue area shows which parts are not covered by the staircase approximation. Purple areas occur where the two other cases overlap.

is closer to the arc than its two neighbored vertices on the inner approximation (cf. Figure 4.7). However this may not necessarily be the case (cf. Figure 4.10).

As we want to subsample in a practical context and therefore reserve more space for the arc drawing than absolutely necessary, this difference is not relevant. Also note, that since we do not know which parity will be needed in advance, this will often lead to edges of length zero (depending on the parity) which still need to be checked for planarity constraints.

While it may be disappointing that we again have to restrict our guarantees of finding a solution if one exists, we should rather focus on the gains from those 2 simplifications: In a practical context problems may never be such that only one family of solutions (scaled versions of a single drawing) exist, however the speed-up due to less constraints will be relevant every time we will apply the ILP approach.

Chapter 5

Practical Evaluation

5.1 Experimental Setup

The staircase approximation ILP as introduced in the previous chapters was implemented in JAVA[1]. The graph drawing framework YFILES[2] is used for loading and saving graph files as well as managing and drawing graphs once loaded in the main memory from a file. For solving the ILP, the solver GUROBI[3] was chosen because of the benefit that it can be managed from a Java program (e.g. creating variables, reading solutions, ...) and therefore not requiring to implement a communication. In contrast to the general algorithm, the implementation expects the input orthogonal shape to be drawn as an orthogonal grid drawing. This however has the benefit, that the orthogonal shape can easily be stored in the GRAPHML format[4].

As test sets we referred to two benchmark sets commonly used in graph drawing: the ROME GRAPHS and the NORTH GRAPHS (also known as AT&T GRAPHS)[5]. Unfortunately, not all of the graphs in those test sets have maximum degree 4 or less. In order to make all graphs respect this constraint, for graphs with higher maximum degree sequentially edges from nodes with degree higher than 4 were removed until no node had degree larger than four[6]. If the graph was disconnected in the process, we split it into its connected components. Components which did not contain any edges where removed.

[1]Available at `https://java.com/download/`.

[2]Available at `https://www.yworks.com/products/yfiles`.

[3]Available at `http://www.gurobi.com/products/gurobi-optimizer`. For academic users as of now there are free academic licenses.

[4]For more information on the GraphML format refer to `http://graphml.graphdrawing.org/`.

[5]Both of which can be downloaded from `http://graphdrawing.org/data.html`.

[6]For this purpose once the graph was loaded by yFiles, a *NodeCursor* object was created with the *Graph.nodes()* method and used for iterating over nodes of the graph. For details on the used object and methods, refer to `http://docs.yworks.com/yfiles/doc/api/`.

Table 5.1: Number of graphs drawn successfully by the two approaches.

Test Set	Graphs in Test Set	Graphs drawn with $n_{staircase} = 1$	Graphs drawn with $n_{staircase} = 2$
Rome Graphs	16134	9864 (61.14 %)	9878 (61.22 %)
North Graphs	2865	2414 (84.26 %)	2415 (84.29 %)

Graphs were then embedded with the *PlanarInformation* class of yFiles and an orthogonal shape was computed with an implementation of TAMASSIA's min-flow algorithm[7]. The edge lengths were then fixed to minimal lengths with an ILP approach[8].

All of the 18999 graphs (given as orthogonal drawings) in the test sets (16134 graphs or components for the Rome graphs and 2865 for the North graphs) created by this procedure were drawn with both $n_{staircase} = 1$ and $n_{staircase} = 2$ (in both cases $M = 1000$). Already for $n_{staircase} = 3$ it could be observed that the linear program solver would need more than 20 hours for the ILP created by the staircase approximation for some graphs. Therefore we did not complete the test for $n_{staircase} = 3$ as the approach did not seem to be practical anymore.

All experiments where executed on a 64 bit UBUNTU 14.04 machine with $15.6GiB$ RAM and 4 INTEL® CORE™ I5-4590 processors running at $3.30GHz$. Time was measured with Java's *java.lang.System.currentTimeMillis()* method[9].

5.2 Experimental Results

For both test sets, both approaches ($n_{staircase} = 1$, i.e. the simple approximation, and $n_{staircase} = 2$) managed to draw more than 60 % of the given orthogonal shapes (cf. Table 5.1) as SC_1 drawings. In both test sets, there were a few orthogonal shapes (14 and 1, respectively) that could only be drawn with the ILP using $n_{staircase} = 2$. The graphs only drawn with the more precise approach were among the graphs that where drawn with largest total edge length and also were rather dense ($|E| \approx 1.5|V|$; cf. Table 5.2).

However, there are also qualitative differences between the two approaches (cf. Figure 5.1). Further we can observe differences in the performance on

[7]The min-flow network implementation was kindly provided by ROBERT KRUG who developed it during his work on Slanted Orthogonal Drawings [BKK+14]. Only the creation of the orthogonal shape based on the result of the min-flow network was missing.

[8]Edge lengths can also be computed in polynomial time with a flow network approach however code written for the Smog ILP could be reused.

[9]Refer to `https://docs.oracle.com/javase/7/docs/api/java/lang/System.html` for the documentation for this method.

Table 5.2: Statistics of graphs only drawn with $n_{staircase} = 2$.

Graph Name	Test Set	Total Edge Length	$\|V\|$	$\|E\|$	#(bent edges)
g.27.16	North	286.2345024704	32	47	10
grafo10151.94.0	Rome	3580.80075250733	132	202	25
grafo10557.98.0	Rome	2076.75212995925	113	153	14
grafo10588.93.0	Rome	7191.35375555133	112	154	17
grafo10634.96.0	Rome	2153.83179278309	114	155	17
grafo10868.96	Rome	1574.21676947629	116	148	17
grafo10954.94.0	Rome	3211.33177771091	102	133	9
grafo10993.95.0	Rome	5657.55288246658	131	194	31
grafo11350.91	Rome	2578.53090462617	109	150	17
grafo4401.57.0	Rome	665.438024953723	60	78	14
grafo8619.84.0	Rome	4931.53087448181	103	146	14
grafo9118.65	Rome	1127.5530633327	73	94	15
grafo9252.84	Rome	5873.27412287184	114	168	23
grafo9633.64.0	Rome	1667.8450699204	80	112	14
grafo9701.84.0	Rome	1014.7079331236	95	123	12

the two test sets: In the north test set, $n_{staircase} = 1$ often sufficed to draw the graph optimal, therefore the scatter plot almost forms a straight line with slope 1 (the linear regression function of LibreOffice Calc[10] computed a slope of 0.956). On the other hand, in the Rome graphs test set clearly the ILP with $n_{staircase} = 2$ often performed better - although there are still quite a lot of data points where both approaches performed equally well. This can be attributed to the following:

- Since we split the graph into its disconnected components after we disconnected it by removing edges to obtain a feasible maximum degree, quite a lot of very simple graphs were created that can be drawn orthogonally without any bends.
- There are graphs that can be drawn as an OC_1 drawing in the original test set.
- There are also quite a lot of graphs where bend minimal orthogonal shapes do not contain arcs which cannot intersect with other parts of the face they are contained in, i.e. they can be drawn as small as possible.

Since also in the further analysis the North graphs proved rather uninteresting (only slight differences between the two approaches. As both could draw many of the rather simple graphs equally well), we shall focus only on the Rome graphs from now on.

[10]Available at `https://www.libreoffice.org/`.

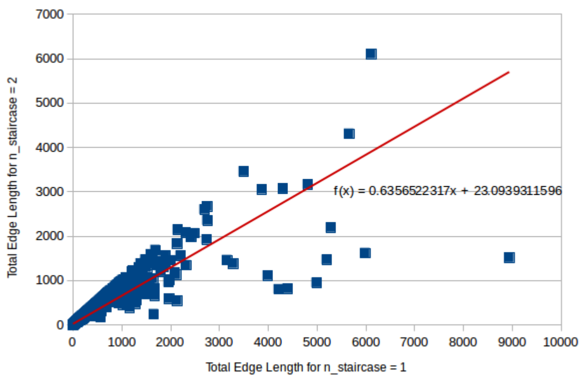
Of course we would also like to know which parameters of an input shape define how small we can draw an SC_1 drawing and how long we will need for computing the drawing. We will observe the effects of the following three parameters of the input shape:

- The *number of vertices* $|V|$ is an obvious choice, as for most algorithms the input size defines its runtime. On the other hand, we are using an ILP approach and it is often hard to track down what really makes an ILP hard to solve.
- The *density of the graph* measured by $\frac{|E|}{|V|}$ is another natural choice for our approach. A higher density can lead to more bends and therefore to more strict restrictions for the position of vertices w.r.t. the position of other vertices. Also, it may increase the likelihood that an arc is not empty but other edges of the face are reaching inside the arc and therefore requiring us to draw the arc big enough to not cut other edges.
- The *bent edge ratio* measured by $\frac{\#(arcs)}{|E|}$ might also be important since more bends result in more severe restrictions. However, it should be noted that also very easy drawings can achieve a high bent edge ratio, e.g. the graph in Figure 5.2.

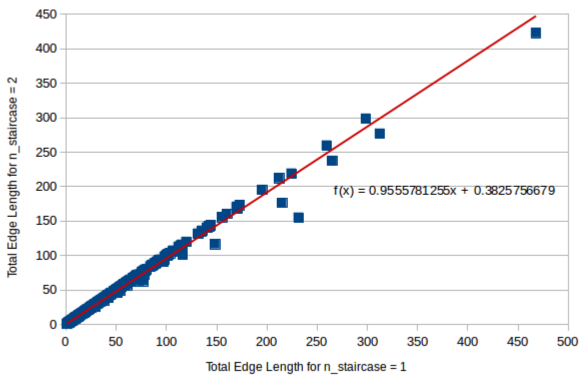
In order to argue about the quality of the two parameter settings ($n_{staircase} = 1$ and $n_{staircase} = 2$), we will use the following measurements averaged over values of the input parameters:

- The *Average Total Edge Length* (for short ATEL) is directly computed by our ILP as the objective function. We will average over a given input parameter value (i.e. number of vertices, density, bent edge ratio).
- The *Average Computation Time* (for short ACT) is the average time our approach needed for the computation for a given input parameter value.
- The *Average of Average Edge Length* (for short AAEL) is the average of the average edge lengths in the computed drawing for each graph that has the given input parameter value.

When analyzing depending on the number of vertices $|V|$, we can observe the results shown in Figure 5.3. The total edge length as well as the average edge length increase with the number of vertices in a parabolic fashion. This is not surprising since we have seen before that in the worst case our drawing size can increase exponentially in $|V|$. On the other hand, when investigating the time needed, we can observe the biggest spikes not at the largest numbers of vertices. Also, when using a logarithmic scale for the ACT (cf. Figure 5.3d), we can see that besides that spikes occur more often with an increasing number of



(a) Scatter plot for Rome graphs.



(b) Scatter plot for North graphs.

Figure 5.1: Scatter plots with linear regression (red) for the performance of both approaches for both test sets. Each data point (blue squares) resembles a graph drawn by both approaches.

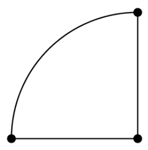
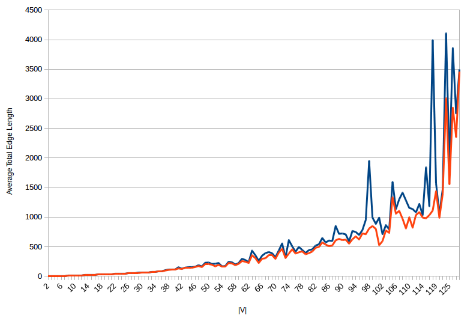
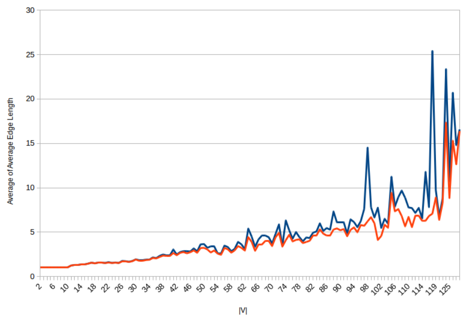


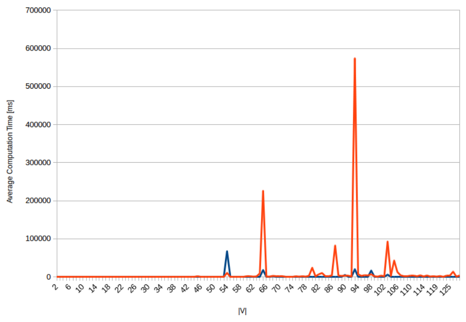
Figure 5.2: An easy to draw graph with rather high bent edge ratio of value $\frac{1}{3}$.



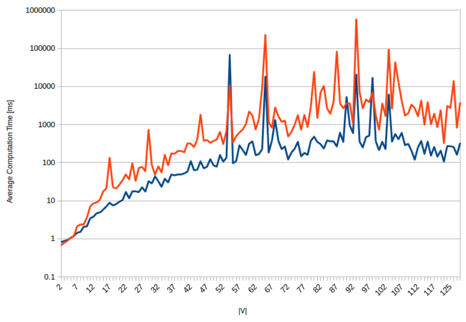
(a) ATEL.



(b) AAEL.



(c) ACT.

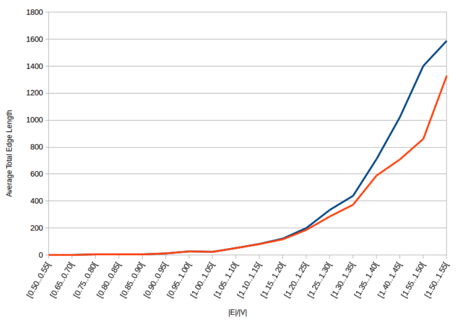


(d) ACT (in logarithmic scale).

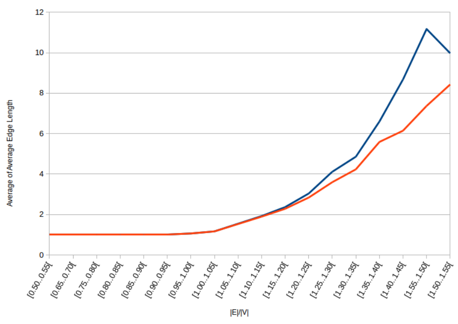
Figure 5.3: All three measurements dependent on the input size of the graph $|V|$ (Figures (c) and (d) both show the ACT where a logarithmic scale is used in (d)). Results for $n_{staircase} = 1$ are drawn in blue, results for $n_{staircase} = 2$ are drawn in red.

vertices, the base line of needed time rises rather slowly in $|V|$. Also, $n_{staircase} = 2$ achieves better edge lengths for increasing $|V|$ whereas also the runtime increases faster than for $n_{staircase} = 1$.

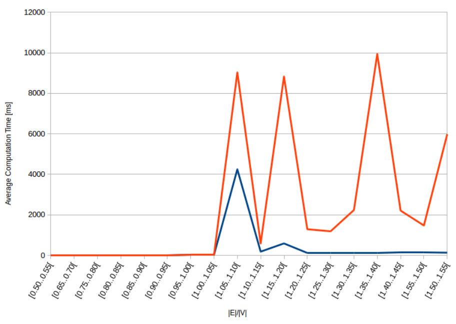
When analyzing for the density of the graphs, we can observe less distorted results (cf. Figure 5.4). Both ATEL and AAEL increase smoothly with the density (the drop-off at the last value for AAEL may be attributed to the fact that the sample size for this value is 1). However, $n_{staircase} = 2$ performs even better w.r.t. AAEL than w.r.t. ATEL compared to $n_{staircase} = 1$. Also, we can observe that the time needed increases with density. We can observe, that the large spikes start to occur when the density starts to be greater than 1 (i.e. if the graph is not a tree) and do so for $n_{staircase} = 2$ for the rest of our density samples. On the other hand, again we can see clear spikes: Not all ILPs for large densities require much time to compute, other can still be solved relatively fast. When we analyze the ACT in logarithmic scale, we can observe that the baseline of time needed hugely increases between $|E|/|V| = 0.9$ and $|E|/|V| = 1.1$ but then more or less stays the same. Also note that there seems to be large distortion in the interval between $|E|/|V| = 1.05$ and $|E|/|V| = 1.1$



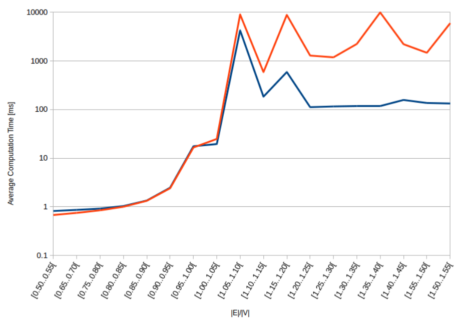
(a) ATEL.



(b) AAEL.



(c) ACT.



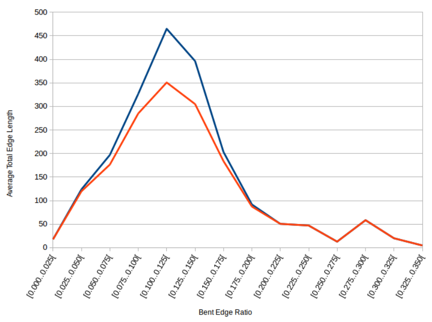
(d) ACT (in logarithmic scale).

Figure 5.4: All three measurements dependent on the density of the graph $\frac{|E|}{|V|}$ (Figures (c) and (d) both show the ACT where a logarithmic scale is used in (d)). Results for $n_{staircase} = 1$ are drawn in blue, results for $n_{staircase} = 2$ are drawn in red.

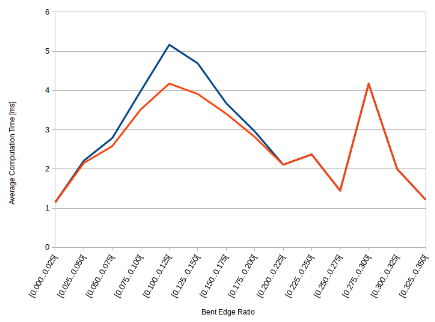
caused by the presence of spikes destroying the continuity of the curve.

For the bent edge ratio unfortunately we cannot observe a clear pattern, as it seems to be greatly influenced by the effect of easy drawings with high bent edge ratios (cf. Figure 5.5). The approach with $n_{staircase} = 2$ again needs more time for computation for literally all bent edge ratios but also performs significantly better for bent edge ratios in the interval $[0.05..0.2]$ (which might be the interval, where easy to draw graphs are not so often present). While this still seems to indicate that the bent edge ratio has an effect on performance, we will not analyze this parameter further as no clear pattern emerged. Also note that the decline in ACT with increasing bent edge ratio as observable in Figure 5.5d might be caused by the presence of those easy drawings with high bent edge ratios.

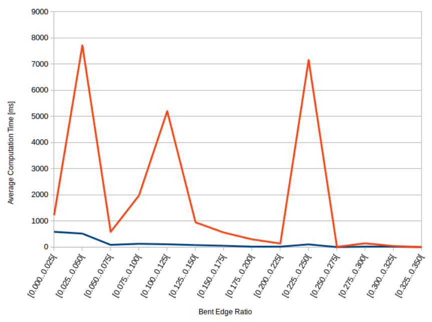
Spikes in computation time reached maximum values of $2,959,604ms$ $(0.822h)$ for $n_{staircase} = 1$ respectively $6,968,163ms$ $(1.936h)$ for $n_{staircase} = 2$. This result is not surprising as ILP solvers are known for having exponential worst case run time and just shows that the ILP for finding a SC_1 drawing can be hard depending on the input orthogonal shape. Interestingly, those spikes



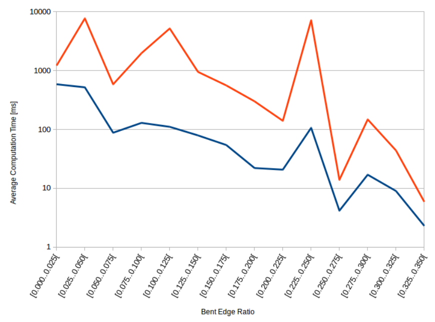
(a) ATEL.



(b) AAEL.

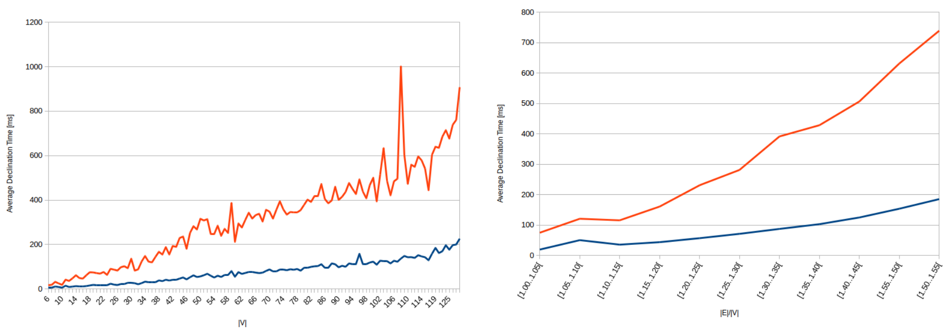


(c) ACT.



(d) ACT (in logarithmic scale).

Figure 5.5: All three measurements dependent on the bent edge ratio of the graph (Figures (c) and (d) both show the ACT where a logarithmic scale is used in (d)). Results for $n_{staircase} = 1$ are drawn in blue, results for $n_{staircase} = 2$ are drawn in red.



(a) Dependent on $|V|$. **(b)** Dependent on $\frac{|E|}{|V|}$.

Figure 5.6: ADT dependent on graph size and density. Results for $n_{staircase} = 1$ are drawn in blue, results for $n_{staircase} = 2$ are drawn in red.

occurred when drawing different graphs. Nevertheless, for the two graphs with largest spikes, also the other approach needed much time for computing a solution. At its largest spike, $n_{staircase} = 1$ needed more time than $n_{staircase} = 2$ in the experimental run, however this result could not be replicated. In particular, when trying to replicate it, the ILP with $n_{staircase} = 2$ needed far longer to be solved indicating that in the experimental run, the ILP solver might have branched differently and therefore terminated faster.

To conclude our discussion of the experimental results, we can also consider how long it takes our ILP approach to detect when a graph cannot be drawn as a SC_1 drawing depending on $n_{staircase}$, i.e. we can measure the *Average Declination Time* (ADT) dependent on either the number of vertices or the density of the graph (cf. Figure 5.6). Again we can observe that the density plot is by far smoother than the plot depending on the number of vertices. Further, we can observe that already for $|V| = 6$ (cf. Figure 5.7) there were graphs that could not be drawn with SC_1 whereas only for densities $\frac{|E|}{|V|} \geq 1$ there were graphs which had this property. The average declination time increases both with graph size and graph density for the following reasons: First of all, there are more constraints that need to be checked for their feasibility. Second, the more variables there are in an ILP (especially the boolean indicator variables for intersections) the more likely it is that after a given number of branches the LP-relaxation is still feasible. In particular the LP-relaxation might not become infeasible until all indicator variables for the same edge pair have been branched (i.e. they can only take integer values).

Since we got the smoothest dependencies between our measurements and the density of the graph while measurements dependent on the graph size yielded similar yet more distorted results, it may also be worth analyzing properties of the test set. It turns out that there is a significant correlation between the number of vertices and the density of the graphs in the Rome

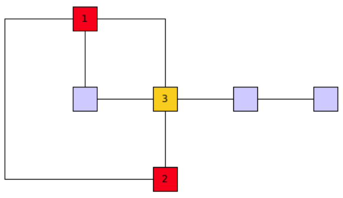


Figure 5.7: An orthogonal shape of a graph with just 6 vertices that cannot be realized as a SC_1 drawing. Because of the edge with 2 bends (which would be drawn as a half circle arc), both red vertices must have the same horizontal coordinate. However, both are connected to the yellow vertex 3. Since the red vertex 1 is connected with a quarter circle arc to 3, it cannot have the same horizontal coordinate. On the other hand, 2 is connected with a straight line to 3 and must have the same horizontal coordinate.

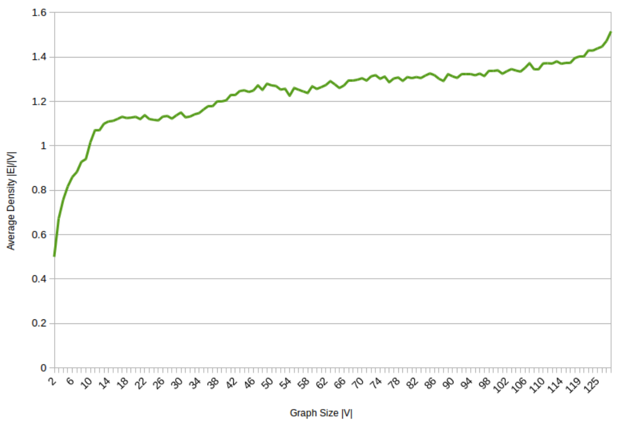


Figure 5.8: Average density of the graphs in the test set depending on the number of vertices. The large slope for small numbers of vertices can be partially explained by the fact that many of the small graphs were created in our preprocessing where we disconnected larger graphs by removing edges.

graphs test set (cf. Figure 5.8). Therefore it may be the case that the total edge lengths computed by both parameter settings differ for larger graph sizes only significantly due to the fact that the test set's average density increases with the graph size. Judging from our results and this bias in the test set, it seems the only parameter we can be really confident about influencing the difference in performance of both algorithms, is the density of the graph.

Chapter 6

Discussion

6.1 Discussion of Our Results

Our practical evaluation suggests that our polynomial time reduction with a fixed $n_{staircase}$ from drawing an orthogonal shape with SC_1 to an ILP can be applied efficiently in a practical context. In particular, even for $n_{staircase} = 1$, which is identical to the simple arc approximation, most of the graphs in the benchmark test sets could already be drawn. The benchmark test of the more sophisticated staircase approximation with $n_{staircase} = 2$ showed, that only 15 more orthogonal shapes could be drawn out of the 6,721 test shapes which could not be drawn with the simple arc approximation[1]. This indicates that cases which require more precise arc approximations could be very rare in practical settings making the simple approximation ILP a viable choice in most cases.

Nevertheless, especially for graphs of larger size and density, the staircase approximation with $n_{staircase} = 2$ could improve the total edge length required for the computed drawing. The simple approximation always reserves square-shaped space for its contents, if the content is not square shaped then there can be much unused area. Further, the inner connector edge lengths are large in the simple approximation and can cause edges in the output drawing to also have the same length. Those patterns are easy to observe. Therefore, if an arc has been drawn too large by the simple approximation we can apply the staircase approximation to achieve better results. Also, if we should encounter a graph which cannot be drawn with the simple approximation we can still use the staircase approximation with $n_{staircase} = 2$ to possibly find a solution.

Several of the orthogonal shapes rejected by the staircase approximation in the practical evaluation were manually investigated. For all the the observed shapes we could explain why they cannot be drawn as a perfect smooth or-

[1]In total, 18,999 were evaluated.

thogonal drawing. Further we identified local configurations in those shapes that do not admit for a perfect smooth orthogonal drawing. These forbidden local configurations will be discussed in Section 6.5. While we did not check every rejected orthogonal shape due to the fact that it is not yet known which properties an orthogonal shape has to fulfil in order to be realizable as an SC_1 drawing. Therefore we could not check them automatically. It still suggests that $n_{staircase} = 2$ might be enough.

However despite the great performance on benchmark test sets, there is no guarantee for a fixed $n_{staircase}$ that our approach will find a drawing even if one exists. In particular, judging by our current knowledge, we need to assume that there are orthogonal shapes which only admit for one special family of perfect smooth orthogonal drawings which is characterized as follows: Each drawing in the family can be obtained from any other drawing of the family by only scaling all edge lengths and radii by the same factor. If we could prove that there is always more freedom in the choices of edge lengths we might be able to guarantee a drawing for $n_{staircase} = 2$.

6.2 Comments on the Runtime

The presented approach shows runtime behaviour typical for an ILP. In Chapter 5 we could observe an increase in average computation time along with the size and density of the graph (which were two properties of the benchmark that correlated), i.e. with an increasing complexity of the created ILP. While the baseline of average computation time retained good values, spikes of large magnitude could also be observed for denser and larger graphs.

When using ILP it is always important to remember that it is a $\mathcal{NP}$-hard problem. Therefore we need to expect that there exist difficult instances which cannot be solved in polynomial time. In particular, ILP solvers use the Branch & Bound algorithm (as introduced in Chapter 2) among other techniques. Branch & Bound does not solve integer solutions directly but operates on LP-Relaxations of the problem which are created by branching variables. Branching means that additional constraints are imposed on variables that are not assigned integer values in the solution of the previous LP-Relaxation creating 2 additional LP-Relaxations that need to be solved. In principle every integer decision variable can be branched multiple times until the ILP solver only has LP-Relaxations left that have worse objective function values than the best known integer solution bound. Thus exponentially many subproblems can be created before the final solution is known.

The runtime is mostly defined by the time the ILP solver needs for computation, replacing the arcs and creating the ILP can be done in quadratic time (we replace arcs by a constant number of edges and we insert constraints for edge pairs). The fact that we created hard ILP instances with our approach indicates that drawing an orthogonal shape with SC_1 could in fact be a hard problem but for now the complexity of this problem remains unknown.

6.3 Nearly Perfect Smooth Orthogonal Drawings

In this section we will discuss a concept on how to resolve a pattern that usually emerged in SC_1 drawings that were amongst the graphs with largest total edge length in our evaluation (cf. Figure 6.1[2]):

1. A rather dense part of the graph can still be drawn effectively with SC_1 using rather few space (blue in the figure).
2. An arc's radius has to be equal to the height or width of the dense part (green). The arc then needs as much space as the dense blue part of the graph.
3. The pattern repeats: Another arc's radius (red) has to be larger than the radius of the previously drawn radius (green).

In very large drawings, this pattern will result in the worst case exponential area requirement as proven by Bekos et al. [BKKS13].

Those issues will are known to arise when using an SC_1 drawing algorithm[3] and are not just a flaw of the approaches discussed in this thesis. A possible solution may be to draw not SC_1 drawings but *Nearly Perfect Smooth Orthogonal Drawings*:

Definition 22 (Nearly Perfect Smooth Orthogonal Drawing)**.** A *k-nearly perfect smooth orthogonal drawing* is a smooth orthogonal drawing of a graph $G = (V, E)$, where k edges $e \in E$ are drawn with 2 edge segments where one of those segments is an arc and the other is a rectilinear line segment whereas all the other edges are drawn with a single segment.

The idea here is to allow some of the critical bent edges to consist of a rectilinear line segment and an arc. In particular, it may be a good idea to either replace the arcs with largest radii (red in Figure 6.1) or the arcs that are connected to dense areas and therefore cannot be drawn with small radius and subsequently contribute to the largest radii (green in the figure).

[2]For better readability, a rather small drawing which already contained the pattern was selected to be presented.

[3]At least as long as the input is an orthogonal shape, a different shape might achieve better results.

Table 6.1: Total edge lengths computed depending on $n_{staircase}$ and the replacement of arcs for the first example graph.

$n_{staircase}$	Original Graph (cf. Figure 6.1)	Green arc replaced (cf. Figure 6.2)	Red arc replaced (cf. Figure 6.3)	Red arc replaced (cf. Figure 6.4)
1	186.9	149.7	163.4	177.4
2	171.4	140.6	162.0	173.4

In order to test which of the two arcs would be the better candidate for replacement, the orthogonal shape was edited with yEd[4] such that exactly one of the arcs would contain a bend (yellow circle in the following Figures 6.2 to 6.4), i.e. we would obtain a 1-nearly perfect smooth orthogonal drawing. For the green arc it is obvious at which endpoint the rectilinear line segment should be located (cf. Figure 6.2), for the red arc both options (cf. Figures 6.3 and 6.4) were tested. Note that removing the edge and redrawing it after calculating the remaining edge lengths will not work in general since then there is no space reserved for the missing edge and we may be unable to draw it (cf. Figure 6.5).

For the three different described replacements for nearly perfect smooth orthogonal drawings on this graph, we could improve with both $n_{staircase} = 1$ and $n_{staircase} = 2$ w.r.t. total edge length except for one case (cf. Table 6.1). For this particular graph it seems generally better to replace the green arc if we allow only to increase the complexity of a single edge. We can even observe, that we might increase the total edge length by replacing the wrong arcs (cf. Figure 6.4: We cannot draw the arc smaller by inserting the rectilinear line segment compared to the drawing in Figure 6.1).

However, replacing the first arc that needs to be drawn somewhat large and then forces other arcs to be drawn larger does not always work. For instance, consider the graph drawn in Figure 6.6: If we replace one of the smaller arcs in the middle of the drawing, we can reduce the total edge length significantly (and by doing so we end up with the drawing in Figure 6.7). However, if we replace one of the larger arcs, we get even less total edge length (cf. Figure 6.8). Finally, to make it even more counter-intuitive, the best drawing achieved even is perfect smooth orthogonal (computed with $n_{staircase} = 2$; cf. Figure 6.9), the two nearly perfect smooth orthogonal options are both worse (cf. Table 6.2).

In conclusion, it seems that nearly perfect smooth orthogonal drawings can improve the total edge length significantly while still maintaining similar aesthetic criteria but only if applied correctly. It seems that the decision on the arc to replace with a smaller arc and a rectilinear line segment to achieve

[4]Available at `https://www.yworks.com/products/yed`.

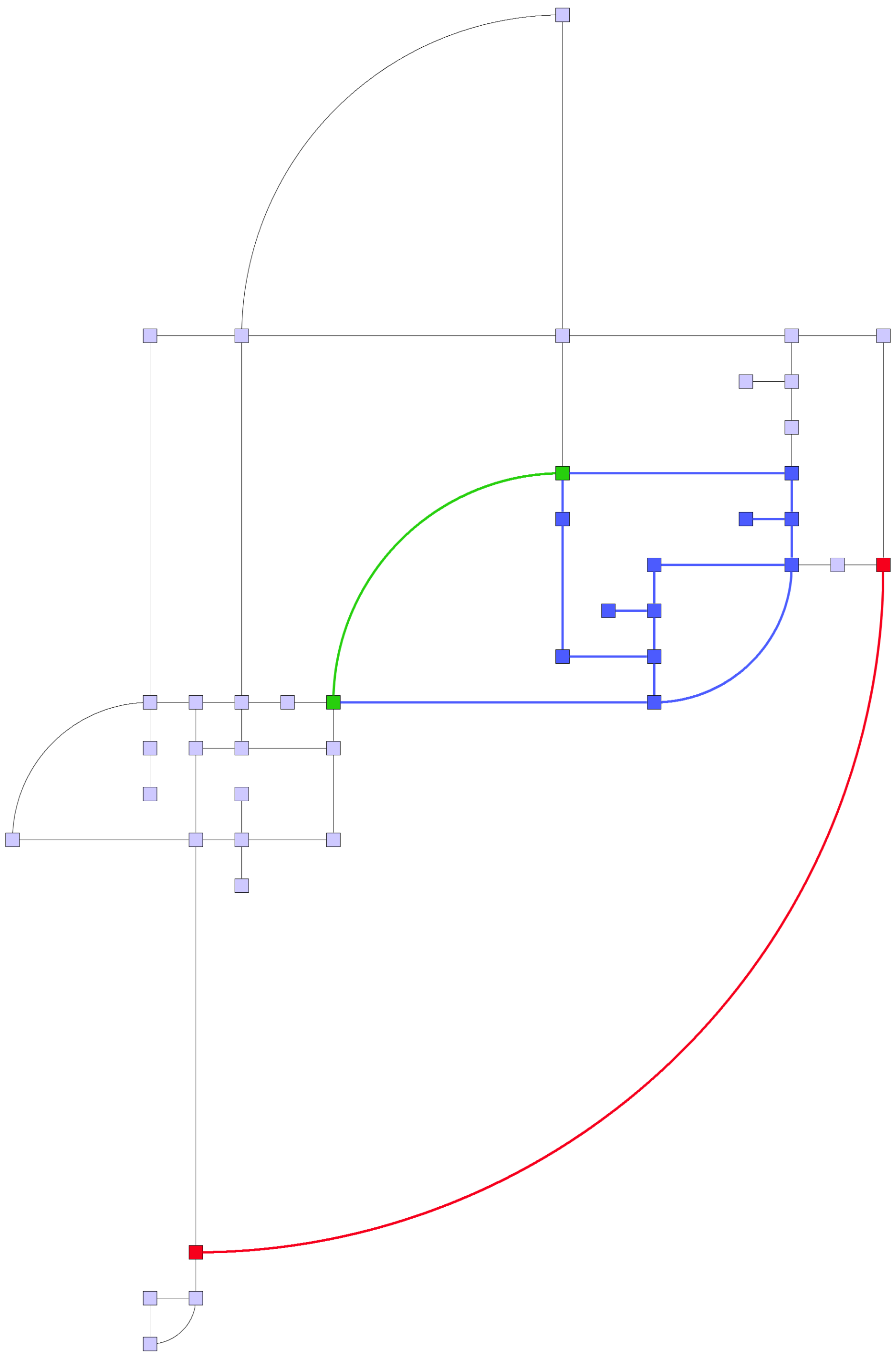


Figure 6.1: Graph with a pattern typical for very large SC_1 drawings (drawn with $n_{staircase} = 2$). The drawing contains a dense area (blue) which causes the green arc to be drawn with a large radius. Subsequently, the red arc's radius has to be even larger. These arcs are candidates for being drawn with two segments in a nearly perfect smooth orthogonal drawing.

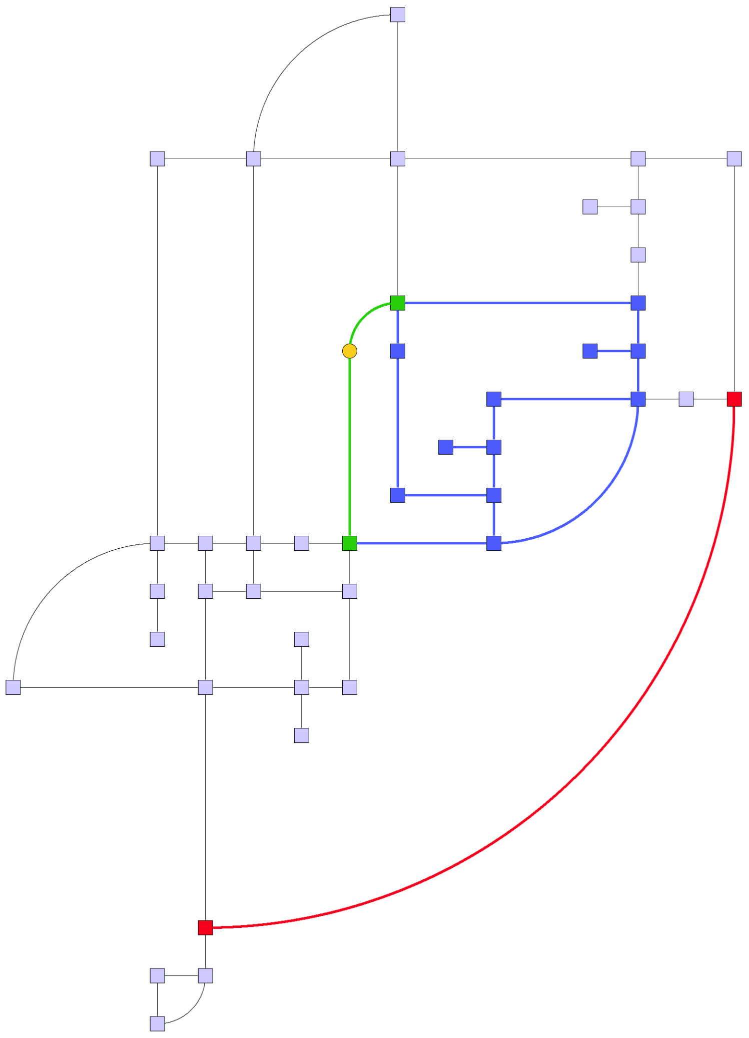


Figure 6.2: The orthogonal shape from Figure 6.1 where the green arc is drawn with 2 segments (drawn with $n_{staircase} = 2$) separated by the yellow vertex. The total edge length as well as both width and height of the drawing are significantly reduced.

Table 6.2: Total edge lengths computed depending on $n_{staircase}$ and the replacement of arcs for the second example graph.

$n_{staircase}$	Original Graph (cf. Figure 6.6)	Inner arc replaced (cf. Figure 6.7)	Outer arc replaced (cf. Figure 6.8)
1	564.1	224.3	175.6
2	164.8	179.1	168.6

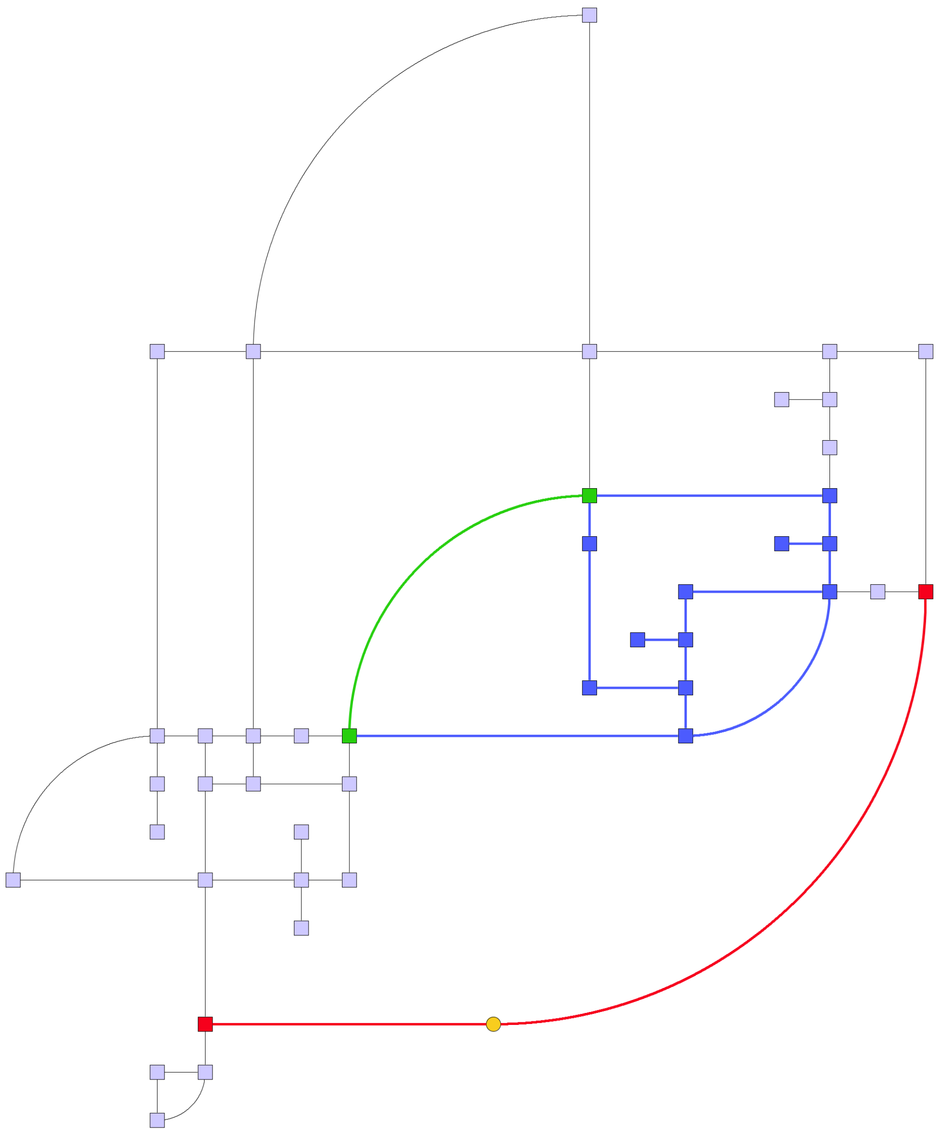


Figure 6.3: The orthogonal shape from Figure 6.1 where the red arc is drawn with 2 segments (drawn with $n_{staircase} = 2$) separated by the yellow vertex. Again the total edge length is reduced compared to the perfect smooth orthogonal drawing, however this time only the height of the drawing could be reduced.

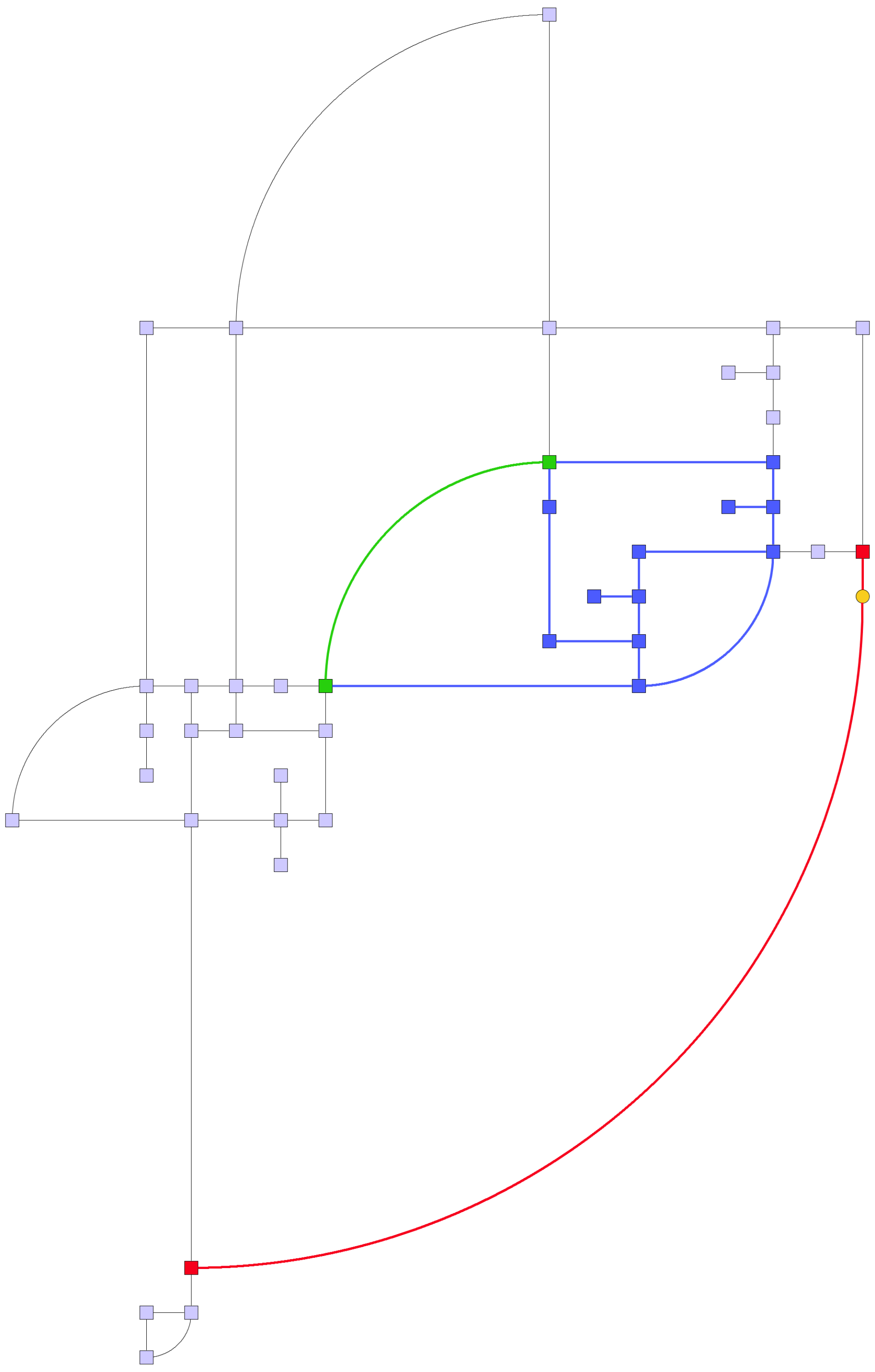


Figure 6.4: The orthogonal shape from Figure 6.1 where the red arc is drawn with 2 segments (drawn with $n_{staircase} = 2$) separated by the yellow vertex. This time the straight-line segment is located at the other endpoint which results in no benefit over the perfect smooth orthogonal drawing.

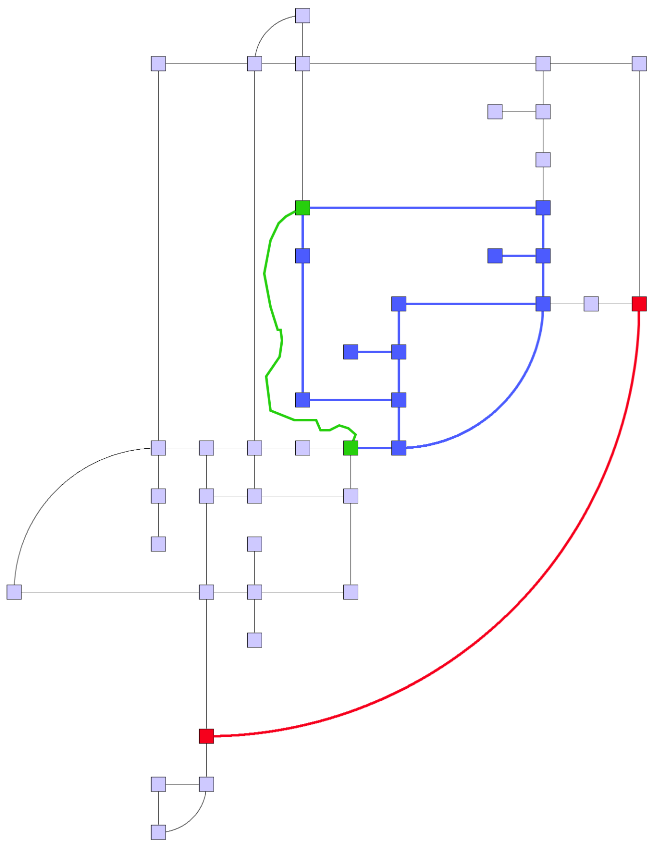


Figure 6.5: The orthogonal shape from Figure 6.1 where the green edge was removed before drawing (drawn with $n_{staircase} = 2$). After drawing there is no grid space left to draw the green edge.

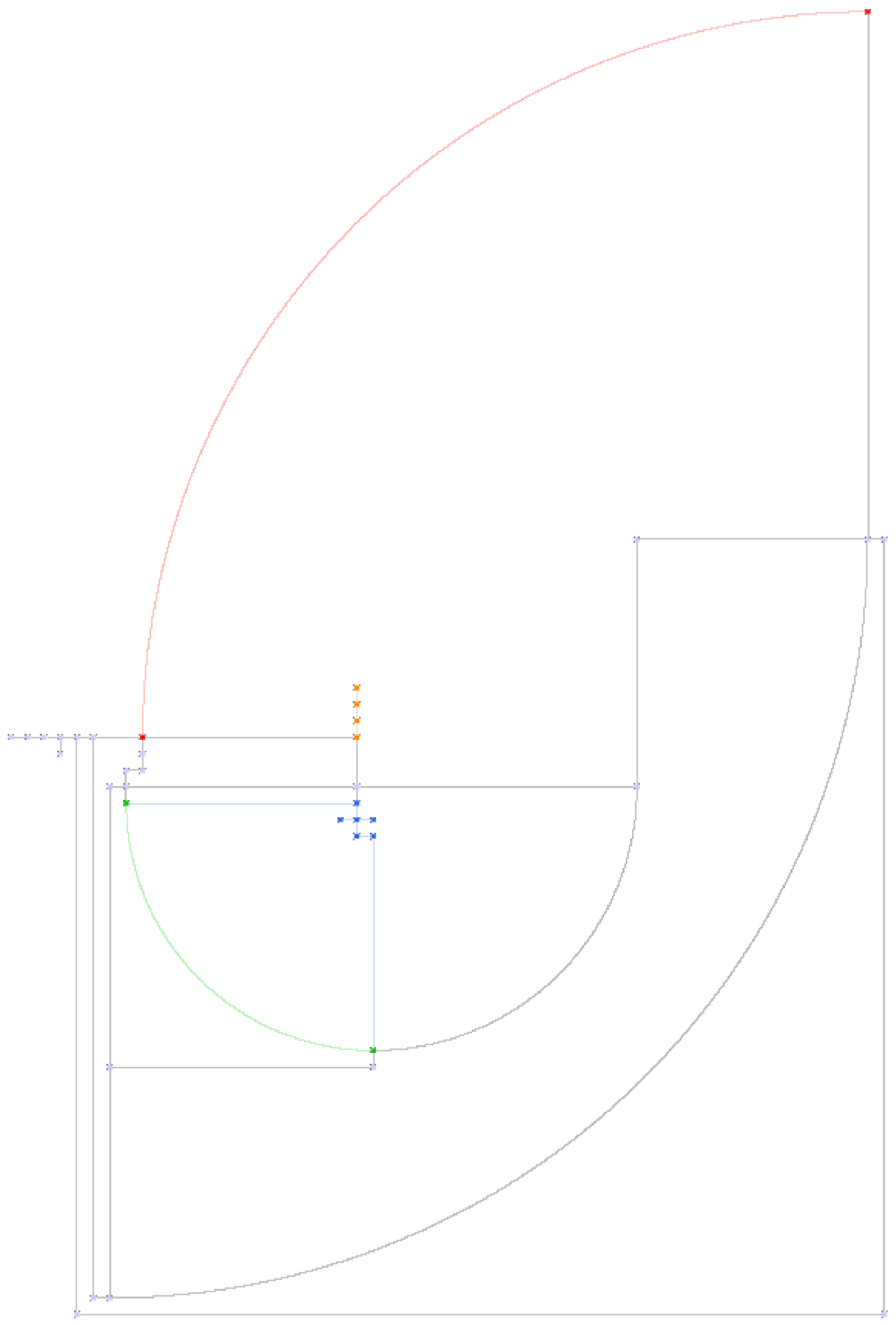


Figure 6.6: A second example graph showing the following pattern responsible for exponential drawing space (drawn with $n_{staircase} = 1$): The rather dense area of the graph (blue) requires the green arc to be drawn with a radius as large as the height of the blue area. However, the green arcs length is also influenced by the inner connector edge length of the red arc which is further restricted by the sequence of bridge edges (orange) pointing into the face. Simultaneously, the green edge also defines the radius of the other arcs in the drawing.

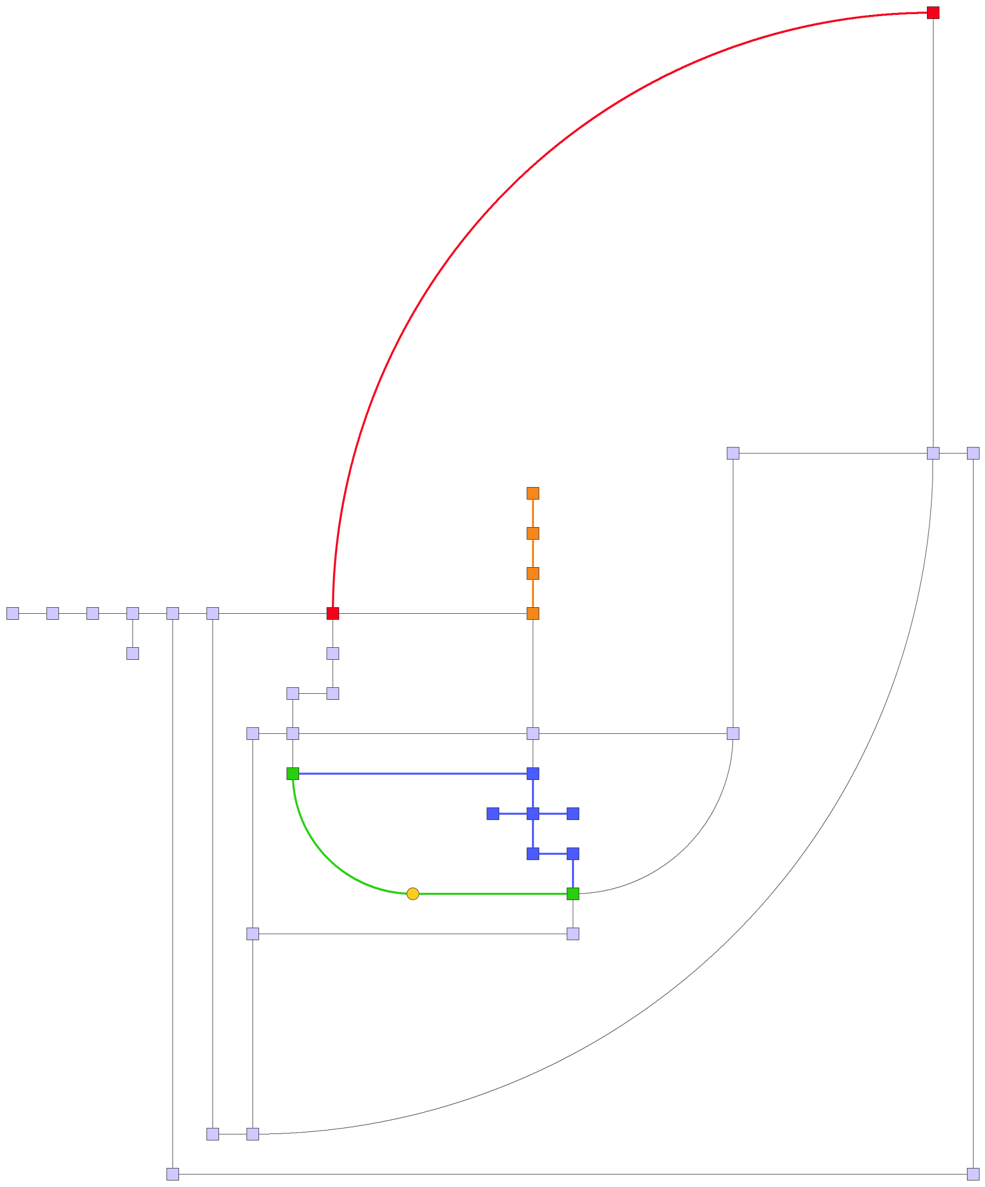


Figure 6.7: The orthogonal shape from Figure 6.6 where the inner green arc is drawn with 2 segments (drawn with $n_{staircase} = 1$) separated by the yellow vertex. By breaking the arc that previously defined the radii of the other arcs, a significant decrease in total edge length and area can be achieved.

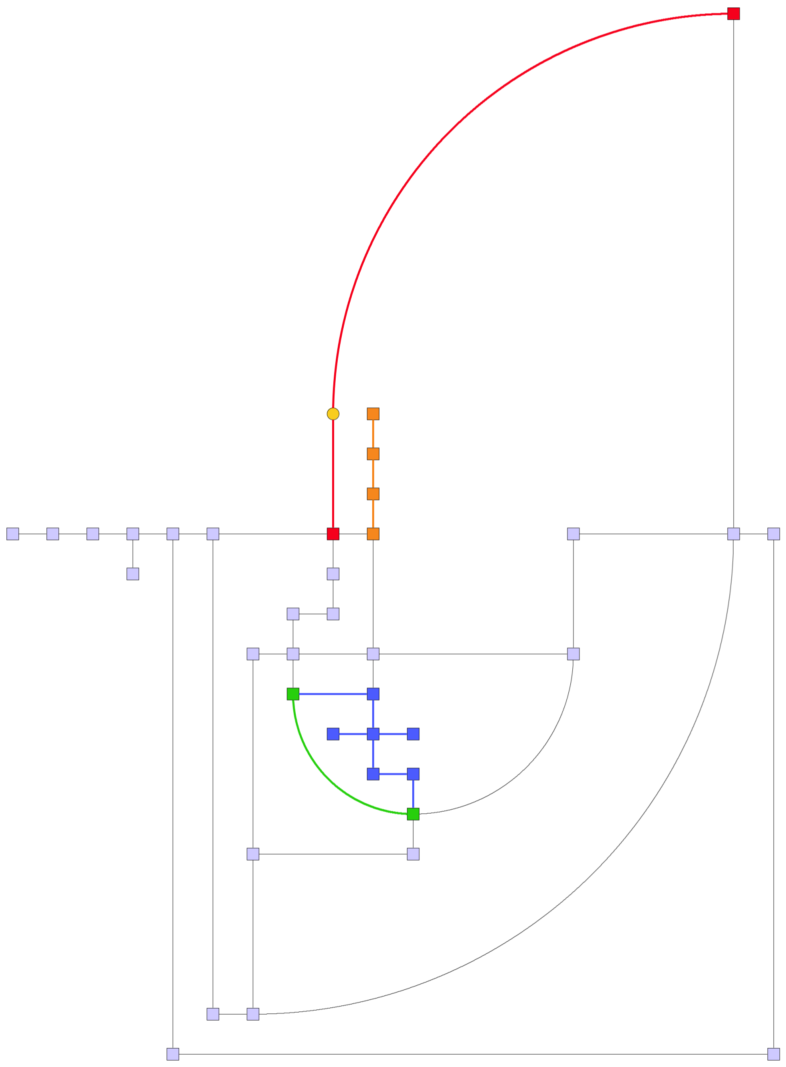


Figure 6.8: The orthogonal shape from Figure 6.6 where the outer red arc is drawn with 2 segments (drawn with $n_{staircase} = 1$) separated by the yellow vertex. For this orthogonal shape splitting the outer arc had a stronger effect than drawing the inner arc that defined the radii of the other arcs. This is contrasts our findings for the other example orthogonal shape (cf. Figure 6.1).

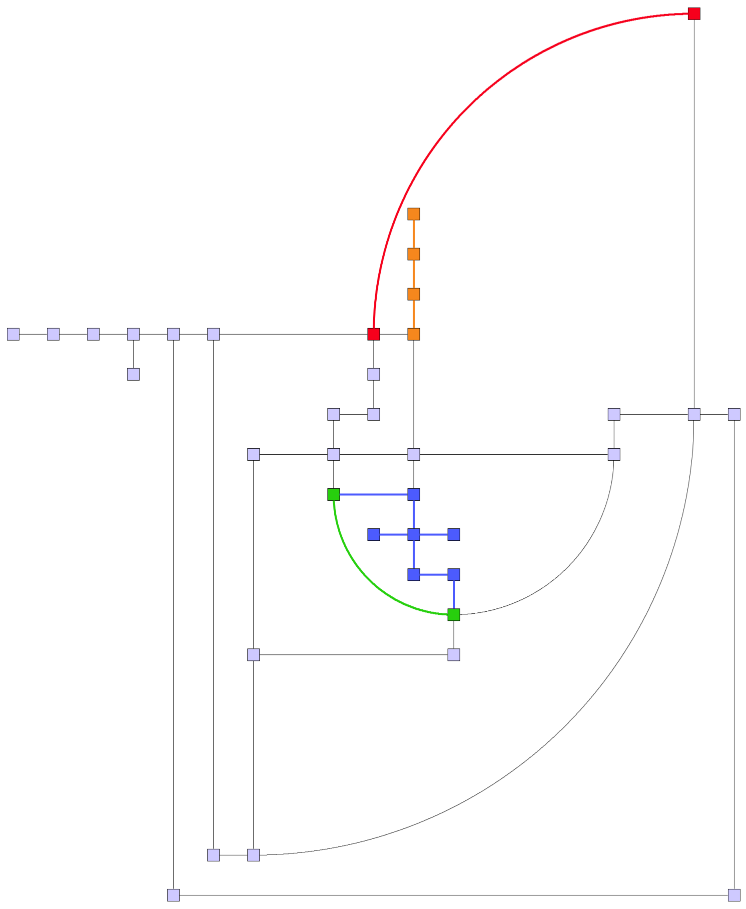


Figure 6.9: The orthogonal shape from Figure 6.6 drawn with $n_{staircase} = 2$ as a perfect smooth orthogonal drawing. Drawing arcs with two segments similar to Figure 6.7 and 6.8 yields a higher total edge length for $n_{staircase} = 2$ than the perfect smooth orthogonal drawing depicted here.

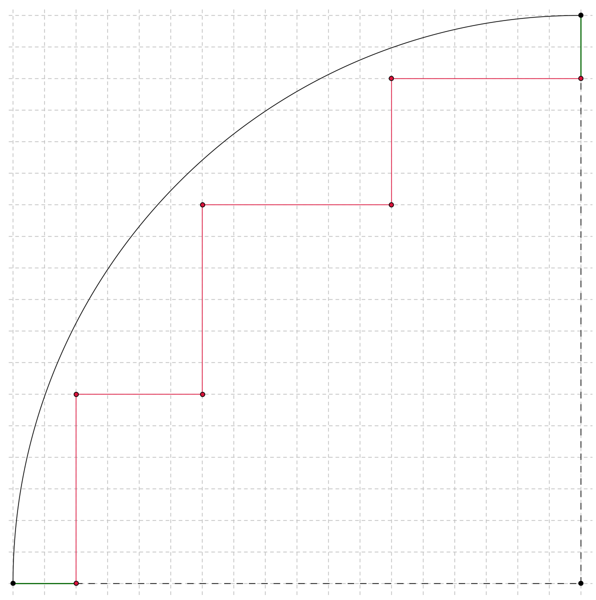


Figure 6.10: An arc of radius 18 and its inner approximation (red; first inner connector edge green) as defined by the staircase approximation.

the best improvement in total edge length is not trivial. Also we already could observe that it also matters at which endpoint of the replaced arc the rectilinear line segment is placed. Therefore the best strategy for nearly perfect smooth orthogonal drawings remains an open problem.

6.4 Varying the Staircase Step Size

We already discussed that we should not exceed $n_{staircase} = 2$ to avoid runtimes which are not applicable for most practical contexts. However, the staircase approximation was defined in Chapter 4 with an arbitrarily chosen definition of how large steps should be (cf Figure 6.10; dividing the inner approximation edges for $n_{staircase} = 1$ in $n_{staircase}$ equally long edges).

This definition of edge lengths was defined under the assumption that $n_{staircase}$ can reach values significantly larger than 2. If we fix the number of edges to the number of edges in the staircase approximation for $n_{staircase} = 2$ we could still vary edge lengths.

For instance, we could focus on minimizing the first inner connector edge's length (cf. Figure 6.11). This seems especially favorable since we first introduced the staircase approximation to deal with the fact that the simple approximation would draw the inner connector edges too long. This resulted in too large arcs that sometimes even prevented us from finding a feasible solution. However, minimizing the first inner connector edge length, i.e. requiring it to be 1, cannot be done easily with only linear constraints, if we wish to

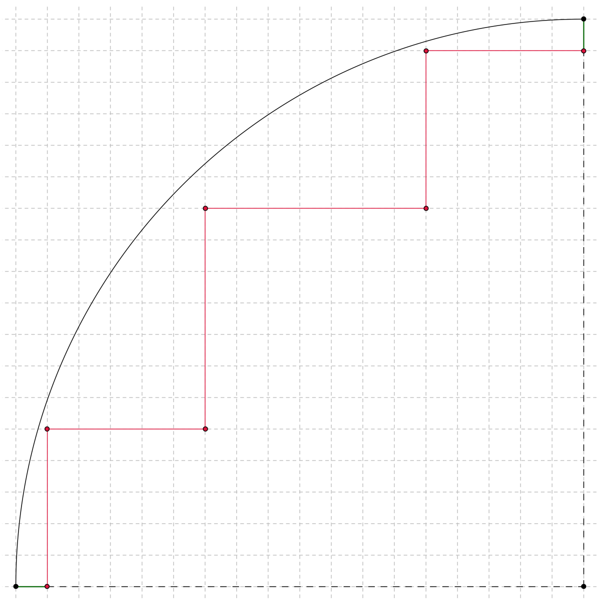


Figure 6.11: An arc of radius 18 and its inner approximation drawn such that the first inner connector edge (green) is as short as possible.

draw the first inner approximation edge connected to the connector edge as large as admitted by the arc. So far when we defined edge lengths we always have been lucky when applying PYTHAGORAS' theorem and came up with a linear equation. But if we would say that the first inner connector's edge length shall be 1 we could not do so[5]. Therefore, if we are restricted to linear constraints, we could not reserve as few space for arc drawing as shown in the figure if we do not know the radius in advance. This can lead to this type of approximation performing worse w.r.t. total edge length than our staircase approach.

Another alternative would be the minimization of space reserved for arc drawing (cf. Figure 6.12). But if we would use this approximation, we would again increase the length of the inner connector edge compared to the staircase approximation which might be undesired. Also, as with minimizing the connector edge length, it might be difficult to formulate this kind of approximation properly with linear constraints.

Finally, when comparing the figures to each other, it also appears that the staircase approximation is a good trade-off between connector edge length and space reserved for arc drawing. However, it remains to study the effects of alternating the edge lengths for the staircase approximation with $n_{staircase} = 2$ in-depth to confidently argue about which edge lengths perform best in

[5]In particular the right triangle we always used for computation would have the two known edge lengths R and $\frac{R-1}{R}$ where R is the radius of the arc - clearly the remaining edge length cannot be expressed linearly dependent on R

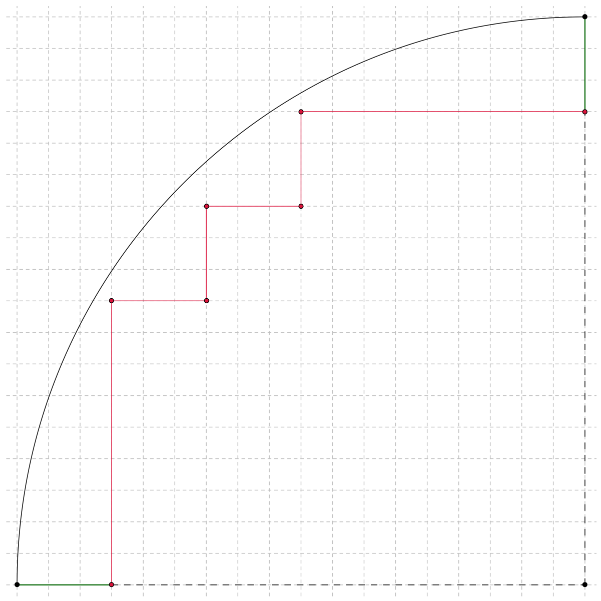


Figure 6.12: An arc of radius 18 and its inner approximation drawn such that the space reserved for drawing the arc is minimized.

practice.

6.5 First Insights for the Shape Step

In this thesis, an ILP was presented which is able to compute a perfect smooth orthogonal drawing for a given orthogonal shape as benchmark tests indicate. As described in Chapter 3 we would like to have a TSM framework for smooth orthogonal drawings. So far, we only discussed the metrics step of the TSM approach. However also algorithms that compute feasible embeddings and orthogonal shapes are needed.

The results of our practical evaluation can also give us first insights in the properties required in the shape step so we can guarantee that the computed shape can be drawn as a SC_1 drawing. Preferably we would also like a resulting drawing to use as few space as possible.

Bend minimization comparable to the bend minimization in Tamassia's min-flow algorithm for orthogonal shapes [Tam87] should also play a role for SC_1 drawings as arcs are responsible for the worst-case exponential space requirement. However, there is a significant difference. For SC_1 drawings, it does not really matter how many bends an edge in the orthogonal shape has as long as the number of bends is between 0 and 3. If we have more than 3 bends in the orthogonal shape, we cannot replace the bent edge with an arc (for 4 bends we would need to insert a full circle, but a full circle edge would have its 2 endpoints overlapping). Therefore, no edge in the orthogonal shape may have more than 3 bends. On the other hand, if we have between 1 and 3 bends along an edge in the orthogonal shape, it will be drawn as a single arc in the final drawing. Of course, a three-quarters circle arc will require more space to draw than a quarter circle arc. But if we can chose between a single three-quarters circle arc and 3 quarter circle arcs[6], it may turn out, that 3 quarter circle arcs are even worse since they can interact with each other and force every subsequent arc to be drawn larger than the one before. Thus, rather than minimizing the number of bends in the orthogonal shape it might be better to instead minimize the number of arcs in the final drawing. However, we do not know yet if in most cases one large arc or the 3 smaller arcs perform better in SC_1 drawings.

Further, the practical evaluation tells us which of the orthogonal shapes in our test set could not be drawn. This allows an analysis of those shapes for local configurations that prevent our ILP from drawing the graph with SC_1. First, let us discuss configurations that only involve a single arc (cf. Figure 6.13):

For half circle arcs we know that their endpoints need to have one common horizontal or vertical coordinate in the final drawing (in Figure 6.13a they need to have the same vertical coordinate). Therefore each path between the two endpoints

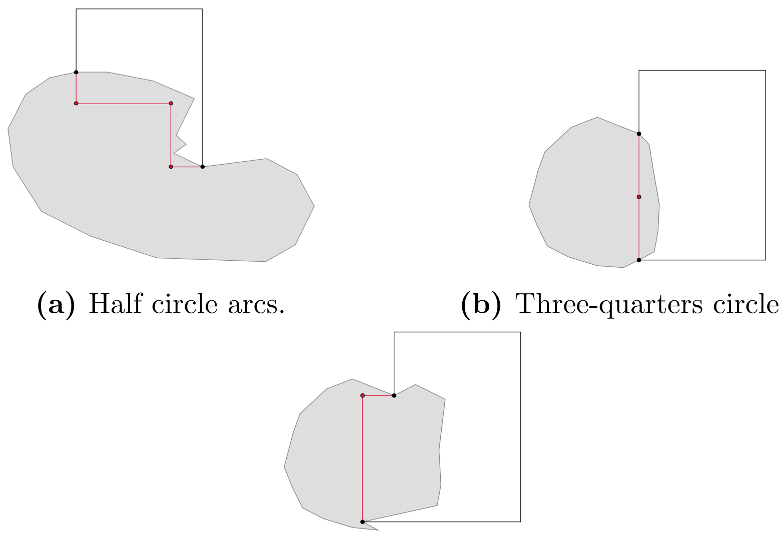
- either needs to have no edge routed in the direction where the two endpoints need to have the same coordinate
- or there need to be at least two: one of them needs to increase the common coordinates value, the other needs to decrease it. In the figure both vertical edges on the path increase the difference of the vertical coordinate of the endpoints

For three-quarters circle arcs on the other hand, we know that both endpoints need to be positioned on a diagonal. Thus at least two edges are needed to be part of any path between the two endpoints where there must be at least one horizontal and one vertical edge (unlike in Figure 6.13b). Also both of those edges between the arcs need to be able to change the distance between the two endpoints in the correct way (unlike in Figure 6.13c).

For quarter circle arcs, it is easy to see, that paths cannot violate their endpoint positioning.

There are also interactions between multiple arcs that can prevent a feasible drawing (cf. Figure 6.14). In Figure 6.14a two 90° arcs have a common endpoint. Therefore the two other endpoints of the arcs would need to be located on a diagonal. Since there is a path with only a horizontal edge between

[6]Those have the same total amount of bends in the orthogonal shape.

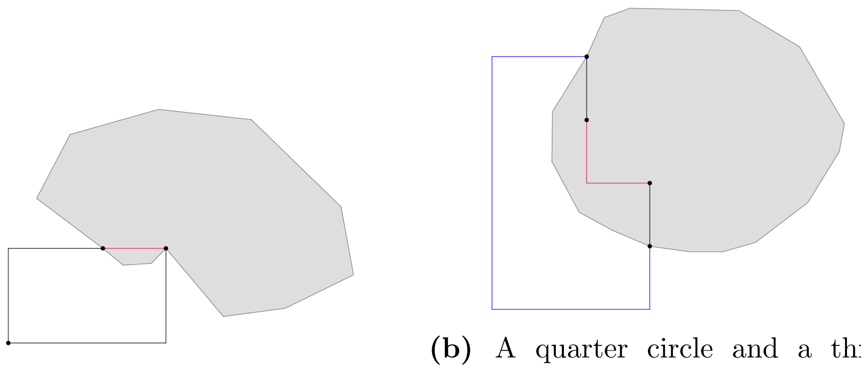


(a) Half circle arcs. **(b)** Three-quarters circle arc. **(c)** Three-quarters circle arc.

Figure 6.13: Forbidden path configurations (red) between the endpoints of a single arc.

the two endpoints this cannot be realized. Similar problems could also arise for other arc types. In Figure 6.14b both a blue three quarters circle arc and a red quarter circle arc need to be drawn. However, the red arc is part of a path between the two endpoints of the blue arc. Since the red arc requires its two endpoints to be located on a diagonal and the remaining edges of the path are all routed vertically, the endpoints of the blue arc cannot be positioned on a diagonal. Thus there is no feasible drawing. Therefore, when analyzing paths between endpoints of an arc, arcs along the path need to be treated differently than the sum of the segments they are composed of.

Finally there are also configurations which include an entire cycle of 4 arcs



(a) Two quarter circle arcs. **(b)** A quarter circle and a three-quarters circle arc.

Figure 6.14: Forbidden configurations between multiple arcs.

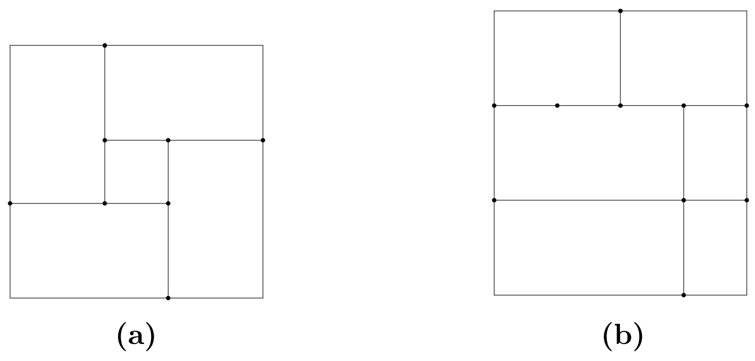

(a) (b)

Figure 6.15: Forbidden configurations between 4 arcs.

(cf. Figure 6.15): In Figure 6.15a the 4 arcs influence each others size in such a way, that if one starts drawing one, every subsequent has to be drawn larger than the one before. Finally one would also need to draw the first arc larger than it is drawn which is a self increasing loop. In Figure 6.15b the two arcs at the top of the drawing and the two arcs in the bottom of the drawing need to span over the same horizontal width with each arc needing to have a radius equal to half the horizontal width of the drawing. However there is a path between the edges separating both pairs of arcs which only includes one horizontal edge - therefore only one pair of arcs may be drawn correctly at each point in time.

While the configurations shown in Figures 6.13 and 6.14 are easy to understand, the configurations in 6.15 seem rather arbitrary. An algorithm for the shape step needs to prevent all the local configurations described in this section from appearing in the orthogonal shape. And there may be even more forbidden configurations as we only listed the ones we encountered in our practical experiments and could notice when manually checking the orthogonal shapes which could not be drawn. Thus, we cannot claim that this list is complete. However already with the current configurations identified, it seems as if computing a feasible orthogonal shape for SC_1 drawings is harder than computing the edge lengths.

Chapter 7

Conclusion and Open Problems

In this thesis, we motivated the use of a Topology-Shape-Metrics approach for perfect smooth orthogonal drawings which allows us to solve parts of the SC_1 drawing problem independently.

For the metrics step, i.e. the assignment of edge lengths to a given orthogonal shape, three different arc approximations were presented that reduce the perfect smooth orthogonal drawing problem to the problem of drawing an OC_1 drawing where additional constraints are imposed on the lengths of some edges. Also an ILP was introduced that can solve this constrained OC_1 drawing problem.

The simple arc approximation already succeeded in drawing most of the orthogonal shapes of benchmark test sets that admitted a perfect smooth orthogonal drawing. Despite this, we could also identify orthogonal shapes which could not be drawn due to the simple arc approximation's limitations.

In order to also draw those shapes with perfect smooth edge complexity, we introduced the staircase approximation. This improved approximation successfully solved the issues of the simple arc approximation. Also the staircase approximation improved the quality of the drawings w.r.t. total edge length. While the staircase approximation already could improve our results for small values of its input parameter increasing this parameter to larger values resulted in large computation times for particularly hard to draw orthogonal shapes.

Also we discussed the minimally optimal approximation as the smallest arc approximation only using axis-aligned straight-line segments which does not reserve any integer point for arc drawing. We could prove that this approximation is guaranteed to find a perfect smooth orthogonal drawing if such a drawing exists and by introducing another parameter we could even show that we can guarantee to find the perfect smooth orthogonal drawing with minimum total edge length. However, we also showed that we would require exponential time and space for this approach and argued that the staircase approximation would perform better if we are restricted to a certain number of edges for approximating arcs. Nevertheless, from our findings for the minimally optimal approximation we could conclude that also the staircase approximation is guaranteed to find the same results when its input parameter has exponential values. Further we could identify the issues that resulted in the exponential input parameter value and it remains unknown if they apply or if a lower bound for the input parameter could be proven to be optimal as well.

The practical evaluation suggested that already small values for the input parameter of the staircase approximation may suffice in practical applications but a proof for a general claim still must still be found. The complexity of finding a perfect smooth orthogonal drawing of a given orthogonal shape still remains unknown mostly due to the fact that it is unknown whether an orthogonal shape may only admit for a single family of perfect smooth orthogonal drawings where each drawing can be obtained from any other drawing of the family by scaling operations.

In the discussion of our results we advised to only apply the staircase approximation with small input parameter values as it often suffices and large values can result in too large run times. Also we motivated nearly perfect smooth orthogonal drawings that may help to reduce the total edge length or make it possible to draw more orthogonal shapes once further studied. Another suggested improvement is to fix the number of edge segments approximating an arc but varying their lengths. For this approach further testing is required. By investigating shapes that could not be drawn by our approaches we could identify local configurations in orthogonal shapes that make it impossible for the shape to admit for a perfect smooth orthogonal drawing.

Finally, the following open problems could be identified:

1. Investigate whether each orthogonal shape that admits for a perfect smooth orthogonal drawing also admits for another perfect smooth orthogonal drawing that is not just a scaled version of the first. Such a result could improve our complexity claim for our reduction from the drawing problem to ILP to polynomial time and polynomial space.

2. Find an algorithm that given a planar embedding computes an orthogonal shape which admits a perfect smooth orthogonal drawing (Shape step in the TSM framework). First insights have been discussed in Chapter 6 but also suggest that the problem of finding such a shape might be very difficult. However, possibly a constructive approach may only create local configurations which can be proven to be drawn perfect smooth orthogonally. Also it is not known yet which planar embeddings admit for a such a shape

3. Investigate which edge lengths are most useful for a given number of edges in a staircase approximation. If it can be proven that the connector edge

length is not as restricting as for the simple approximation, minimizing the area for arc drawing would be best. However, if the connector edge length is still important for the staircase approximation, it should be minimized instead.

4. Identify the graphs that admit for a k-nearly perfect smooth orthogonal drawing, preferably for very small constant k's. Allowing a few edges to be drawn with non-perfect smooth orthogonal complexity may also allow for algorithms with better run time complexity than our ILP approach. Also, the area requirement for nearly perfect smooth orthogonal drawings may have a better upper bound. Thus, nearly perfect smooth orthogonal drawings might provide better results for usage in practice while still retaining a very low edge complexity.

Bibliography

[ABK$^+$14] Muhammad Jawaherul Alam, Michael A. Bekos, Michael Kaufmann, Philipp Kindermann, Stephen G. Kobourov, and Alexander Wolff. Smooth orthogonal drawings of planar graphs. In *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, pages 144–155, 2014.

[BETT98] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1998.

[BGPR14] Michael A. Bekos, Martin Gronemann, Sergey Pupyrev, and Chrysanthi N. Raftopoulou. Perfect smooth orthogonal drawings. In *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications, Chania, Crete, Greece, July 7-9, 2014*, pages 76–81, 2014.

[BKK$^+$14] Michael A. Bekos, Michael Kaufmann, Robert Krug, Thorsten Ludwig, Stefan Näher, and Vincenzo Roselli. Slanted orthogonal drawings: Model, algorithms and evaluations. *Journal of Graph Algorithms and Applications*, 18(3):459–489, 2014.

[BKKS13] Michael A. Bekos, Michael Kaufmann, Stephen G. Kobourov, and Antonios Symvonis. Smooth orthogonal layouts. *Journal of Graph Algorithms and Applications*, 17(5):575–595, 2013.

[BLTW15] Pierre Bonami, Andrea Lodi, Andrea Tramontani, and Sven Wiese. On mathematical programming with indicator constraints. *Math. Program.*, 151(1):191–223, June 2015.

[DFPP90] H. De Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[EFK01] Markus Eiglsperger, Sándor P. Fekete, and Gunnar W. Klau. Drawing graphs. chapter Orthogonal Graph Drawing, pages 121–171. Springer-Verlag, London, UK, UK, 2001.

[HEH09] W. Huang, P. Eades, and Seok-Hee Hong. A graph reading behavior: Geodesic-path tendency. In *2009 IEEE Pacific Visualization Symposium*, pages 137–144, April 2009.

[Hä14] Bernhard Häussner. Implementierung eines Algorithmus für das glatt-orthogonale Zeichnen planarer Graphen. B.S. Thesis, Julius-Maximilians-Universität Würzburg, 2014.

[NR04a] Takao Nishizeki and Md. Saidur Rahman. *Graph Theoretic Foundations*, volume 12 of *Lecture Notes Series on Computing*, chapter 2, pages 19–31. World Scientific Publishing Co. Pte. Ltd., 2004.

[NR04b] Takao Nishizeki and Md. Saidur Rahman. *Properties of Drawings*, volume 12 of *Lecture Notes Series on Computing*, chapter 1.4, pages 10–11. World Scientific Publishing Co. Pte. Ltd., 2004.

[Tam87] Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, June 1987.

[Tar74] R.Endre Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2(6):160 – 161, 1974.

[Van10] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*, volume 196 of *International Series in Operations Research & Management Science.* Springer US, 3. edition, 2010.

[Wei01] René Weiskircher. *Drawing Planar Graphs*, pages 23–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Tübingen, den 21. Oktober 2016 Unterschrift