Karlsruhe Institute of Technology

Drawing Metro Maps on Concentric Circles

Master Thesis of

Lukas Barth

At the Department of Informatics
Institute of Theoretical Computer Science

Reviewers:	Dr. Martin Nöllenburg
	Prof. Dr. rer. nat. Peter Sanders
Advisor:	Benjamin Niedermann

Time Period: 16th July 2015 – 15th January 2016

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 15th January 2016

Abstract

This thesis examines algorithms for the drawing of metro maps. The most important elements of a metro map are stations and lines connecting the stations. In the context of this thesis, we restrict the elements used for representing lines to one of two classes: On the one hand, circular segments lying on circles with a common center S are allowed. On the other hand, line segments lying on lines passing through S are allowed.

To make the metro maps as legible as possible, we try to create a drawing in which the lines bend as little as possible. To plan a journey in the metro network, the users of metro maps must be able to quickly and correctly follow the lines visually. In this undertaking, bends are a disturbance. To this end, we adapt the *Topology-Shape-Metrics* framework by Roberto Tamassia, the purpose of which originally is bend minimization in orthogonal drawings of graphs. A second objective is producing drawings that take up as little space as possible. For doing so, we present a compaction approach based on *Simulated Annealing*. The compaction metaheuristic proposed by us is also suitable for performing the *Shape* step of the unaltered *Topology-Shape-Metrics* framework, too. Thus, we present a general improvement of this framework in itself.

We practically evaluate the adapted framework for drawing metro maps with concentric circles and demonstrate its usability even for complex metro networks such as Berlin or London, at the same time also determining reasonable values for the parameters of our framework.

For two subproblems that must be solved when drawing metro maps, we show $\mathcal{NP}$-hardness: First, a given metro network must be converted into a graph of maximum degree 4 since vertices with a larger degree cannot be represented in the desired drawing style. It would be desirable to do so in such a way that the best drawing of the resulting graph has as few edge-bends as possible; we show this problem to be $\mathcal{NP}$-hard. Second, the final step of our framework compactifies a preliminary drawing. We show that achieving an optimal compaction is also $\mathcal{NP}$-hard.

Furthermore, we examine another naturally arising idea: While the framework used by us minimizes the number of edge-bends for an orthogonal drawing style, which we then transform into the desired drawing style, it would be desirable to perform bend-minimization with a concentric-circle drawing style directly. We model the problem and illustrate some arising difficulties which make direct transferral of the used techniques from the orthogonal case impossible.

Deutsche Zusammenfassung

In dieser Arbeit geht es um das algorithmische Zeichnen von Nahverkehrsnetzplänen. Die wichtigsten zu zeichnenden Elemente eines solchen Netzplanes sind Stationen und Linien, die diese Stationen verbinden. In dieser Arbeit beschränken wir uns darauf, die Linien nur mit den folgenden zwei Elementen darzustellen: Zum einen sind Kreissegmente erlaubt, die auf Kreisen mit einem gemeinsamen Zentrum S liegen. Zum anderen sind Strecken erlaubt, die auf Linien durch S liegen.

Um die Netzpläne möglichst lesbar zu gestalten, versuchen wir eine Zeichnung zu erzeugen, in der die Linien möglichst selten abknicken. Beim Planen von Fahrten müssen die Benutzer der Pläne in der Lage sein, diesen Linien schnell und sicher

zu folgen, wobei Knicke störend wirken. Hierzu passen wir das *Topology-Shape-Metrics*-Framework von Roberto Tamassia an, welches in seiner ursprünglichen Form für die Knickminimierung in orthogonalen Graphzeichnungen gedacht ist. Des Weiteren möchten wir möglichst kompakte Zeichnungen erstellen. Hierfür stellen wir einen auf *Simulated Annealing* basierenden Ansatz vor. Die von uns für die Kompaktifizierung vorgestellte Metaheuristik ist auch geeigneit, den *Shape*-Schritt in einem nicht modifizierten *Topology-Shape-Metrics*-Framework zu übernehmen. Damit stellen wir also auch eine generelle Verbesserung dieses Frameworks vor.

Das sich mit unseren Änderungen insgesamt ergebende Framework für das Zeichnen von Netzplänen mit konzentrischen Kreisen evaluieren wir praktisch und demonstrieren seine Praktikabilität auch für komplexe Netzwerke wie zum Beispiel Berlin und London. Gleichzeitig bestimmen wir so auch sinnvolle Parameter für unser Framework.

Von zwei Teilproblemen, die zur Erstellung einer guten Zeichnung gelöst werden müssen, zeigen wir die $\mathcal{NP}$-Schwere: Zum einen muss ein gegebenes Verkehrsnetz in einen Graphen mit Maximalgrad 4 umgewandelt werden, da Knoten mit einem höheren Grad im gewünschten Zeichenstil nicht darstellbar sind. Wir zeigen, dass es $\mathcal{NP}$-schwer ist, den Graphen so in einen Graphen mit Maximalgrad 4 umzuwandeln, dass die bestmögliche Zeichnung möglichst wenige geknickte Kanten aufweist. Zum anderen möchten wir in einem abschließenden Schritt eine vorläufige Zeichnung kompaktifizieren. Wir zeigen, dass dies ebenfalls $\mathcal{NP}$-schwer ist.

Außerdem gehen wir kurz auf eine weitere, sich natürlich ergebende Idee ein: Während das von uns verwendete Framework die Knickminimierung in einem orthogonalen Zeichenstil durchführt, den wir dann später in den gewünschten Zeichenstil überführen, wäre es wünschenswert, die Knickminimierung direkt für eine Zeichnung mit konzentrischen Kreisen durchzuführen. Wir modellieren dies und zeigen Probleme auf, die die direkte Übertragung der Technik aus dem orthogonalen Fall verhindern.

Contents

1. Introduction

Metro Maps are maps of public transit networks. These maps are primarily used by the passengers of the transit network for navigation and route planning. Virtually any city with a public transit network also has a variety of metro maps, differing in size, style and the information displayed. Most of the time, metro maps are schematic maps, focusing on the topology of the metro network instead of representing the topography of the city and its surroundings. There are a variety of drawing styles used to draw these metro maps. In the most common drawing style, metro lines are represented by straight lines running vertically, horizontally or at 45° angles. This style is called an *octolinear* drawing style (also called *octilinear* by some authors) because of the eight possible orientations for edges. Stations are represented as disks or rectangles with rounded corners drawn upon the metro lines.

Most of these maps do contain only a limited number of other features, such as topographic information (like the course of a river), tariff information or the like. An example of such a drawing can be seen in Figure 1.1. However, many variations to the octolinear drawing style have been used as well, for example styles that allow only six different orientations of lines, called *hexalinear* drawings (see an example in Figure 1.2a), or styles drawing metro lines as Bézier curves (see an example in Figure 1.2b). Recently, the circular nature of the London Underground inspired Maxwell J. Roberts [RNC16] to try a drawing style that emphasizes the existence of these circles, drawing metro lines with concentric circles. The initial maps produced by Roberts were featured in numerous media, and thus sparked interest in drawing circular metro maps. In this drawing style, a geometric point S is chosen as center, and metro lines are drawn as a sequence of circular arc segments, the underlying circles of which have S as their center, and straight line segments, the underlying lines of which pass through S; an example can be found in Figure 1.2c. This is the drawing style that this thesis examines.

Figure 1.1.: Excerpt from a Berlin metro map using concentric circles. Image copyright by Maxwell J. Roberts.

The tasks users try to solve with these schematic maps usually include locating single metro stations, for example points of departure or arrival, and planning complex routes that include interchanges at stations. Thus, the ability to quickly locate stations as well

(a) Hexalinear Design (b) Curvilinear Design (c) Concentric Circle Design

Figure 1.2.: Different drawing styles applied to the same excerpt of the Berlin metro map. Images copyright by Maxwell J. Roberts, www.tubemapcentral.com

as being able to quickly find (shortest) routes within the depicted transit network are objectives for which the drawing must be optimized.

Historically, the first metro maps to become famous were the maps of the London Underground, first published by Henry Beck in the year 1933 [Gar94]. They adopted the principles of the octolinear drawing style stated above, and have contributed to the fact that this style of drawing metro maps has become the de-facto standard.

So far, all metro maps that are actually being used have been designed by hand, although computer assistance might have been used. This is possible because the number of metro networks in the world is manageable, and these networks change rarely. However, even for human designers, the task of creating an easily understandable metro map is far from trivial. An example for this this is the Paris metro: All its official metro maps up until 1982 were drawn mostly on top of topological maps, with stations at their topological locations and metro lines following their actual course through the city (cf. [OPL08, p. 155]) - thus, these maps could not really be considered schematizations. There were plenty of unofficial maps to be bought, even one map done by Henry Beck, which adhered to the design principles he applied to his famous London Underground maps. However, all maps not closely representing the topological layout of the city were refused by RATP, the Paris metro authority, often arguing that these kinds of maps do "not suit Paris" [OPL08, p. 153]. Development of the Réseau Express Régional (Regional Express Network - RER) in the 1960s and 1970s increased the density of the transit network in Paris considerably, and forced RATP to reconsider changing the official map to something more abstract, leading to the first official map loosely adhering to Beck's principles in 1982. Since then, many more official versions have been produced, with the latest dating from 2010. But even after many generations of Paris metro maps and much manual optimization, experimentation, and evaluation, the current Paris metro map is still subject to much criticism (cf. [RNL+13]).

But even though most metro mapping efforts are done manually, there are applications where quickly-adapting metro map drawings would be desirable: For example, with the widespread use of smart phones, which feature considerable computational power but relatively small screens, it would be welcomed if it was possible to automatically adapt the metro map displayed on a smart phone's screen to the user's wishes. Thinkable adaptions include focusing the map on certain areas, highlighting or hiding map elements, rearranging the map to make certain routes especially visible, and many others.

This thesis focuses on developing algorithms that produce metro maps adhering to the circular style introduced above. To produce metro map drawings with a maximum legibility, our primary concern is to minimize the number of bent metro lines, since this facilitates

quickly and correctly following the metro lines visually. In accordance with Roberto Tamassia's *Topology-Shape-Metrics* framework, we perform bend minimization in a first step via a specialized flow network, producing a preliminary layout of the graph in which the shape of all edges is fixed. In a second step, we compactify this preliminary drawing using a metaheuristic approach based on *Simulated Annealing*. An experimental evaluation practically determines good values for the parameters of our framework and demonstrates the feasibility of our approach even for complex metro networks such as Berlin or London.

Finally, it should be mentioned that, although throughout this thesis we are always talking about metro map drawings, the metro map layout problem can also be seen as a drawing metaphor that may be applied to other types of information. For example, certain graphs representing the interaction of proteins in cancer cells, so called metabolic pathways, have a structure that is similar to the structure of metro networks [HW02]. Other examples of information sets that potentially could be visualized using techniques intended for metro maps include automated layouts of project plans [SRB+05], i.e. plans visualizing the collaboration and the dependencies of different teams cooperating on a larger project, overview maps of websites [SGSK01], or certain subproblems of VLSI design [Tei02]. In all of these examples, the underlying information may change a lot faster than with metro networks, which strengthens the need for an automated way of rapidly creating circular maps.

1.1. Contribution and Outline

We now give a broad overview of the contents of this thesis and summarize the results presented.

In Chapter 2, we present the model that forms the basis for all presented techniques. We formally state the problems that our approaches are intended to solve and introduce some frequently used definitions. Chapter 3 recapitulates some techniques introduced by other authors, which are used and modified throughout this thesis.

In Section 4.1 throughout Section 4.3, we present multiple modifications of Tamassia's Topology-Shape-Metrics framework, adapting it to produce circular metro map drawings. Section 4.1 introduces an extension allowing us to produce a circular metro map drawing, while the Topology-Shape-Metrics framework originally was intended for orthogonal drawings. In Section 4.2, we demonstrate techniques for adapting the pure edge-bend-minimization approach of the framework to a setting more suited for metro maps, avoiding bends on metro lines at stations as well as on edges.

To finally produce a compact drawing, we present a compaction approach based on simulated annealing in Section 4.3. This approach is not specific to our use case and can therefore also be used in unmodified versions of the Topology-Shape-Metrics framework. Our technique not only improves on Tamassia's original technique regarding area minimization, it also makes it possible to optimize the drawing's metrics for multiple criteria.

In Section 4.4, we show $\mathcal{NP}$-hardness of the subproblem of producing an area-minimal circular metro map drawing based on the intermediate results computed by the modified Topology-Shape-Metrics framework. We proceed to show $\mathcal{NP}$-hardness of another subproblem in Section 4.5, namely converting the input data into a graph of maximum degree 4 so that the number of edge-bends are minimized.

We conclude Chapter 4 with an experimental evaluation of the proposed techniques in Section 4.6, in which we also give recommendations on how to parameterize our framework.

In Chapter 5, we consider the idea of transferring the bend-minimization technique used throughout Chapter 4 from an orthogonal to a circular setting. We describe why the intuitive way of doing so, while perhaps seeming obvious, does not lead to a valid drawing.

Finally, Chapter 6 wraps up this thesis with a subjective assessment of the results presented and an outlook to future work.

1.2. Related Work

This section gives an overview of related research that has already been conducted. First, we give a broad overview of research regarding drawing metro maps in general, and second we summarize research approaches into drawing metro maps algorithmically.

Drawing Metro Maps

Within the information visualization and the psychology communities, some research into drawing metro maps in general has been conducted. For example, Galotti et al. [GPB15] examined an *information limit*, i.e. a maximum amount of information that humans can usually process while planning a trip; they state that in large cities, 80% of the trips exceed this limit. On the other hand, Bartram [Bar80] confirms in a user study that a schematic map is superior to a geographic map or textual descriptions regarding the route planning performance of his study participants. Some publications also concern themselves with advantages and disadvantages of actual official metro maps: For example Avelar and Hurni [AH06] closely examine the official metro maps of Zürich, Madrid, and others.

Also of interest regarding this thesis is research comparing different drawing styles. Most importantly, Roberts et al. [RNC16] compare a classic octolinear metro map of the Berlin metro network to one drawn with concentric circles, i.e. in the drawing style this thesis examines. They performed a user study, in which participants where asked to plan journeys using maps of both types, and measured objective performance as well as subjective assessments of both maps. Objectively, while both drawing styles resulted in similar quality of the planned journeys, concentric-circle maps performed slightly worse in terms of the time it took the participants to plan their journeys. The authors also discovered a dissociation between objective and subjective measures, i.e. a tendency for map users to subjectively rate maps contrary to the objective performance that these users achieve with them. Subjective ratings also favored the octolinear drawing style. However, the authors partially explain this with the *exposure effect*, which is a well-documented effect in psychology (see for example Bornstein [Bor89]). It states that the mere familiarity with one type of stimuli results in users favoring these stimuli over other, less familiar stimuli. Since virtually all official metro maps today employ the octolinear style, this might partially explain the participants' bias towards this type of map. However, the authors explicitly state that more research is necessary in this field; particularly studies comparing more than just one fixed pair of actual maps would be desirable.

Drawing Metro Maps Algorithmically

The results outlined so far mostly concern themselves with metro maps and their quality and usability, and can be a useful guideline for designers of metro maps. However, our aim is to algorithmically draw metro maps. To apply results from the field of Graph Drawing, we treat a metro map as a geometrically embedded graph, in which edges represent (possibly multiple) metro lines, and vertices represent metro stations. With this interpretation, drawing metro maps and similar graphs algorithmically has been studied to some extent. Nöllenburg [Nö14] not only summarizes most results, but also gives a comprehensive list of objectives and design principles usually applied to metro map drawing. A first important result is also given by Nöllenburg [Nö05], namely that minimizing the number of edge-bends while also preserving a given embedding is $\mathcal{NP}$-hard in an octolinear metro map drawing. Since minimizing the number of edge-bends is one of the primary concerns in many settings,

no exact polynomial-time algorithms are to be expected for most metro map drawing efforts.

Consequentially, some well-known graph drawing heuristics have been adapted to draw metro maps: Hong et al. [HMdN06] adapt force-directed layout techniques to produce octolinear drawings by adding a force pulling edges towards one of the allowed orientations. Also using a force-directed approach, Fink et al. [FHN+13] present an approach for representing metro lines with Bézier curves.

Aside from heuristic approaches, there are a number of publications evaluating Mixed-Integer Programming (MIP) as a solution technique. Nöllenburg and Wolff [NW11] present an MIP that not only computes provable optimal (regarding the authors' quality criteria) octolinear metro-map layouts within a maximum of several hours for complex metro networks, but that also solves the label placement problem (see below). Based on this MIP, Wu et al. [WTLY12][WTH+13] present approaches for generating map drawings that highlight certain routes or are annotated with large labels. Finally, Fink et al. [FLW14] introduce an MIP for concentric metro map drawings, taking into account a multitude of quality criteria. However, the performance of their MIP limits the applicability of their approach to rather small instances such as Vienna or Montréal.

Other Subproblems

There are some problems that must be solved when algorithmically drawing a metro map, which we do not examine in this thesis. For one, after computing a geometric embedding of the graph representing the metro network, it is usually desired to also place labels in the drawing, denoting the names of the stations and other information. Most of the time, the algorithm for placing such labels is tightly coupled with the algorithm computing the geometric embedding, since the geometric embedding determines the space available for the labels. However, if labels are to be placed after the geometric embedding of the graph has been determined, the problem can be seen as a general *map labeling* problem. There is plenty of research in this area; Wolff [Wol96] provides an extensive bibliography. However, there are also approaches focusing specifically on labeling metro maps. Haunert and Niedermann [HN15] not only show $\mathcal{NP}$-hardness of labeling even a drawing of a single metro line, but also give an efficient algorithm for a restricted version of the problem. Also, the MIP presented by Nöllenburg and Wolff [NW11] produces a labeling of the map, as does the force-directed approach by Hong et al. [HMdN06].

Another subproblem not being examined here is the problem of minimizing the number of metro-line crossings that happen between metro lines running along the same edges. This problem has been shown to be $\mathcal{NP}$-hard even in some restricted cases by Fink and Pupyrev [FP13].

2. Model

This thesis is about algorithms for drawing graphs. To reasonably discuss such algorithms, it is necessary to formally specify what graphs are to be drawn, and what constraints apply to the desired drawings. In the case of this thesis, the graphs to be drawn are graphs derived from metro networks, and the primary purpose of the constraints applied to the drawing is to make the resulting drawings as legible as possible for travelers using the map to navigate the metro network.

We now formalize the model used throughout this thesis as well as some frequently used definitions.

2.1. General

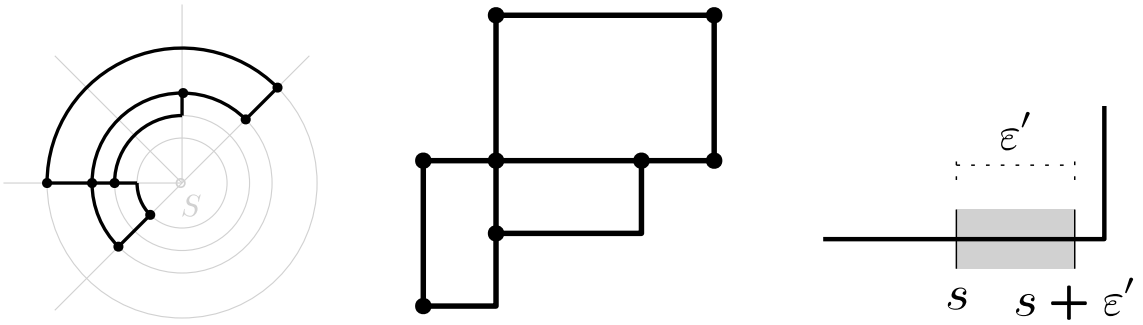
When drawing graphs algorithmically, one must formally specify what the desired output of the algorithm is. We call these constraints the *drawing criteria*. Usually, one such requirement limits the geometric objects available to the algorithm for representing the graph's elements. While vertices are usually just represented by small dots or boxes, the question of how to represent the edges is more interesting. In this thesis, we use the concepts of *orthogonal* and *ortho-radial* drawings. Both specify which geometric objects may be used to represent edges of the graph.

Intuitively, in an *orthogonal* drawing, every edge must be represented by a connected sequence of axis-aligned line segments, where the axes are two orthogonal, straight axes, i.e. the axes of a common cartesian coordinate system. Conversely, in an *ortho-radial* drawing, every edge must be represented by an alternating sequence of two possible geometric objects:

- line segments, which all lie on lines going through a common point S
- circle segments, which all lie on circles that have S as their center

Figure 2.1a shows an example of such an ortho-radial drawing of a graph with S as the center. Figure 2.1b shows an orthogonal drawing of the same graph.

We now first formalize requirements that the ortho-radial and orthogonal drawing style both have in common. We call this a *drawing with independent directions*:



(a) Example of an ortho-radial drawing

(b) Example of an orthogonal drawing

(c) Illustration of a small neighborhood of a point on an edge

Definition 2.1 (Drawing with Independent Directions)**.** *Given a graph $G = (V, E)$, a* drawing with independent directions *consists of:*

- *a function $f_V \colon V \to \mathbb{R} \times \mathbb{R}$ assigning two-dimensional coordinates to the vertices*
- *a function $f_E \colon E \times [0, 1] \to \mathbb{R} \times \mathbb{R}$ assigning a two-dimensional coordinate to every point on an edge*

so that

(1) $\forall e \in E, \forall s \in [0, 1) \colon \exists \varepsilon' > 0, \delta \in \mathbb{R} \colon \forall \varepsilon \leq \varepsilon', \varepsilon > 0 \colon$
$(f_E(e, s) = f_E(e, s + \varepsilon) + (0, \delta) \vee f_E(e, s) = f_E(e, s + \varepsilon) + (\delta, 0))$

(2) $\forall e \in E \colon e = (v_1, v_2) \Rightarrow$
$(f_E(e, 0) = f_V(v_1) \wedge f_E(e, 1) = f_V(v_2)) \vee ((f_E(e, 1) = f_V(v_1) \wedge f_E(e, 0) = f_V(v_2))$

(3) $\forall e \in E, s \in [0, 1) \colon \lim_{\delta \to 0} |f_E(e, s) - f_E(e, s + \delta)| = 0$

Condition (1) makes sure that all edges consist of just a sequence of segments, where every segment is parallel to one of the two possible dimensions by requiring that for every point on the edge, you find a small neighborhood of points on the edge around it, the coordinates of which differ in one dimension only. For the case that the dimensions are treated as cartesian coordinates, this is illustrated in Figure 2.1c: The edge shown consists of a horizontal and a vertical segment. Still, for a point s on it, a small neighborhood from s to $s + \varepsilon'$ can be found, in which the edge is either only horizontal (as in the example) or only vertical. Condition (2) ensures that every edge is routed to the position of its respective vertices. Condition (3) requires all edges to be continuous.

With this in place, we can now formally define the *ortho-radial* and the *orthogonal* drawing styles:

Definition 2.2 (Orthogonal Drawing)**.** *Given a graph $G = (V, E)$, an* orthogonal drawing *consists of two functions f_V and f_E as defined in Definition 2.1, in which all two-dimensional coordinates are interpreted as cartesian coordinates, with the first coordinate representing the x-dimension and the second coordinate representing the y-dimension.*

Definition 2.3 (Ortho-Radial Drawing)**.** *Given a graph $G = (V, E)$, an* ortho-radial drawing *consists of two functions f_V and f_E as defined in Definition 2.1, in which all two-dimensional coordinates are interpreted as polar coordinates, with the first dimension representing the radius, and the second dimension representing the angle.*

We need two more definitions: While the *outer face* of a drawing in both cases is the face that is unbounded, the *center face* only exists in an ortho-radial drawing and is the face that contains S. Note that both terms may refer to the same face of a drawing.

Since we often deal with two-dimensional coordinates throughout this thesis, we establish this notation for referring to individual coordinates: If c is a two-dimensional coordinate, c_1 refers to the first element (the x- or r-dimension), and c_2 refers to the second element (the y- or θ-dimension).

2.2. The Main Problem

In this section, we define the main problem that we try to present approaches for.

Input Data

Everything in this thesis works on data derived from a *metro network*. A metro network is the public transportation network of a certain region. It consists of *stations*, which may be bus stops, train stations, etc., and *lines*, which are a fixed sequence of stations. Each station is associated with a geographic coordinate.

Typical metro networks have a lot more information associated with them, such as timetables, metro lines that change the sequence of stations depending on the time, etc. We ignore all this additional information throughout this thesis. We also assume that in a metro line, every stop either appears only once, or appears twice and is the first and the last stop of a metro line. In the latter case, we treat the metro line as a closed loop.

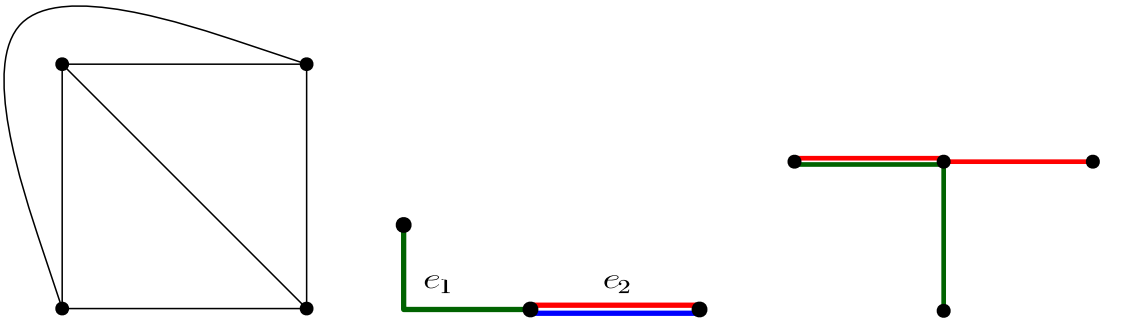
We transform this metro network into an undirected graph $G = (V, E)$, vertices of which represent the stations of the public transportation network, and two vertices have an edge between them if at least one metro line contains both stations consecutively. Every edge is annotated with the number of metro lines containing the respective two vertices consecutively, i.e. the number of metro lines going from one stop to the other without stopping in between. We also derive from the metro network $L \subseteq \{p \mid p \in V^k, k \in \{1, \dots |V|\} \wedge \forall (p_i, p_{i+1}) \colon \{p_i, p_{i+1}\} \in E\}$, a set of paths on G, where every path in L represents a metro line in the metro network.

Additionally, we derive a *combinatorial embedding* from the geographic positions given by the stations. We do this by simply ordering the edges around a vertex in the order that they would appear in in a straight-line drawing with the geographic positions as vertex positions. We assume this embedding to be planar.

Note that the latter assumption might not be true instantly, since the graph derived from the metro networks sometimes is non-planar. Also, the way we find the combinatorial embedding may sometimes introduce new non-planarities. Consider the example in Figure 2.2a: Clearly, a K_4 is planar. If, however, with this positioning of the vertices, the edges are ordered as they would be in a straight line drawing, the resulting combinatorial embedding is non-planar. In these cases, for every pair of edges that would cross in a straight-line drawing using the geographical coordinates, we insert a new vertex into the graph, and subdivide both edges with this vertex. This way, all crossings become vertices of degree 4 and the graph is planar.

Hence, an instance of our problem consists of:

- a graph $G = (V, E)$ with a combinatorial embedding
- a set L of paths on G, representing metro lines



(a) Example of a graph that would be non-planar if all edges were drawn as straight lines

(b) Example of an edge with multiple lines bending

(c) Example of a metro line bending at a station

Figure 2.2.: Illustrations of the problem model

The Drawing

We want to produce an ortho-radial drawing as defined above from this graph. To this end it is desirable to be able to choose which face is drawn as the center face, i.e. the face that contains the origin of the coordinate system in the final drawing. Aside from that, there are a few *drawing conventions*, which are requirements that the drawing must fulfill, and *quality criteria* which should be met as much as possible:

Drawing Conventions

(C1) The drawing must respect the geographical network topology, i.e. preserve the given combinatorial embedding.

(C2) All vertices must be separated from each other and from all edges by a minimum distance.

(C3) The drawing must be planar.

Quality Criteria

These are the quality criteria that we try to optimize:

(O1) Minimize the total area used. In case of an ortho-radial drawing, this is equivalent to minimizing the largest radius used.

(O2) Minimize the number of edge-bends, i.e. minimize the total number of circular and straight segments that all edges of the graph are composed of.

(O2a) Minimize the number of times a metro line bends on an edge.

(O3) Minimize the number of times a metro line bends at a station.

(O4) Minimize the standard deviation of the distance of two neighboring stations, where distance is measured along the edge between neighboring stations.

The reasoning behind criterion (O1) is straightforward: Any usable metro map must be printed or displayed on a limited area, thus must be scaled to fit. The less area the drawing consumes, the larger the scaling factor can be, making the drawing more readable.

Criteria (O2) and (O2a) are closely related: While (O2) aims at minimizing the raw number of times that an edge of the graph representing the metro network is bent, (O2a) weights

each edge with the number of metro lines running via that edge. The fact that bends should be avoided in the first place is not just intuitive, but also an important design criterion for metro maps, as for example emphasized by Maxwell J. Roberts in his works on metro map design (cf. [Rob12, p. 107]). That edges carrying a lot of metro lines are drawn without bends with a higher priority than edges with less metro lines is also intuitive, since these edges are not only looked at more often by people using the map, but the larger number of metro lines creates a visual disturbance by itself, amplifying the disturbance introduced by a bend. Figure 2.2b, in which different colors indicate individual metro lines, illustrates the case that shall be avoided by (O2a): Here, bending e_1 only bends one line. If e_2 was bent instead, this would bend two lines, which would be an inferior solution.

Criterion (O3) specifies that metro lines should pass straight through metro stations (or rather: their graphical representations) instead of bending at the point where the metro station is. Maxwell J. Roberts states that metro stations should always be placed on straight metro lines if possible, since the visual clutter especially around larger metro stations makes it hard to visually follow bent metro lines at these points. Figure 2.2c illustrates (O3): Here, the green line has a 90° bend at the station, which is to be avoided. However, if the green line was to be drawn without a bend, the red line would have a 90° bend at the same station. Thus, this drawing is optimal in terms of (O3).

Finally, criterion (O4) tries to make the appearance of the metro map drawing more uniform by encouraging a uniform distribution of the metro stations over the course of their respective metro lines.

Objective Function

The objective function we try to optimize combines the aforementioned quality criteria (O1) - (O4). Since these criteria are competing with each other, we introduce a weighting factor for every one of the criteria: ω_{area} is the weight of $(O1)$, ω_{bends} the weight of $(O2)$, ω_{line} the weight of $(O2a)$, ω_{station} the weight of $(O3)$, and ω_{dist} the weight of $(O4)$.

The objective function is then:

Definition 2.4. *Main Objective*

$$
\begin{aligned}
OBJ = \quad & \omega_{area} \cdot \max_{v \in V}((f_V(v))_1) \\
+ \quad & \omega_{bends} \cdot \ \#\ \textit{of edge-bends in the drawing} \\
+ \quad & \omega_{line} \cdot \ \#\ \textit{of line-bends in the drawing} \\
+ \quad & \omega_{station} \cdot \ \#\ \textit{of lines bending at stations in the drawing} \\
+ \quad & \omega_{dist} \cdot \ \textit{standard deviation of the distance of neighboring stations}
\end{aligned}
$$

In conclusion, we sum up our main problem in the following definition:

Definition 2.5. Circular Metro Maps *Given a graph $G = (V, E)$ with a planar combinatorial embedding, a set L of paths on G and two not necessarily distinct faces f_{center} and f_{outer}, find an ortho-radial drawing of G so that (C1) - (C4) are satisfied, f_{center} is the center face, f_{outer} is the outer face, and OBJ is minimized.*

2.3. Common Definitions

We now introduce and define some terms used commonly throughout this thesis.

2.3.1. Rotation

In an orthogonal or ortho-radial representation of a connected graph, it is often necessary to talk about the shape of a path p, i.e. the question of how many right and left turns one takes when traversing the path. We formalize this as the rotation of an edge and a simple path p, which both are always defined in the context of an orthogonal (or ortho-radial) representation:

Definition 2.6 (Edge Rotation)**.** *Given a graph $G = (V, E)$ together with an orthogonal representation, the* edgerot$\colon E \to Z$ *function is defined as the number of right bends on e minus the number of left bends on e.*

Please note that for this definition to be useful, edges must be assumed to be directed. However, even in an undirected graph we can define a *Path Rotation* by inducing a direction on the edges of the path:

Definition 2.7 (Path Rotation)**.** *Given a simple path $p = (e_1, \ldots, e_k)$, let the edges of the path be directed as they are traversed in the path.*

Then, the $\mathrm{rot}_p\colon p \to \mathbb{Z}$ *function is defined as follows:*

- $\mathrm{rot}_p(e_1) = 0$
- *if the path makes a bend left between e_l and e_{l+1}, then*

$$\mathrm{rot}_p(e_{l+1}) = \mathrm{rot}_p(e_l) + \mathrm{edgerot}(e_l) - 1$$

- *if the path makes a bend right between e_l and e_{l+1}, then*

$$\mathrm{rot}_p(e_{l+1}) = \mathrm{rot}_p(e_l) + \mathrm{edgerot}(e_l) + 1$$

- *if the path makes no bend between e_l and e_{l+1}, then*

$$\mathrm{rot}_p(e_{l+1}) = \mathrm{rot}_p(e_l) + \mathrm{edgerot}(e_l)$$

We then define the rotation *of e_i (on p) to be the value of* $\mathrm{rot}_p(e_i)$.

If the path p is not simple, an edge might appear multiple times in p. If that is the case, it it important to note that the same edge might have different induced orientations at different occurrences in the path! We sometimes omit the index p if it is obvious which path is the reference. We can now derive from this the notion of a rotation on a cycle. Assume that p is a cycle, i.e. the start vertex of e_1 and the end vertex of e_k are the same. We then define:

Definition 2.8 (Cycle Rotation)**.** *Given a simple cycle $p = (e_1, \ldots, e_k)$, the* rotation of the cycle p *is:*

- $\mathrm{rot}_p(e_k) + \mathrm{edgerot}(e_k) + 1$ *if the cycle makes a bend left between e_k and e_1*
- $\mathrm{rot}_p(e_k) + \mathrm{edgerot}(e_k) - 1$ *if the cycle makes a bend right between e_k and e_1*
- $\mathrm{rot}_p(e_k) + \mathrm{edgerot}(e_k)$ *if the cycle makes no bend between e_k and e_1*

And finally, since faces of a planar graph can be interpreted as cycles, we define a rotation for them:

Definition 2.9 (Face Rotation)**.** *Given a face f, let p be the cycle of the face's edges, in clockwise order. The* rotation of f *is then the rotation of the cycle p.*

3. Preliminaries

In this chapter, we briefly summarize already existing techniques that we use and adapt throughout this thesis. More specifically, in Section 3.1, we present the *Topology-Shape-Metrics* framework proposed by Roberto Tamassia [Tam87], while in Section 3.2, we give an overview over the metaheuristic *Simulated Annealing*.

3.1. Summary of the Topology-Shape-Metrics Framework

In this thesis, the *Topology-Shape-Metrics* framework originally developed by Roberto Tamassia is used and modified extensively. This section briefly introduces the ideas behind the framework first described in [Tam87]. For more details, see the original paper, on which this section is based.

The Topology-Shape-Metrics framework's aim is to produce a planar drawing of a (planar) graph on a rectilinear grid. It does not fully specify an algorithm to do so, but rather a framework consisting of building blocks of such an algorithm.

According to this framework, the drawing is computed in three separate steps:

1. Find a planar embedding of the input graph (*Topology*)
2. Compute the shape of all edges and how they are distributed around their vertices (*Shape*)
3. Assign coordinates to all vertices and bends (*Metrics*)

The first step, finding a planar embedding, can be shown to be $\mathcal{NP}$-hard should the embedding be found in such a way that certain criteria (for example the number of edge-bends) are optimal in the resulting drawing (cf. [GT01]). Therefore, we (and Tamassia) just assume a combinatorial embedding to be given and do not further address this step.

In the following, we summarize steps two and three of Tamassia's framework, since these are the steps we use in our approach.

3.1.1. Orthogonal Representation

Of most interest to us is the second step, which results in what is called an *Orthogonal Representation*. This orthogonal representation describes the shape of all edges, i.e. how they bend, as well as the angles between all edges incident to a vertex. With this orthogonal

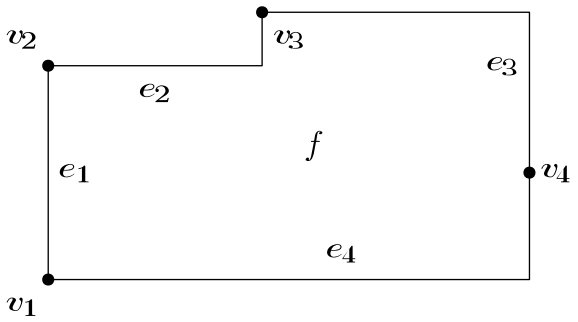


Figure 3.1.: Example of an orthogonal representation

representation, the drawing of the graph is almost fixed, with only the lengths of the edges being unspecified.[1]

Computing the orthogonal representation is done in such a way that the total number of bends on the edges is minimized. This minimality is an interesting property for our use case.

Formally, an orthogonal representation is defined as follows:

Definition 3.1 (Orthogonal Representation)**.** *Given a graph $G = (V, E)$, and a planar embedding of G resulting in the set of faces F. An orthogonal representation is a function $H\colon F \to \{E \times \{0,1\}^\star \times \{90, 180, 270, 360\}\}^*$, assigning to every $f \in F$ a circularly ordered list. All elements of these lists have the form (e, s, a), where*

- *e is an edge of G,*
- *s is a binary string,*
- *a is an integer in the set $\{90, 180, 270, 360\}$.*

With this definition, the **s** field assigns every edge (on every face) a sequence of left and right bends, where a 1 stands for a right and a 0 stands for a left bend. Note that, since faces are always specified in clockwise order, a right bend is always an inwards bend and a left bend is always an outwards bend for a face. Additionally, **a** specifies the angle between **e** and the next edge in f. The $\mathrm{rot}(f)$ function, as defined in Section 2.3.1, can then be computed by just looking at $H(f)$ (with $ZEROES(s)$ and $ONES(s)$ being the number of 1s and 0s in s, respectively):

$$\mathrm{rot}(f) = \mathrm{rot}(H(f)) = \sum_{(e,s,a)\in H(f)} ZEROES(s) - ONES(s) + (2 - a/90)$$

We illustrate this with an example in Figure 3.1. Here, the orthogonal representation looks as follows:

$$\begin{aligned} H(f) = (\quad & (e_1, \quad (), \quad 90), \\ & (e_2, \quad (0), \quad 90), \\ & (e_3, \quad (1), \quad 180), \\ & (e_4, \quad (1), \quad 90)) \end{aligned}$$

An orthogonal representation is *valid* if:

(1) The **e** lists for all faces conforms to the planar embedding of G, i.e.

$$\forall f \in F\colon ((e_i, \cdot, \cdot), (e_j, \cdot, \cdot)) \in H(f) \Rightarrow (e_i, e_j) \in f$$

[1] Also, strictly speaking, it is not specified how the drawing is rotated.

(2) Let $(e, s, a) \in H(f)$ and $(e, s', a') \in H(f')$ (note that e is the same in both entries), then s can be obtained from s' by reversing the string and exchanging 0's and 1's

(3) All faces are properly closed, i.e.

$$\forall f \in F\colon \sum_{(e,s,a)\in H(f)} \mathrm{rot}(s,a) = \begin{cases} -4 & \text{if f is the external face} \\ 4 & \text{if f is an internal face} \end{cases}$$

(4) For every vertex, the sum of angles between its incident edges is 360.

Property (1) makes sure that the representation describes the given graph. Property (2) states that the lists of bends of the same edge on two different faces must be consistent with each other. Observe that for every edge holds: If it appears in one direction on one of its incident faces, it appears in the other direction on its other incident face. With that in mind, *consistent* in this context means that the order as well as the direction of the bends must be reversed. In property (3), the fact that every face must be closed is enforced (with a special notion of closed for the outer face). Property (4) ensures that the incidences of every vertex are consistent.

3.1.2. Computing an Orthogonal Representation

The algorithm presented by Tamassia for computing such an orthogonal representation is designed to minimize the number of bends. Since we adapt the algorithm for our purposes, we now summarize the original version for the reader's convenience. Again, for full details, please refer to the original paper by Tamassia [Tam87].

Tamassia's algorithm works by computing an integer flow network on a graph that is a mix of the original graph and its dual. The basic idea is to represent 90° angles as one unit of flow in that integer flow network. We first introduce the Min-Cost-Flow problem, and then specify an instance of Min-Cost-Flow that computes an orthogonal representation.

Definition 3.2 (Min-Cost-Flow)**.** *Given*

- *a directed Graph* $G_f = (V_f, E_f)$
- *a demand function* $d\colon V_f \to \mathbb{Z}$*,*
- *a cost function* $c\colon E_f \to \mathbb{N}_0$*,*
- *an upper bound for the flow per edge:* $u\colon E_f \to \mathbb{N}_0 \cup \{\infty\}$*,*
- *a lower bound for the flow per edge:* $l\colon E_f \to \mathbb{N}_0$*.*

find a flow function $f\colon E_f \to \mathbb{N}_0$ *such that*

(1) $\forall e \in E_f\colon f(e) \leq u(e)$

(2) $\forall e \in E_f\colon f(e) \geq l(e)$

(3) $\forall v \in V_f\colon \sum_{(u,v)\in E_f} f((u,v)) - \sum_{(v,u)\in E_f} f((v,u)) = d(v)$

(4) $\sum_{(u,v)\in E_f} c((u,v))f((u,v))$ *is minimal among all* f *that satisfy (1) - (3)*

In this definition, requirements (1) and (2) enforce the per-edge capacities, while requirement (3) enforces that the demand (or surplus, which is modeled as negative demand) of every vertex is satisfied. Condition (4) is the minimality criterion. Solving a Min-Cost-Flow instance is in $\mathcal{P}$, as shown by Lawler [Law76].

We now specify an instance of Min-Cost-Flow that computes an orthogonal representation with a minimum number of bends:

Definition 3.3 (Min-Cost-Flow to compute an Orthogonal Representation)**.** *Given a graph $G = (V, E)$ with an embedding and the resulting bidirected dual graph $G' = (V', E')$, let v'_0 be the dual vertex of the outer face.*

We set:

(1) $V_f = V \cup V'$

(2) $E_f = E' \cup \{(v, f) : v \in V, f \in V' \wedge v \text{ is incident to the face represented by } f\}$

(3) $c(e) = \begin{cases} 1 & \text{if } e \in E' \\ 0 & \text{otherwise} \end{cases}$

(4) $\forall v' \in V' \colon d(v') = 4 - \deg(v')$

(5) $\forall v \in V \colon d(v) = \deg(v) - 4$

(6) $\forall e \in E_f \colon \begin{matrix} u(e) & = \infty \\ l(e) & = 0 \end{matrix}$

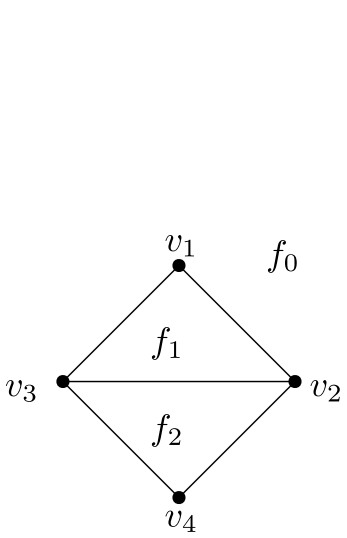
Here, condition (3) is used to minimize the number of bends in the orthogonal representation. Since every unit of flow on an edge between two faces later results in a bend on an edge of the orthogonal representation, we set the costs for these units of flow to 1. Condition (4) enforces that the correct number of additional 90° angles is distributed around every vertex in V (additionally to the one 90° angle that every incident face must be assigned in any case). With condition (5), the vertices representing faces get demand or surplus so that they have the correct amount of bends on their boundary, as explained below.[2]

An example is found in Figure 3.2: While Figure 3.2a shows an embedded example graph with four vertices and three faces, Figure 3.2b shows the flow network resulting from that graph.

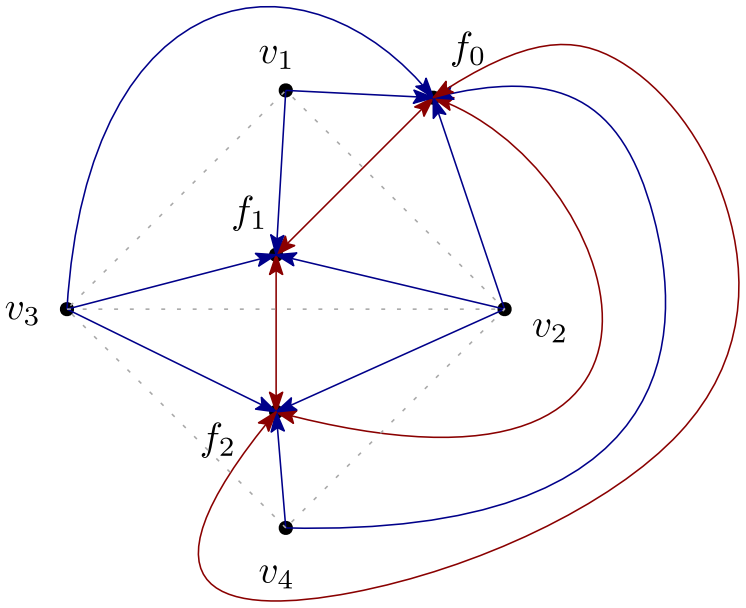
Every vertex v has on any of its incident faces at least a 90° angle; we call this the default case. This 90° angle corresponds to an inwards bend on that face, contributing 1 to the rotation of the face, see Figure 3.3a. If the vertex has degree 3 or less, it must distribute $4 - \deg(v)$ additional 90° angles between its incident faces. It is important to note that, although the first 90° angle contributed 1 to the face's rotation, these additional 90° angles do not further increase the rotation of the face, but instead decrease it by 1 each. If a vertex assigns a total angle of 180° to a face, the rotation contributed by this vertex is 0 (see Figure 3.3c), if it assigns 270°, the contributed rotation is -1 (see Figure 3.3b), and if it assigns 360°, the contributed rotation is -2 (see Figure 3.3d). Thus, flow from a vertex to a face decreases the total rotation of that face. Please note that since the edges between vertices in V and vertices in V' are directed from V to V' only, no flow can be pushed from vertices representing faces back to original vertices.

Flow on one of the dual edges (i.e. edges in E') represents bends on the original edges in E: If $e' \in E'$ carries a flow of k units in the solution, the dual edge e of e' has k bends in the orthogonal representation. Every bend has a 90° and a 270° angle. In this case, the 90° angle is assigned to the face that the flow originated from. See Figure 3.4: Figure 3.4b shows the resulting shape if the dual edge carries one unit of flow (in direction of the arrow), Figure 3.4c shows the result of two units of flow. Thus, for every face, incoming flow from one of the neighboring faces decreases the face's rotation by 1 per flow-unit, and outgoing flow to other faces increases the rotation by 1 per flow-unit. Altogether, any incoming flow, be it from vertices or faces, decreases a face's rotation, while any outgoing flow (which is only possible to neighboring faces) increases the rotation.

[2] Note that a negative demand value is a surplus!



(a) Example Graph



(b) Resulting Flow Network. Gray dotted edges do not belong to the flow network but are there to show the original graph. Blue edges are the edges going from vertices in V to the vertices representing incident faces. Red edges are the edges of the dual graph G'. Note that these are bidirected, as opposed to the blue edges.

Figure 3.2.: An example Graph and the resulting Flow Network

Now, if a face f has four vertices, it is by default supplied with a rotation of 4 by these vertices, but if $\deg(f) < 4$, then it needs $4 - \deg(f)$ additional rotation. This is solved by making the face a source of $4 - \deg(f)$ units of flow, which it has to push to its neighboring faces, resulting in a total rotation of 4 at this face. If, however, $\deg(f) > 4$, then the default case would supply f with a rotation that is $\deg(f) - 4$ too large. Thus, we make f a sink of $\deg(f) - 4$ units of flow, which it has to draw either from its vertices or its neighboring faces, resulting in a total rotation of 4.

Since the objective is to minimize the number of bends on edges, we make this problem a minimum-cost flow problem, associating every edge between faces with a per-unit cost of 1, and all other edges with a cost of 0. This results in the solution's cost to exactly correspond to the number of bends on edges in the resulting orthogonal representation.

3.1.3. Metrics

The final step in Tamassia's framework computes coordinates for vertices, which is done in two steps: First, all faces are rectangulated, i.e. subdivided into rectangles. Second, a rectangle compaction technique is applied.

Rectangulation

We demonstrate Tamassia's way of rectangulating faces with the example of Figure 3.5a. Here, face f has an outward bend at vertex v_0. Any face containing an outward bend can obviously not be rectangular. On the other hand, any face that has no outward bend can only have exactly four inward bends since it must have a rotation of 4. Thus, we show how to eliminate these outward bends. Tamassia argues that, as long as a graph layout contains at least one of these outward bends, it is always possible to find a bend such that

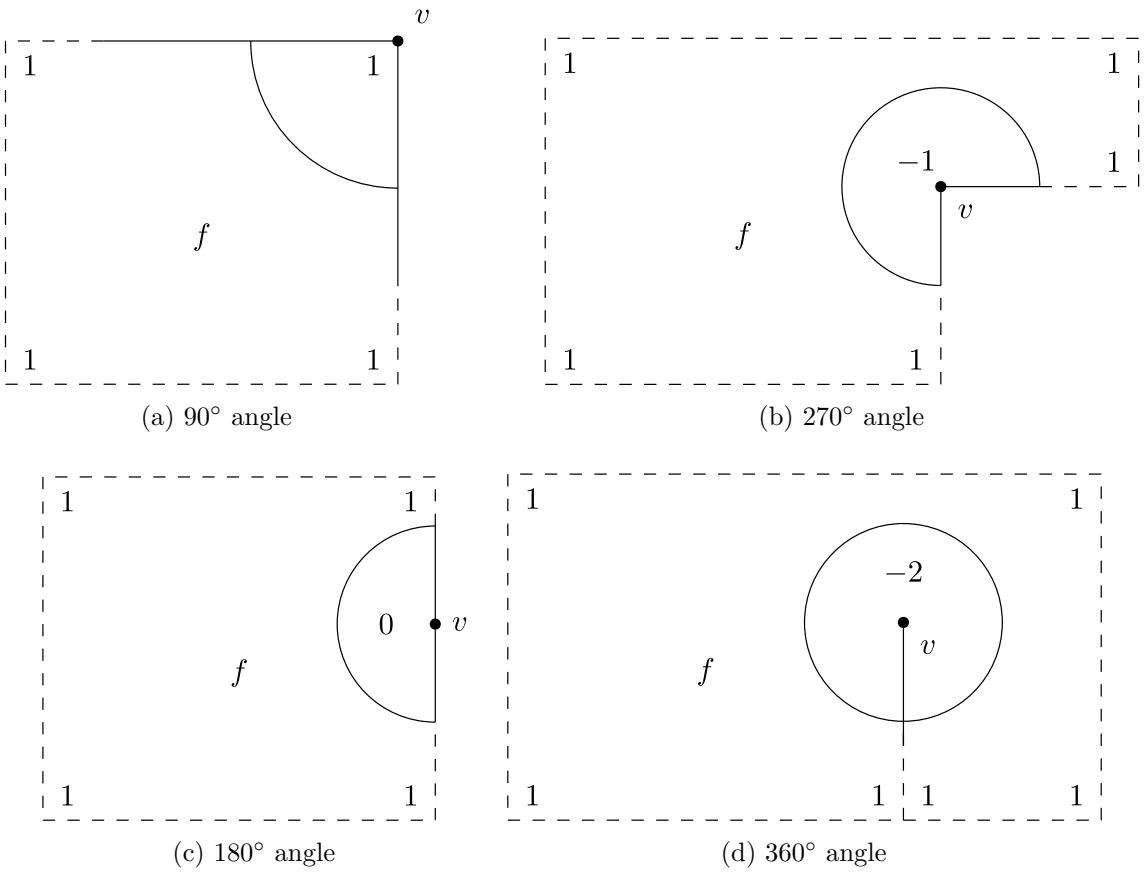


Figure 3.3.: Illustrating the rotation contributed by all possible angles around a vertex. Numbers specify the contributed rotation by the respective corner.

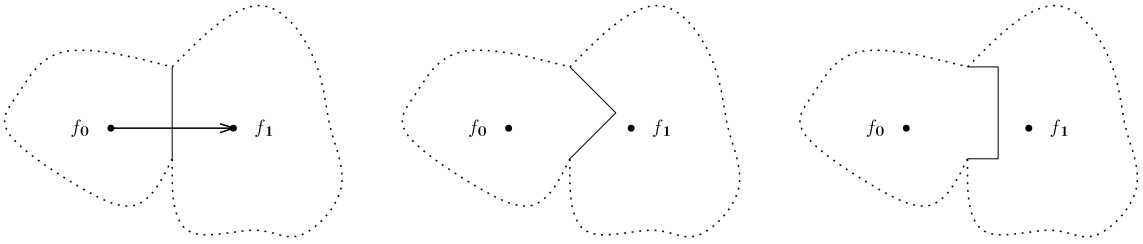
the next two bends following clockwise on its face (in the example of Figure 3.5a at v_1 and v_2) are inward bends. The proof for this can be found in [Tam87].

After these two inwards bends, we can find an edge (in our example e) such that we can connect the outwards bend to a new vertex on e. Doing so removes one outward bend from the graph. Tamassia repeats this until the orthogonal representation is rectangulated.

Rectangle Compaction

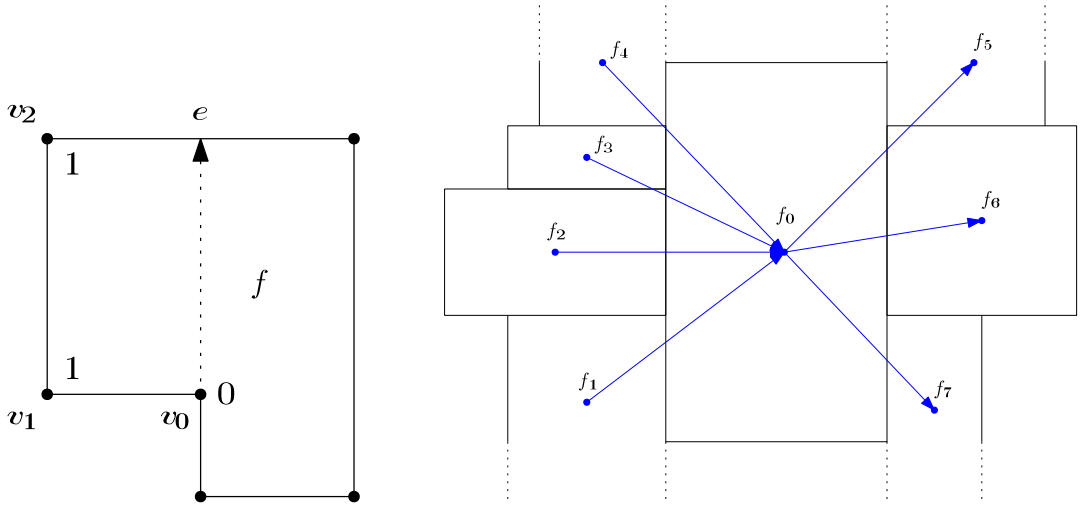
The Topology-Shapes-Metrics framework then uses a rectangle compaction algorithm first described by Hsueh [Hsu80]. Another description is found in [BETT98, Chapter 5.4]. We outline this technique only very roughly, since we will not be using it in detail throughout this thesis.

The algorithm works by computing two flow networks on the graph's dual: One for determining the x-, and one for determining the y-coordinates. The basic idea is that, for every rectangle in the layout, the lengths of its left and right (or top and bottom) sides must be equal, and also must be composed of (partial) lengths of its neighboring rectangles. Figure 3.5b shows an excerpt of the network used for determining the y-coordinates. This network uses only those edges of the dual graph which are dual to a vertical edge. Also, all edges (and thus all flow) is directed from left to right in this network. Finally, the flow on each edge is used as the length of the corresponding dual edge. It is easy to see that this results in the left and right sides of f_0 having equal lengths, and that these lengths are compatible with the heights of the neighboring rectangles. The network for determining the x-coordinates works accordingly. Assigning unit costs to all edges results in the lengths being chosen minimally.



(a) Flow on a Dual Edge (b) Result of One Unit of Flow (c) Result of Two Units of Flow

Figure 3.4.: Relationship between flow on dual edges and the resulting shape of the edges



(a) Rectangulation of a face

(b) A part of the flow network for determining y-coordinates. Black is the original graph with its rectangulated orthogonal representation, blue is the flow network on the dual graph.

Figure 3.5.: Illustrations of Rectangulations

Please note that this technique of finding coordinates is not optimal regarding the space the drawing is taking up (and probably many criteria more), although the compaction algorithm is. The resulting coordinates depend on how the faces of the original graph have been rectangulated, and rectangulating the graph as presented above is not guaranteed to be a good choice.

3.2. Introduction to Simulated Annealing

Parts of this thesis use the black-box metaheuristic *Simulated Annealing*, which can be used to approximate the global optimum of a function. This section shortly describes the ideas behind simulated annealing.

The annealing process that this heuristic is simulating is a technique for processing steel. Raw steel is heated and then slowly cooled again, which causes the steel molecules to form a very strong crystal lattice. From a physical point of view, this special crystal lattice is in a state of low energy (compared to other lattices), making it stable. Since this process minimizes the energy of the overall system by controlled cooling, simulated annealing often uses the terms *temperature* and *energy*.

When using simulated annealing to minimize an optimization problem, the energy usually refers to the objective function to be minimized. Each feasible solution has such an energy

associated with it that the optimal solution has the lowest energy. For minimization problems, the value of the objective function is usually used as energy. Simulated annealing is an iterative process. In each iteration, a new solution is generated from the current one. This solution is then accepted (i.e. made the current solution) if it has less energy, or, if it has more energy than the current solution, it is accepted with a probability based on the difference between the energies of the current and the new solution and the current temperature. A widely used probability function is (with T being the current temperature, and Δ being the difference in energies):

$$f(\Delta) = e^{-\Delta/T}$$

Clearly, the acceptance probability decreases with increasing Δ and decreasing T. With this, the simulated annealing process is able to pick solutions from all over the solution space at first (when T is still large). However, when the temperature decreases, it becomes gradually less likely that solutions that have a higher energy than the current solution are accepted.

For a successful application of simulated annealing, it is important to find a good *cooling schedule*. The cooling schedule determines the start temperature, as well as how fast the temperature is lowered and how many iterations are computed per temperature step.

In conclusion, simulated annealing can be described with Algorithm 3.1.

Algorithm 3.1: Simulated Annealing outline

```
   Input: energy function f
 1 solution = generate_random_solution();
 2 t = initial_temperature();
 3 repeat
 4 |   repeat
 5 |   |   new_solution = find_neighbour(solution);
 6 |   |   if accept(t, f(new_solution), f(solution)) then
 7 |   |   |   solution = new_solution;
 8 |   until enough steps on this temperature;
 9 |   t = cool_down(t);
10 until stop criterion;
```

While it is nontrivial to find a good cooling schedule for simulated annealing, there are schedules for which can be shown that the probability of the simulated annealing process finding the optimal solution converges against 1 with the number of iterations (cf. Aarts and Van Laarhoven [AVL85]). However, these schedules usually are not well-suited for practical applications.

4. Topology-Shape-Metrics in the Plane

In this chapter, we present an approach for drawing metro maps on concentric circles that works by first producing an orthogonal drawing, and then applying a transformation resulting in an ortho-radial drawing.

Minimizing the number of bends in the final drawing (see ($O2$) in Section 2.2) is a very important criterion according to Roberts [Rob12, p. 107]. We therefore adapt Tamassia's Topology-Shape-Metrics framework (explained in its basic form in Section 3.1), the primary objective of which is to minimize the number of bends in the layout.

In Section 4.1, we first describe some preprocessing steps as well as the transformation that is used to transform the orthogonal drawing into an ortho-radial drawing, since having this transformation in mind is important for some of the reasoning in the following sections. Section 4.2 then presents our version of Tamassia's *Shape* step, resulting in an orthogonal representation. In Section 4.3, we describe a way of assigning coordinates to vertices and bends, thereby presenting an improved version of Tamassia's *Metrics* step.

Section 4.4 and Section 4.5 show the details of two $\mathcal{NP}$-hardness proofs that we refer to from the preceding sections, and which make some of the presented heuristics necessary.

Finally, in Section 4.6 we present an implementation of the proposed solution and its variations and evaluate it experimentally and determine reasonable values for the parameters of our framework.

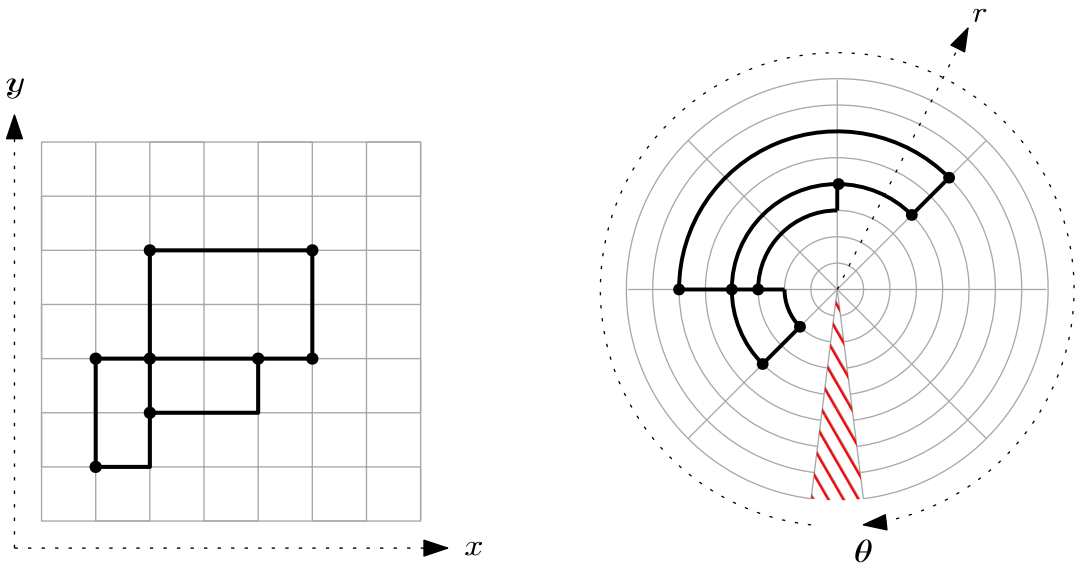
4.1. Preparations and Transformation

This section first shows the transformation we use to produce an ortho-radial drawing from an orthogonal drawing, since having seen the transformation is useful for understanding the techniques we propose below. We then introduce a number of preprocessing steps.

4.1.1. Transformation from Orthogonal to Ortho-radial

We now describe how to transform an orthogonal drawing into an ortho-radial drawing.[1] We assume the input layout to be given in cartesian coordinates, i.e. (x, y) pairs, and want the resulting layout to be specified in polar coordinates, i.e. (r, θ) pairs. Such a transformation is illustrated in Figure 4.1.

[1] For a formal definition of both drawing styles, refer to Definition 2.2 respectively Definition 2.3.



(a) A square grid with a graph drawn orthogonally

(b) A circular grid with the same graph, transformed into an ortho-radial drawing. The red striped area is the blank space introduced by ε.

Figure 4.1.: Example of a transformation from a orthogonal drawing into an ortho-radial drawing

Let P be the set of all points to be transformed. Then, for some small $\varepsilon > 0$ the transformation T is defined as:

$$T\colon P \subseteq (\mathbb{R}^+ \times \mathbb{R}^+) \to (\mathbb{R}^+ \times [0; 2\pi])$$

$$x_{\max} = \max_{(x,y)\in P} x + \varepsilon$$

$$T(x, y) = \left(y, \frac{2\pi x}{x_{\max}}\right)$$

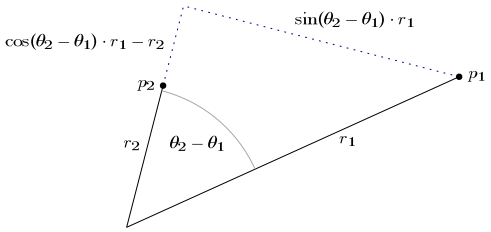
The projection T works by simply using the y-coordinate as the new radial coordinate, and linearly projecting the x-coordinate into the interval $[0; \frac{2\pi}{1+\varepsilon}]$ for the orbital coordinate.

We add ε to $x_{\max}$ so that after the transformation no points are located on the $0/2\pi$ boundary, because this could lead to crossings or overlaps. The blank space introduced by this is shown as a red striped area in Figure 4.1b. Note that this transformation maps a vertex pair with equal x coordinates to a vertex pair with equal θ coordinates. Because of this, vertical edges in the input layout become (straight) radial edges in the output. Similarly, horizontal edges in the input become orbital edges in the output. Thus, if the input is a valid orthogonal drawing, the output is a valid ortho-radial drawing.

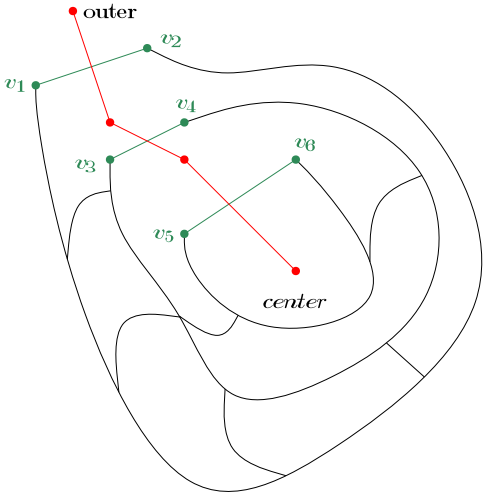
This transformation has two immediate consequences that are relevant for our approach: First, let $p_1 = (r_1, \theta_1)$ and $p_2 = (r_2, \theta_2)$ be two points, and without loss of generality assume that $r_1 \geq r_2$. Then, their distance (as illustrated in Figure 4.2a) is:

$$\operatorname{dist}(p_1, p_2) = \sqrt{(\sin(\theta_2 - \theta_1) \cdot r_1)^2 + (\cos(\theta_2 - \theta_1) \cdot r_1 - r_2)^2}$$

The larger radius r_1 thus is a positive factor for the distance. Consequently, enforcing a minimum distance of points in the ortho-radial case requires that in the reverse-transformed



(a) Distance in Polar Coordinates



(b) An example of how to transform G into G_{cut}. Black is the original G, with green being the parts that are cut out. Red is the shortest path in the dual graph.

Figure 4.2.

orthogonal drawing, lower points (with smaller y-coordinate) must maintain a larger minimum distance than points with larger y-coordinate.

The second consequence of this simple transformation is that we do not utilize the circular nature of the θ coordinate, since that coordinate is determined by a linear projection from the (not circular) x dimension of the orthogonal drawing. For example, if we have two points $p_1 = (r, \theta_1)$ and $p_2 = (r, \theta_2)$ representing two vertices, and assume $\theta_1 > \theta_2$, these two vertices can not have an edge going clockwise from p_1 to p_2. We address this problem in Section 4.1.2.

4.1.2. Closing the Gap

We now try to mitigate the problem that no edges can be routed crossing the 0 / 2π boundary for a drawing derived from the transformation presented in Section 4.1.1. A consequence of this problem is that this approach cannot draw a graph such that any cycle in the graph has the point $S = (0, 0)$ (the center of the ortho-radial coordinate system) on one side and the outer face on the other side. However, being able to draw cycles in such a way is highly desirable to fully exploit the possibilities offered by an ortho-radial drawing style. Thus, we now present a technique that enables us to compute drawings with cycles separating the center of the drawing from the outer face.

If we are given an embedding of the graph G (which we assume throughout this thesis), and a face f in which S should be positioned (called the *center face*), we can determine which cycles would have to cross the 0 / 2π boundary: These are exactly all cycles separating the outer from the center face. Knowing this, we are able to transform G into a graph G_{cut}, in which all these cycles have been cut open. If we now lay out G_{cut} geometrically in such a way that the cut-open parts are laid out on the boundary of the drawing, we are able to close the cycles again across the boundary.

To identify the edges to be cut (i.e. removed from the graph), we perform a simple shortest path search in the dual of G, starting with the vertex representing the center face, and ending in the vertex representing the outer face. For every dual edge that is part of this shortest path, the corresponding edge is part of the cut, see Figure 4.2b. Clearly, this cuts

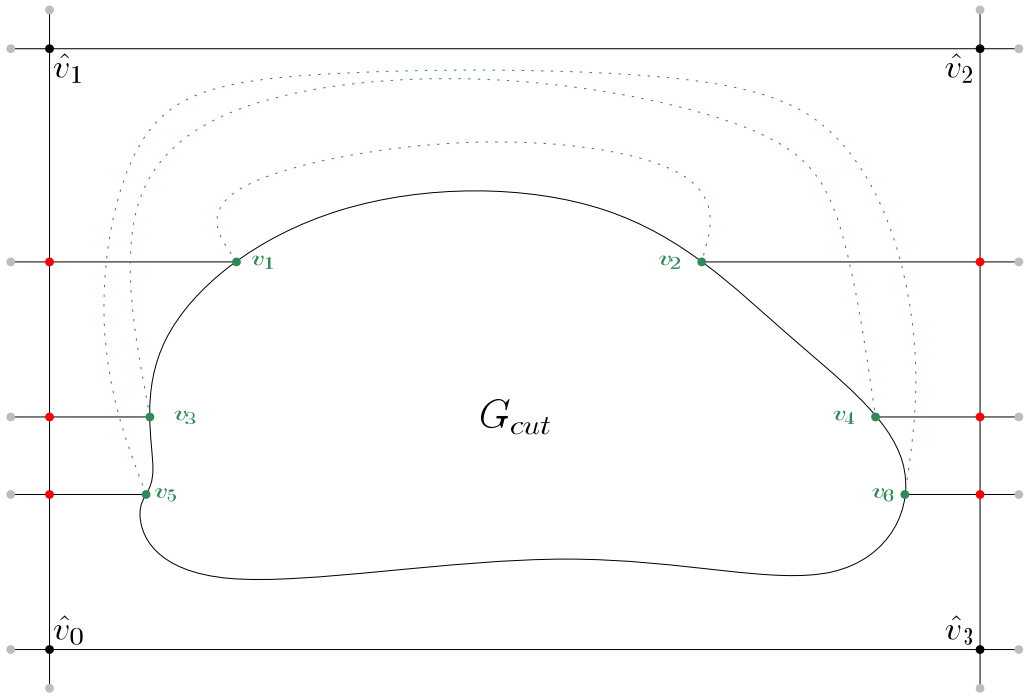


Figure 4.3.: Assembly of the frame around G_{cut}. Gray elements are dummies to enforce proper layout. Red vertices are ports, while green vertices v_1 - v_6 are endpoints of edges that have been cut, and correspond to their counterparts in Figure 4.2b. The cut edges are shown as dotted green lines, but only for illustration. These are not parts of G_{cut} or G''.

all cycles separating the center from the outer face, and also is minimal among all such cuts. Let the cut edges be $\tilde{E} = (\tilde{e}_1, \tilde{e}_2, \tilde{e}_3, \dots)$. We assume the $\tilde{e}_i$ to be ordered from inner to outer, i.e. $\tilde{e}_1$ is the edge that corresponds to the first dual edge in the shortest path, $\tilde{e}_2$ is the edge that corresponds to the second dual edge, etc. Call the new graph $G' = (V, E \setminus \tilde{E})$.

We now need to make sure that in the final embedding, we can again close the cycles that we cut open. For this, we modify the input graph G' of Tamassia's flow network (cf. Section 3.1.2) and also the flow network itself. First, observe that in G', the endpoints of all edges in $\tilde{E}$ lie on the outer face of G'.

Now we put G' in a rectangular frame made out of four vertices $\hat{v}_0, \hat{v}_1, \hat{v}_2, \hat{v}_3$, see Figure 4.3. We also add dummy vertices around each of these four vertices so that we can control how the four 90° angles around the vertices are distributed - these are the unnamed gray vertices in Figure 4.3. To make sure that the endpoints of edges in $\tilde{E}$ (just called *endpoints* for the rest of this section, shown in green in Figure 4.3) end up on the left and right boundary of the drawing, respectively, we now connect them to the left respectively right boundary of the frame, by adding one vertex per endpoint on the frame (call them the *ports*, shown in red in Figure 4.3) and connecting these to the endpoints in the correct order. Again, dummy vertices enforce the correct layout around the inserted port vertices by making sure that these vertices have degree 4, and thus leaving no choice for the embedding of their neighbors around them. Call this final graph G''.

We finally need to make sure that the frame around G' actually stays rectangular, i.e. has no edge-bends. To achieve this, consider the dual edges of all edges that form the frame. Setting the capacity of these dual edges in the flow network to 0 results in no edge-bends being assigned to the edges of the frame, forcing the frame to stay a rectangle.

(a) The situation before e_1 and e_2 were merged

(b) The situation after e_2 was merged into e_1

Figure 4.4.: Example of merging two edges to reduce vertex degree

Formally, the flow network given in Section 3.1.2 is modified as follows:

- the network Graph G_f is $G'' = (V'', E'')$,
- the upper bound for the flow per edge, u, is:

$$u(e) = u(v,w) = \begin{cases} \infty & \text{if } e \in E \\ \infty & \text{if } \{v, \cdot\} \in \tilde{E} \vee \{w, \cdot\} \in \tilde{E} \\ 0 & \text{if } e \text{ is part of the frame} \end{cases}$$

It is easy to see that this modified flow network is always solvable. Clearly, any valid layout of the graph induces a feasible solution to the flow network by assigning flows based on the actual shape of the edges and the distribution of angles around the vertices. Thus, every layout that keeps the edges making up the frame straight induces a feasible solution. If, however, a valid layout for G exists,[2] then G', from which only some edges have been removed, can be laid out, too. Ignoring the edges connecting G' to the frame for a moment, this valid layout can then be embedded in the space inside the frame. Since all endpoints appear on the outer boundary of G' and keep their order in the layout, we can easily route the edges connecting these endpoints to the port vertices on the frame.

Closing the Cycles

Since we forced the frame to be laid out as a rectangle, after the transformation described in Section 4.1.1, the left and right borders of the frame are assigned polar coordinates with angles of 0 and $\frac{2\pi}{1+\varepsilon}$, respectively. Note that as depicted in Figure 4.1b, the area between the orbital coordinates $\frac{2\pi}{1+\varepsilon}$ and 2π is empty. We can use this area to route the edges that we have cut from the original graph.

4.1.3. Limiting the Degree to 4

The drawing style used throughout this thesis allows for only four edges per vertex. The input graphs derived from metro networks however do usually include vertices with larger degrees. When reducing the degree of a vertex, we cannot just remove edges. This would leave metro lines disconnected, and the map would be unusable. What we do instead is this: We merge two (or more) edges of a vertex, and split them up again a short distance away from the vertex. Formally, merging and splitting of two edges e_1 and e_2 is done by

[2] Which we may assume, since our input graph G is required to be planar, and as Tamassia [Tam87] shows, a valid layout is found for every planar graph with this technique.

inserting a new vertex in one of them, and then re-hanging the other edge to that vertex. Consider the example in Figure 4.4: Figure 4.4a illustrates the situation before anything was merged. We decided to merge e_1 and e_2. Figure 4.4b shows how e_2 was merged into e_1.

We must now show how to select the edges for merging. It would be desirable to make that choice in such a way that the best possible drawing of the resulting graph has a minimal number of edge-bends. However, we can show this problem to be $\mathcal{NP}$-hard in Section 4.5. Therefore, we propose a simple way of limiting the degree. This technique does not have any guarantee as to the number of edge-bends introduced by the degree limitation.

Aside from edge-bends, it is reasonable to assume that edges carrying too many metro lines create visual disturbances, especially at points where these metro lines split or merge. Therefore, we propose the following steps to transform a vertex of degree larger than 4 into a vertex of degree 4: Let v be a vertex with $\deg(v) > 4$. For all consecutive pairs e_1, e_2 of edges incident to v, compute the union of the set of metro lines running via e_1 and the set of metro lines running via e_2. Merge that pair e_1, e_2 for which the cardinality of that union is minimal. Repeat this until the degree of v is 4.

This way, the number of metro lines per edge is minimized.

4.2. Shape - Computing an Orthogonal Representation

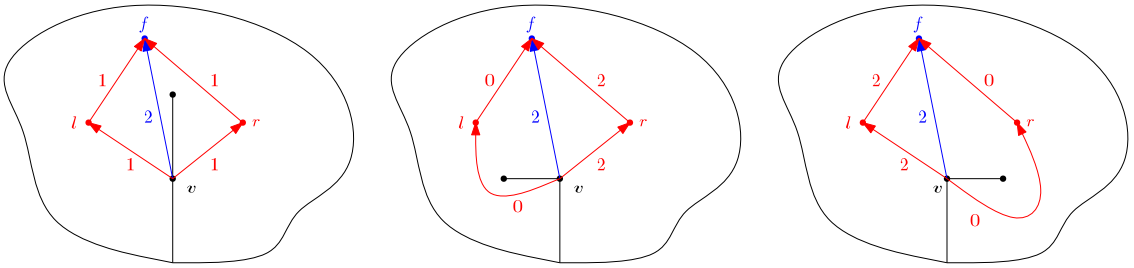
This section describes how we compute an orthogonal representation, i.e. perform the *Shape* step of Tamassia's framework. We summarize the original version of the algorithm in Section 3.1.1.

We first present a modification that minimizes the number of metro lines bending at metro stations, which is something that Tamassia's framework cannot do, since it has no notion of metro lines. Then we show another modification that changes the framework's optimization from minimizing the number of bent edges to minimizing the number of bent metro lines, considering that an edge of the graph can support more than one metro line.

4.2.1. Avoiding Bends at Stations

As stated (and motivated) in Section 2.2 as quality criterion (O3), we want to avoid the bending of metro lines at stations. We do so by smartly arranging the incidences of vertices around the vertices. To this end, we modify Tamassia's flow network, respectively the already modified version we presented in the previous sections. We have to distinguish between vertices of different degrees. For vertices of degree 1 or 4, we have only one way of distributing the four 90° angles around the vertices. Thus we only examine vertices of degree 2 or 3 here.

Please note that in the following, we treat a vertex of degree d to have d incident faces. This is obviously not always true, and in fact, this is an element of Tamassia's flow network that is under-specified: Take a vertex of degree 2 that has only one incident face. Obviously, this vertex gives its two surplus units of flow to the only face to which it is incident. In such a case, Tamassia's flow network does not specify what the distribution of the two 90° angles around the vertex should look like. See the three different possibilities in Figure 4.5 for an example: For every one of the three different results, the flow in Tamassia's flow network is the same. Tamassia probably left this out because any of the three possible distributions $(180°/180°, 90°/270°, 270°/90°)$ yield a valid orthogonal representation. However, in the following, this makes a difference. In such cases, we just introduce two proxy vertices for the vertex representing the face in the flow network, see the red parts of Figure 4.5: Now, flow flowing from v to f via r can be distinguished from flow via l, and the corresponding angles can unambiguously be assigned. For vertices of



(a) 180°/180° assignment of angles (b) 90°/270° assignment of angles (c) 270°/90° assignment of angles

Figure 4.5.: Different possibilities of assigning angles around a degree-two vertex v. Blue parts are parts of Tamassia's original flow network, red parts are parts of our modification. Numbers show the flow on the edges.

degree 3, we use an analogue transformation, in which we need to insert three dummy vertices instead of two.

We now show how to minimize the number of lines bending at stations for vertices of degree 2 or 3.

Vertices of Degree 2

For a vertex v of degree 2, the question of what distribution of the two edges around the vertex is optimal is easy to answer. Let e_1 and e_2 be the edges incident to v. Let p be a path in G. In the following, we write $(e_i, e_j) \sqsubset p$ if and only if $p = (\dots, e_i, e_j, \dots)$ or $p = (\dots, e_j, e_i, \dots)$, thus if p contains e_i and e_j consecutively.

If there is at least one line passing through v, thus a path p exists in the set[3] L with $(e_1, e_2) \sqsubset p$, then both faces incident to v should be assigned an 180° angle at v so that the lines can pass straight through v.

In Tamassia's flow network, this would correspond to v giving one unit of flow to those two vertices that represent the faces incident to v. Let the two incident faces of v be represented by f_1 and f_2 in the flow network. If we set the cost of the lines passing through v bending at v to c, we modify the flow network around v as seen in Figure 4.6: The dummy vertices d_1 and d_2 are added and connected between v and f_1 and f_2, respectively. The edges used to do so (e_5, e_6, e_7 and e_8) are all assigned capacity 1 and cost 0. Edges e_3 and e_4, which were originally responsible for flow from v to the faces, are assigned infinite capacity and a cost of c, with c being the price to pay for the lines bending at v. When later determining the distribution of angles around v, the flow on e_5 is considered as if it was flow on e_3, while the flow on e_6 is considered as if it was flow on e_4.

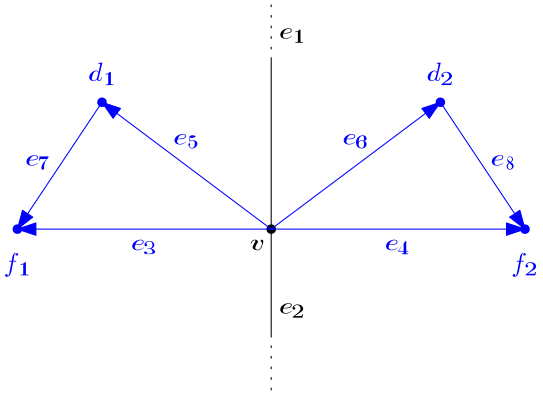


Figure 4.6.: Modification to avoid station-bends for a degree-two vertex. Black parts are parts of G, while blue parts are parts of the resulting flow network.

[3] L is a set of paths on G, specifying the individual metro lines. See Section 2.2 for the formal definition of L.

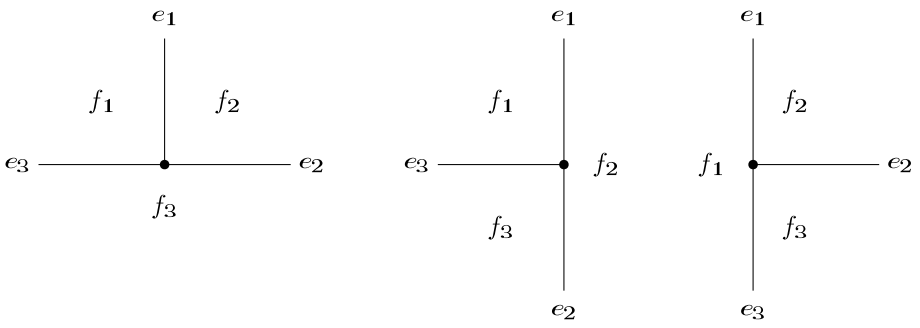


Figure 4.7.: Possible ways of arranging edges around a vertex of degree 3

This way, the desired situation, namely v giving one unit of flow to each of its incident faces, can be achieved with a cost of 0, since one unit of flow can pass without cost via d_1 and d_2, respectively. If, however, the flow network made the lines passing through v bend in v, this would mean that either f_1 or f_2 receives two units of flow, forcing one unit of flow to use either e_3 or e_4 and thus incurring a cost of c.

It remains to be said how to chose c. It is plausible that a station-bend involving lots of lines is worse than a station-bend involving only few lines. Thus, we set c to the number of lines bending at v times the station-bend optimization factor ω_{station} (see Definition 2.4):

$$c = \omega_{\text{station}} \cdot |\{p \in L \mid p \sqsupset (e_1, e_2)\}|$$

Vertices of Degree 3

Vertices of degree 3 are handled very similarly to vertices of degree two. If v has degree 3 (with the three edges e_1, e_2, e_3), it has to assign one additional 90° angle to one of its three incident faces, resulting in three possible choices as illustrated in Figure 4.7. For each choice, one pair of the three edges is laid out in a straight line. In each of the cases, it is easy to compute how many lines are bent at v and thus derive a badness in accordance with the definition above. Assume that e_2 and e_3 are the two straight edges (as seen leftmost in Figure 4.7). Then, the number of bent lines is $|\{p \in L \mid p \sqsupset (e_1, e_2)\}| + |\{p \in L \mid p \sqsupset (e_1, e_3)\}|$. Since this situation is the result of v giving one unit of flow to f_3, we use this badness together with the constant ω_{station} as cost c_3 to be assigned to the edge leading from v to f_3. Similarly, we can define costs c_i for all edges leading from v to f_i:

$$\begin{aligned}
c_1 &= \omega_{\text{station}} \cdot (|\{p \in L \mid p \sqsupset (e_2, e_1)\}| + |\{p \in L \mid p \sqsupset (e_2, e_3)\}|) \\
c_2 &= \omega_{\text{station}} \cdot (|\{p \in L \mid p \sqsupset (e_3, e_2)\}| + |\{p \in L \mid p \sqsupset (e_3, e_1)\}|) \\
c_3 &= \omega_{\text{station}} \cdot (|\{p \in L \mid p \sqsupset (e_1, e_2)\}| + |\{p \in L \mid p \sqsupset (e_1, e_3)\}|)
\end{aligned}$$

This way, we can set the cost incurred to the flow network to be the number of lines bending at stations weighted by the desired factor. We later determine good values for ω_{station} experimentally in Section 4.6.

4.2.2. Minimizing the Number of Bent Metro Lines

We now introduce a further modification, which can be applied to Tamassia's Topology-Shape-Metrics framework in addition to the adaption shown in the previous section. The aim of this modification is to minimize the number of times a metro line is bent on an edge. For an illustration, refer back to Figure 2.2b: Here, bending edge e_1 only induces

one line-bend, while bending e_2 would cause two metro lines to bend. This corresponds to the quality criterion (O2a) introduced in Section 2.2.

The only change necessary is a simple change to the edge costs in the constructed flow network: As described in Section 3.1.2 and in Definition 3.3, in the flow network, the cost of every edge in G's dual graph are 1 per flow unit. Flow on edges of G's dual represents the bends on the edges of G's orthogonal representation. For details, refer back to Section 3.1.2. Thus, every edge-bend incurs a cost of 1 in the flow network.

We first define a function assigning to every edge the number of metro lines running via that edge. Let the set L of paths on G be defined as in Section 2.2, with every path in L representing one metro line.

Definition 4.1 (Line-Count function)**.** *Given a graph $G = (V, E)$ and a set L of paths on G. Then, the* line-count function *is*

$$\delta \colon E \to \mathbb{N}$$

with

$$\delta(e) = |\{p \mid p \in L \wedge e \in p\}|$$

We now change the definition of c in the flow network:

Definition 4.2 (Min-Cost-Flow to compute an Orthogonal Representation minimizing Line-Bends)**.** *Given a graph $G = (V, E)$ with an embedding, a set L of paths on G, and δ as defined in Definition 4.1. Let $G' = (V', E')$ be the dual of G. Let a flow network be defined as in Definition 3.3.*

Given the cost function c, let the new cost function be c', with

$$c'(e) \;=\; \begin{cases} c(e) + \omega_{line}\delta(e) & \text{if } e \in E' \\ c(e) & \text{otherwise} \end{cases}$$

With this definition, the cost incurred by flow on an edge of G's dual, i.e. flow resulting in an edge being bent in the orthogonal representation, incurs cost that include the number of metro lines being bent weighted by the appropriate factor ω_{line}. We later practically determine good values for ω_{line}; see Section 4.6.3.1.

4.3. Metrics - Assigning Coordinates

After an orthogonal representation has been computed, actual coordinates must be assigned to the vertices. In the original Topology-Shape-Metrics framework, Tamassia proposes the technique shown in Section 3.1.3. However, we can prove the problem of assigning coordinates to be $\mathcal{NP}$-hard (see Section 4.4), thus there can be no efficient optimal solution algorithm.[4] In fact, Tamassia's solution in a way does no optimization at all: In the following we present a space of possible solutions, from which Tamassia's technique just randomly picks one.

We present a way of exploring this solution space, and use this possibility to apply the metaheuristic simulated annealing to the problem. In doing so, we not only demonstrate a compaction technique that improves on Tamassia's technique in terms of area minimization, but it also enables us to optimize the drawing for multiple criteria.

We use the black-box metaheuristic simulated annealing for our approach, which has been roughly outlined in Section 3.2. To do so, we need to specify three components:

[4] Under the assumption that $\mathcal{P} \neq \mathcal{NP}$.

1. A cooling schedule
2. An energy function, i.e. an objective function on the graph layout
3. A way to randomly perturb the found solution

Here, we do not go into detail about the cooling schedule, since this choice is not (or at least not necessarily) problem-specific. In practice (see Section 4.6), we are using an off-the-shelf solution that includes a reasonable cooling schedule.

We do also not specify a fixed energy function here, but rather assume that a useful energy function should reflect the objectives stated in Section 2.2. We evaluate different energy functions practically in Section 4.6. Please note however that, since here we only are optimizing the final step of assigning coordinates to the computed orthogonal representation, not all criteria can reasonably be part of the energy function: For example, the number of bends is fixed once the orthogonal representation has been computed, thus including it in the energy function would not be useful.

More interesting is the question of how to randomly generate a new solution from an existing one.

Reaugmentation

We first define what, in the context of this optimization problem, we call an *augmentation* of an orthogonal representation of the graph $G = (V, E)$.

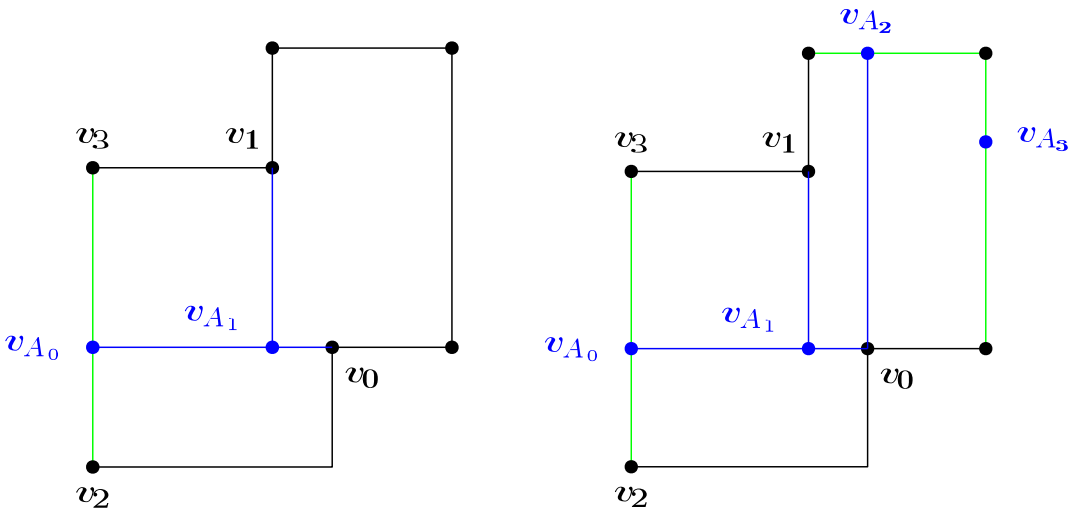
Definition 4.3 (Augmentation)**.** *Given a graph $G = (V, E)$ together with an orthogonal representation of G, an* augmentation *consists of:*

- *A set E_A of edges added to the orthogonal representation*
- *A function $o_A \colon (u, v) \in E_A \to \{1, 2, 3, 4\}$ assigning an orientation to every added edge, where 1 represents "up", 2 represents "right", 3 represents "down" and 4 represents "left".*
- *A set $V_A \subseteq (E_A \cup E)$ of inserted vertices subdividing edges in E or edges in E_A. Since this may look peculiar, note that we are just identifying the inserted vertices with the original edges that they subdivided. Also, since every vertex in V_A subdivides an edge, thereby removing and replacing it with two new edges, there must be at least two edges in E_A incident to each vertex in V_A.*

An augmentation is *valid* if, after applying all subdivisions in V_A and inserting all edges in E_A with the orientation given by o_A, every face of the augmented orthogonal representation is a rectangle. Also, the result must of course still be a valid planar layout.

Strictly speaking, for the above definition of o_A to be useful, the edges of the orthogonal representation must be oriented, too, which they are not. But since their relative orientations are fixed, arbitrarily orienting one edge induces an orientation on all edges. Also, since this arbitrary choice only rotates the end result by multiples of 90°, the quality of the solution is not impeded by this choice. Also, for o_A to be valid, clearly $o_A(u, v) \equiv o_A(v, u) + 2 \mod 4$ must hold.

Figure 4.8a shows a simple example of this. Here, E_A consists of the blue and green edges, with the difference being that green edges are replacing original edges that have been subdivided, while blue edges are inserted to slice the faces. V_A consists of v_{A_0} and v_{A_1}, with v_{A_0} being identified with the edge $\{v_2, v_3\}$ and v_{A_1} being identified with the edge $\{v_0, v_{A_0}\}$.



(a) A simple augmentation example. Blue edges and vertices are added in the augmentation. Green edges are replacing an original edge that was subdivided.

(b) An example for an augmentation that is not minimal

Figure 4.8.: Examples of augmentations

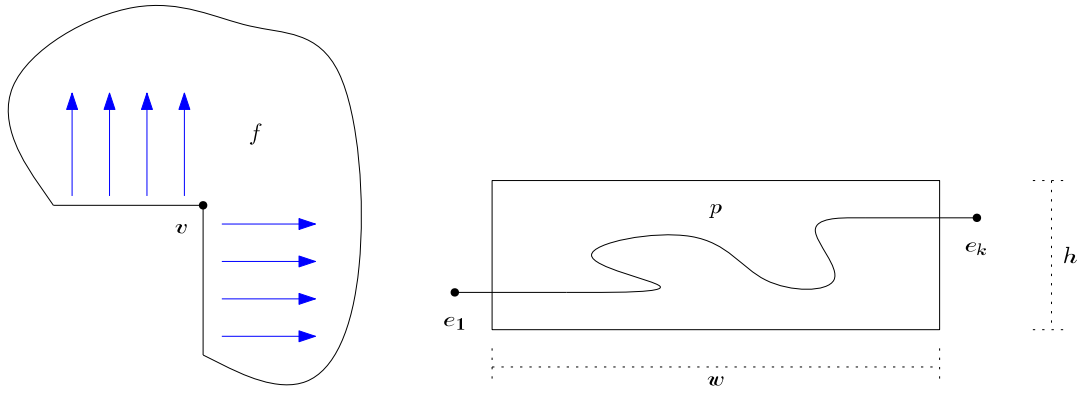
Also, we are only considering minimal augmentations here. We call an augmentation *minimal* if we cannot remove any element from E_A or V_A without either misrepresenting the original graph (because we deleted an edge that is part of a subdivided original edge) or making the resulting layout non-rectangulated. Figure 4.8b shows an example of a non-minimal augmentation. Removing v_{A_3} (and reinserting the original edge) does no harm. Also, the edge between v_0 and v_{A_2} is unnecessary, resulting in v_{A_2} also becoming obsolete.

From the description in Section 3.1.3 it becomes immediately clear that the quality of the compaction algorithm depends (only) on our choice of how to augment the graph until every face is a rectangle, since the subsequent rectangle compaction algorithm is optimal.

While Tamassia proposes a way of augmenting the orthogonal representation, his suggestion is by far not the only possibility. The metaheuristic that we want to apply later needs a way of exploring the solution space, so we present a way of producing any possible valid augmentation of the orthogonal representation that can be achieved by iteratively slicing non-rectangular faces. With this, it is easy to produce a new augmentation of arbitrary similarity to an existing augmentation: Given an augmented orthogonal representation, select a set of edges that were introduced during the augmentation, and delete them (and the vertices introduced by these edges). This results in some of the faces not being rectangular anymore. Then, randomly select one of the possibilities of augmenting these faces again. The number of edges deleted determines how similar the final augmentation is to the original augmentation.

We therefore present a more general method of rectangulating faces: Instead of connecting any outward bend to the next suitable edge (or more correctly, a vertex subdividing that edge), we find a list of all possible edges for doing so and randomly select one of them.

First, we modify the problem a little: In the original form, for every vertex v with an outward bend, we are looking for edges to which we can connect that vertex with an axis-aligned line segment. Instead, we now consider the two edges incident to v on the face that the outward bend is on. Call them e_a and e_b. We now look for edges that can be drawn in such a way that we can connect these edges to either e_a or e_b with an axis-aligned line



(a) An outward bend on a face and arrows indicating all edges reachable from it

(b) A straight path

Figure 4.9.: Illustrations for proof of Lemma 4.5

segment. See Figure 4.9a for an example. Here, f has an outward bend at v. Interesting are all edges of f that can be reached by one of the blue arrows, i.e. could be connected to one of the two edges incident to v by a straight horizontal respective vertical line segment. These edges can then also be reached from v.

We therefore characterize which edges can be laid out in such a way relative to each other. First, we need the notion of a *straight path*:[5]

Definition 4.4 (Straight Path)**.** *A path $p = (e_1, \ldots, e_k)$ of length k in $G = (V, E)$ is a* straight path*, if the last edge in p, e_k, has rotation 0 on the path p, i.e.* $\mathrm{rot}_p(e_k) = 0$.

The above definition basically states that the path *unwinds* any bends it makes before coming to an end.

In the following, we make a series of statements as to how such straight paths can be drawn in an orthogonal drawing. To make matters easier, during the proofs we present we sometimes handle edges as if there were no bends on the edges, i.e. as if all bends on the path happened *between* the edges, not *on* the edges. While this of course is not the case in general orthogonal representations, for the purpose of the following proofs, we can in fact treat an edge with bends as a series of multiple bend-free edges that have been assigned the correct orientations in the orthogonal representation. This does not change any results throughout the next lemmas, since they only concern themselves with how paths can be drawn, and in this undertaking it is negligible whether we draw a series of straight edges or a single edge with multiple bends. The important consequence of this point of view, which we use multiple times, is that the rotation of two consecutive edges on the path can only differ by at most 1.

We start by proving the following lemma:

Lemma 4.5. *Given a graph $G = (V, E)$, a straight path p of length k in G, and two real numbers w and h, we can find an orthogonal drawing of G so that the subpath $(e_2, \ldots, e_{k-1})$ is laid out inside a box B of dimensions $w \times h$, and e_1 and e_k cross opposite borders of B.*

You can find an example of such a layout in Figure 4.9b. Note that in this lemma, we explicitly do not state that we are able to compute an orthogonal *grid* drawing of the desired dimensions. To see that this would obviously not always be possible, just consider an arbitrary path and $w = 0.5$ and $h = 0.5$ (assuming a grid size of 1.0).

[5] For the definition of a rotation, refer to Section 2.3.1.

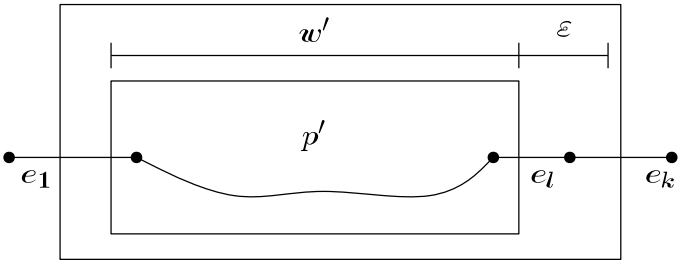


Figure 4.10.: Inductive construction of a straight path in a box in the case that the second to last edge already has rot = 0

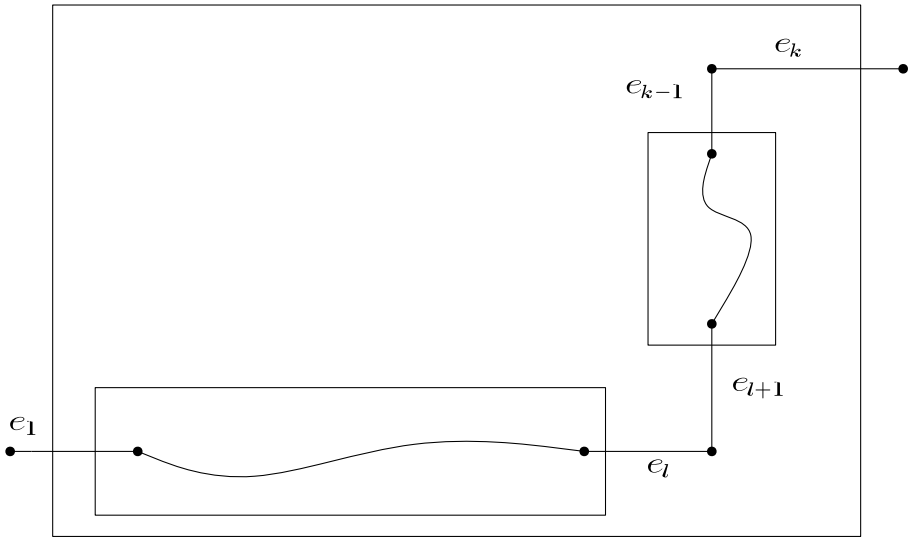


Figure 4.11.: General case of constructing a straight path inside a box

Proof. The fact that p can be laid out in a box of given dimensions can be shown by induction over the path length k:

For $k = 1$, the path consists of just an edge, which can trivially be laid out as required. Now assume that we have shown the lemma for all straight paths of length $k - 1$ and have a straight path of length k. Let e_l be the last edge[6]n p before e_k with $\mathrm{rot}_p(e_l) = 0$. Note that, since $\mathrm{rot}_p(e_1) = 0$ by definition, we can always find such an edge. In the case that $l = k - 1$, i.e. the edge directly before e_k already has rotation 0, this part of the proof is easy: We draw the path $p' = (e_1, \dots e_l)$, which is a path of length $k - 1$, inside a box of dimensions $w' \leq w - \varepsilon$ and $h' \leq h$, which we can as per induction hypothesis. We can then just connect e_k to e_l and are done. See Figure 4.10 for an illustration.

If $l < k - 1$, then the edge e_{l+1} must either have a rotation of 1 or -1, since $\mathrm{rot}_p(e_l) = 0$ and the rotations of two consecutive edges can only differ by at most 1. Since both cases are symmetric, without loss of generality we only consider the case $\mathrm{rot}_p(e_{l+1}) = 1$ here. From this follows $\mathrm{rot}_p(e_{k-1}) = 1$: The rotation of e_{k-1} can obviously only be -1 or 1. Since the rotation does not jump along the path, but can only ever increase or decrease by 1, $\mathrm{rot}_p(e_{k-1}) = -1$ would mean that there is an edge between e_{l+1} and e_{k-1} with a rotation of 0, which cannot be by the choice of e_l. Thus, $\mathrm{rot}_p(e_{k-1}) = 1$.

With this, the subpath $(e_{l+1}, \dots, e_{k-1})$ becomes a straight path itself, with length of at most $k - 1$. Thus, by assumption, we can lay it out in a box of arbitrary size. The final construction can be seen in Figure 4.11: Since we can determine arbitrary sizes for the two

[6] i

inner boxes (and the edges connecting them), we can also realize arbitrary sizes for the outer box. □

Now that we know this about straight paths, we return to our original problem: Which edges can be drawn in such a way that they can be connected to a given edge e via a straight line segment orthogonal to both edges? We can answer this question with the following lemma:

Lemma 4.6. *Given a graph $G = (V, E)$ and an edge e_1 on a face f, if a path $p = (e_1, \ldots, e_k)$ with $\mathrm{rot}_p(e_k) = 2$ exists on f, then an orthogonal layout of G can be found so that e_1 and e_k can be connected by a line segment orthogonal to e_1 and e_k, and which splits f into two faces.*

Proof. Let p be a path such as above of length k. Since $\mathrm{rot}_p(e_1) = 0$ and $\mathrm{rot}_p(e_k) = 2$, there must be a first edge with rotation of 1. Let that edge be e_i. The same holds for a rotation of 2: Let e_j be the first edge with rotation of 2.

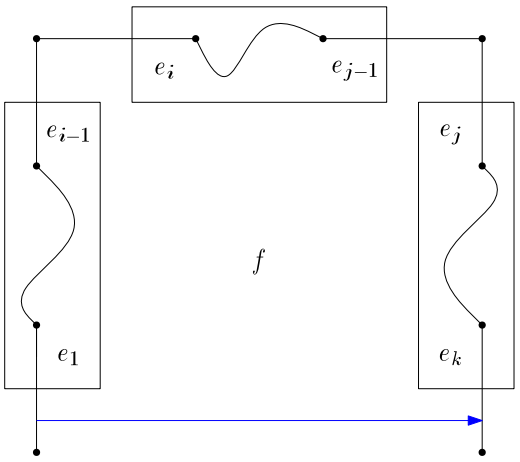
With these definitions, the paths $p_1 = (e_1, \ldots, e_{i-1})$, $p_2 = (e_i, \ldots, e_{j-1})$ and $p_3 = (e_j, \ldots, e_k)$ are straight paths as required in Lemma 4.5. Since we can lay them out in boxes of arbitrary sizes, we can then piece them together as depicted in Figure 4.12a.

Now, since we can assign arbitrary heights to the two vertical boxes (and the edges e_1 and e_k), we can make sure that e_1 and e_k overlap vertically. Let the path p be called the *upper half* of f. Since p is completely contained in the three boxes in this layout, no parts of it can obstruct the space between e_1 and e_k. Let the rest of f (a path p' from e_k to e_1) be called the *lower half* of f. We must also show that no part of this lower half obstructs the space between e_1 and e_k. However, since every face must have a rotation of 4, and p had a rotation of 2, p' must also have a rotation of 2. Thus, the same argument holds for p'. In total, a layout as illustrated in Figure 4.12b results. In conclusion, no part of the face f can obstruct the space between e_1 and e_k in such a layout. □

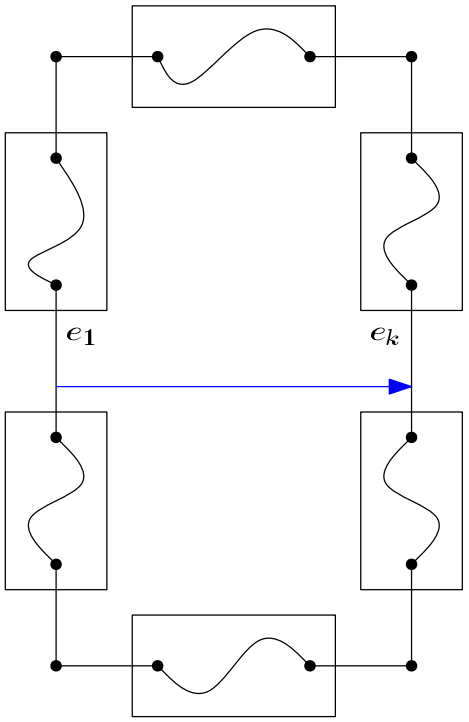
We have now shown that all edges with a rotation of 2 relative to an edge e are candidates for being drawn opposite to it, and thus candidates for an edge needed to rectangulate the face at an endpoint of e. Now we show that these edges are in fact the only possible edges for doing so.

Lemma 4.7. *Given a graph $G = (V, E)$ and an edge e_1 on a face f, if the path $p = (e_1, \ldots, e_k)$ that runs clockwise around f from e_1 to e_k results in $\mathrm{rot}_p(e_k) \neq 2$, then no orthogonal layout of G can be found so that e_1 and e_k can be connected by a line segment orthogonal to e_1 and e_k, splitting f into two faces.*

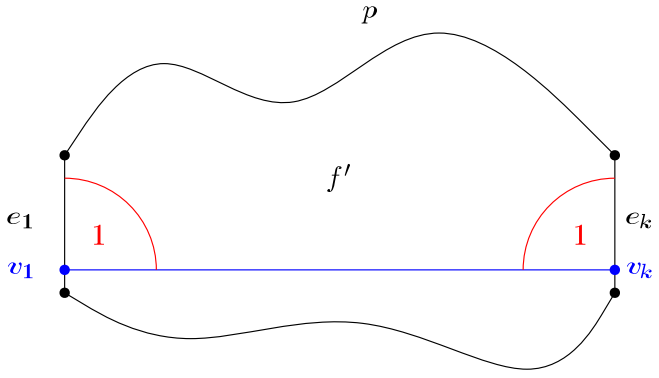
Proof. Assume p exists as required above, and also assume that it is possible to lay out G orthogonally so that e_1 and e_k can be connected by a line orthogonal to e_1 and e_k. This situation is illustrated in Figure 4.12c, in which the blue vertices v_1 and v_k and the blue edge $\{v_1, v_k\}$ are inserted to connect e_1 to e_k. This splits f into two parts, the upper part of which includes p. Call that part f' as in the figure. Now, f' consists of the edges $(e_1, e_2, \ldots, e_k = e_k, \{v_k, v_1\})$. As illustrated in red in Figure 4.12c, the part $(e_k, \{v_k, v_1\}, e_1)$ has rotation of 2. Since we know by assumption that p, which forms the rest of f', does not have a rotation of 2, the total rotation of f' cannot be 4. This contradicts the requirement that every face in an orthogonal drawing must have a rotation of 4. Thus, p cannot have anything but a rotation of 2 if such a connection is possible. □



(a) Piecing together three straight paths to make one path of rotation 2. The blue arrow shows the visibility between the first and the last edge.

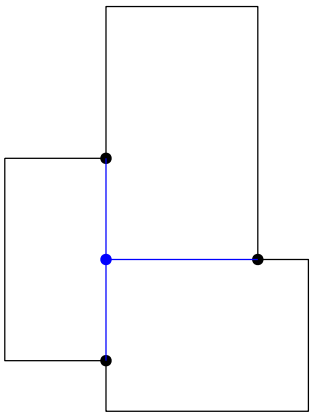


(b) The complete construction allowing e_1 and e_k to be drawn opposite each other. The blue arrow shows the visibility between e_1 and e_k.

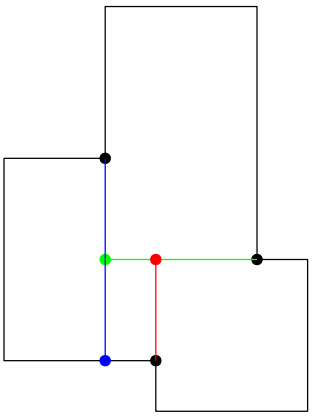


(c) Face f after connecting e_1 to e_k. Blue parts are inserted to facilitate the connection. Red parts indicate the added rotation.

Figure 4.12.: Decomposition of a face into straight paths so that edges can be laid out opposite each other



(a) A rectangulation that is never produced by the proposed algorithm. Blue parts are inserted for rectangulation.



(b) A rectangulation of the same orthogonal representation that can be produced and results in the same or better objective. Blue is inserted first, then green, then red.

Figure 4.13.: Illustrations of rectangulations

By combining Lemma 4.6 with Lemma 4.7, we have shown that every possible way of connecting an outward bend in a face to an edge on that face can be computed by enumerating all edges that have a rotation of 2 relative to the two edges participating in that outward bend.

Wrapping this section up, we propose a metaheuristic approach for solving the coordinate assignment problem: The different possible augmentations of the input orthogonal representation are the solution space in which to find a solution that best fulfills certain (to be specified) objectives such as required space. We have a simple way of generating new solutions close to a given solution within that solution space by deleting edges used for augmentation and then again randomly choosing an augmentation as described above. This is all we need to apply simulated annealing.

One final remark: Initially, we stated that with the presented method it is possible to create all augmentations that can be achieved by iteratively slicing not yet rectangular faces. Clearly, this method does not produce *all* possible augmentations respective rectangulations: See Figure 4.13a for an example of a rectangulation that is never produced by repeatedly slicing faces. However, the method we propose is very general regarding the results: Consider Figure 4.13b for a rectangulation that can be produced by our method. Here, the blue cut was introduced first, then green, and then red. Now, if the compaction algorithm is adapted slightly to remove the minimum distance around vertices introduced for rectangulation (i.e. everything not black in Figure 4.13b), then the compaction algorithm on this graph yields the same result as on the rectangulation in Figure 4.13a by moving the red and green vertices as well as the blue and its neighboring black vertex on top of each other.

4.4. $\mathcal{NP}$-hardness of Coordinate Assignment

After an orthogonal representation has been computed, coordinates must be assigned. In the original Topology-Shape-Metrics framework, Tamassia solves this by computing two flow networks, as presented in Section 3.1.3. However, this approach is not only not optimal in terms of space usage by the resulting drawing, but also does not exactly reflect the requirements in our case: We do not necessarily need integer coordinates along the x-Axis (which becomes the θ-axis once we transformed into polar coordinates, cf. Section 4.1.1). Tamassia uses integer coordinates to enforce a certain minimum distance between vertices,

but in our case, this minimum distance is dynamic: For reasons explained in Section 4.1.1, we need a larger minimum distance (in terms of polar coordinates) the closer we get to the center, i.e. with decreasing y (or r after transformation).

We illustrate this with Figure 4.14: Here, v_1 has the same orbital coordinate as v_3, and v_2 has the same orbital coordinate as v_4. However, since the radius of v_1 and v_2 is smaller than the radius of v_3 and v_4, no vertex can be placed between v_1 and v_2, while a vertex can fit between v_3 and v_4. Thus, in terms of the orbital coordinates, we need a dynamic notion of a minimum distance.

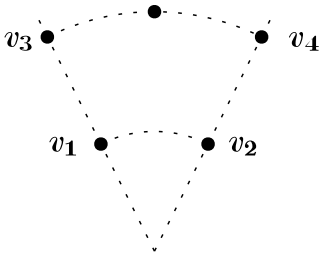


Figure 4.14.: Illustration of different minimum distances

While Tamassia did not make a statement as to the computational complexity of the problem of assigning coordinates, Patrignani [Pat99] proves $\mathcal{NP}$-hardness not only of a compaction that has optimal area usage, but also for minimizing the total or maximal edge length.

However, our problem is slightly different from an ordinary compaction of an orthogonal representation, since we do not have to adhere to a fixed minimum distance in the θ-coordinate, i.e. we are not limited to a fixed grid, as explained above. On the other hand, our θ-coordinate is limited to the interval $[0, 2\pi)$, since all coordinates have to be drawn on a circle.

We now prove that this problem is still $\mathcal{NP}$-hard. Following the naming scheme in [Pat99], we formalize the problem to be solved as follows. Note that we work directly in polar coordinates.

Definition 4.8 (Dynamic Distance Orthogonal Area Compaction)**.** *Given are*

- *a graph $G = (V, E)$ together with an orthogonal representation of it, in which all edges are oriented*
- *a function $d\colon \mathbb{N} \to \mathbb{R}$ assigning a minimal orbital distance to every radial coordinate with $r > r' \Rightarrow d(r) < d(r')$*
- *an integer K*

Find two functions

$$f_r\colon V \to \mathbb{N}$$

$$f_\theta\colon V \to [0, 2\pi)$$

such that:

(1) treating $f_V(v) = (f_r(x), f_\theta(v))$ as the polar coordinate[7] of v, a planar ortho-radial drawing results that adheres to the given orthogonal representation

(2) for every v with $f_r(v) = k$, the length of the minimal circular arc between v and any other vertex with the same radius is at least $d(k)$

(3) $\forall v \in V : f_r(v) < K$

Note that the additional requirement that an orientation must be given together with the orthogonal representation removes the ambiguity of having four possible rotations of the whole drawing. In fact, the following proof holds without this requirement, but becomes more clear if we may assume fixed directions for the edges used in our construction.

[7] For the definition of f_V, see Definition 2.3 in Section 2.1.

Also, in our case, the function d is as simple as $d(r) = c/r$, with c being some constant. This results in the minimum distances between two vertices on the same radius (measured along the circular segment connecting both) to be the same at every radius.

Theorem 4.9. Dynamic Distance Orthogonal Area Compaction *is $\mathcal{NP}$-hard. This remains even if d is chosen as $d(r) = c/r$ for some constant c.*

First note that the restriction of d (to the case that is interesting when drawing in an ortho-radial style) is important here: If the choice of d if not restricted, one can choose $d(r) = 1$, making the problem equivalent to an ordinary orthogonal grid compaction, which has been shown to be $\mathcal{NP}$-hard in [Pat99].

We reduce from 3-SAT, which is stated as follows by Garey and Johnson [GJ79, p. 259]:

Definition 4.10 (3-SAT)**.** *Given are an instance consisting of a set U of variables and a collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$. Is there a satisfying truth assignment for C?*

To show $\mathcal{NP}$-hardness of Dynamic Distance Orthogonal Area Compaction, we must transform any given 3-SAT instance into a graph G (together with an orthogonal representation), an integer K and a function $d(r) = c/r$, such that if and only if G can be laid out with the radius of every vertex at most K, the given 3-SAT instance is satisfiable.

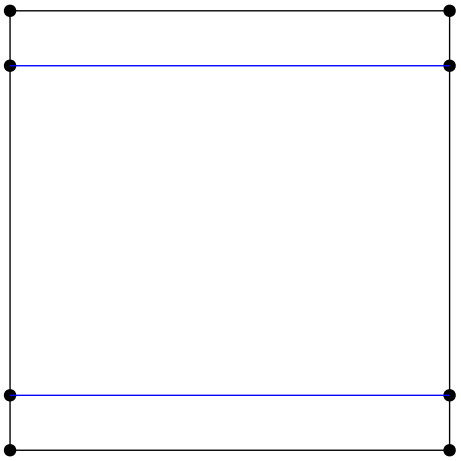
The main task in doing so is constructing the graph G and its orthogonal representation, which we show in detail here. We do not explicitly state how the orthogonal representation is made up - in every step of the construction, the orientation of the edges should be clear. Given an instance of 3-SAT, consisting of n variables and m clauses, we construct an instance of Dynamic Distance Orthogonal Area Compaction in three steps: First, we build a frame inside of which all other parts of the graph are placed. We then construct gadgets for the variables x_i, forming columns inside the frame. Last, we construct gadgets for the occurrence of a variable in a clause, which form rows in the frame, and thus cross the columns forming the variables. At these crossings, the usage of a variable in a clause is encoded. Three of these occurrence gadgets then make up one clause gadget.

In the following, all our illustrations are drawn orthogonally, although we are working with polar coordinates and an ortho-radial drawing style. Here, the x-coordinate of our illustrations represents the θ-coordinate of the actual ortho-radial drawings, and the y-coordinate represents the r-coordinate. We do this because the gadgets used in the proof largely rely on rectangular shapes, and these are easier to see in an orthogonal drawing style.

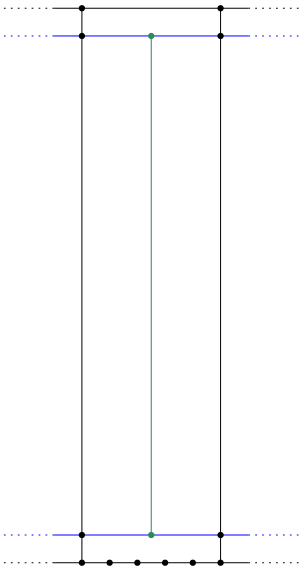
Note also that while in the following we specify widths and heights, measured in vertices, for most of our gadgets, and enforce *minimum* widths and height by inserting enough vertices, the gadgets could of course be drawn with larger widths and height, i.e. be extended. We want to prevent this, and thus we make sure that no gadgets can extend without in turn extending the frame. We then adjust K so that a drawing height of K can only be achieved by embedding the bottom of the frame as low as possible, thus resulting in an infeasible solution if the frame was extended by one of the gadgets contained within.

4.4.1. Frame and Variable Gadgets

The frame is constructed as seen in Figure 4.15a: It is simply a box of vertices plus two edges. The two edges shown in blue each span the whole box at the very top and the very bottom of it horizontally. We call these two edges the upper respective lower *rail*. The variable gadgets are connected to these rails as well as to the upper and lower borders of



(a) The frame around the graph generated for the proof of Theorem 4.9



(b) The variable gadget for the proof of Theorem 4.9. The blue edges are the rails from Figure 4.15a. The green edge is the *switch edge* of this gadget.

Figure 4.15.: Frame and variable gadget for the proof of Theorem 4.9

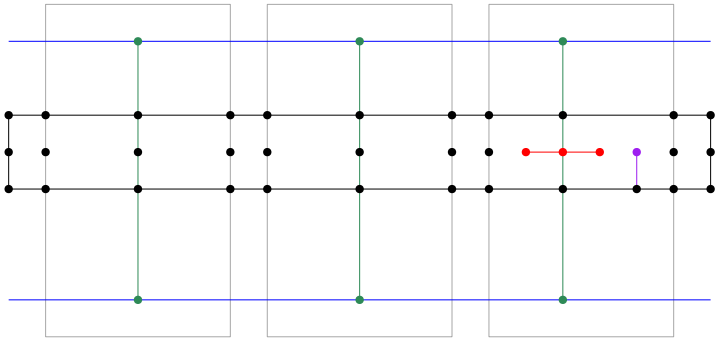
the frame. A variable gadget is itself a box, with an edge inside the box running from top to bottom, called the *switch edge* of the gadget. The variable gadget is connected to both rails such that the upper rail connects just below the top of the variable gadget, the lower rail connects above the bottom of the variable gadget. The top (respective bottom) of the variable gadget lies on the top (respective bottom) of the frame. The switch edge is then just connected to the two rails, so that it can slide left or right inside the variable gadget. See Figure 4.15b for an illustration.

4.4.2. Occurrence and Clause Gadgets

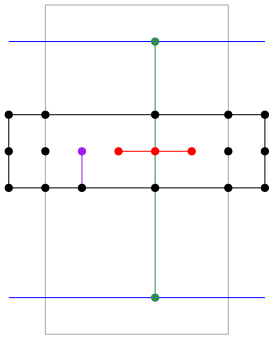
The gadget representing an occurrence of a variable in a clause is another box, this time spanning the width of the frame. Every occurrence gadget crosses every variable gadget. See Figure 4.16a for an example. Here, an occurrence of the variable x_3 is shown. Note the red and purple parts that are added inside the crossing area of the gadget for x_3 and the occurrence gadget: One vertex is attached to each side of the switch edge, shown in red, and another vertex is attached to the inside of the occurrence gadget, shown in purple. In this case, that additional purple vertex is attached right of the switch edge, representing a positive occurrence, i.e. the positive literal of x_3 occurring in the clause that the occurrence gadget belongs to. A negative occurrence is represented by attaching that purple vertex to the left of the switch edge, as illustrated in Figure 4.16b.

We say that an occurrence gadget and the variable gadget of this occurrence *intersect*. Note that while an occurrence gadgets *crosses all* variable gadgets, it *intersect* with only *one* variable gadget.

The idea is to encode the truth value of a variable in the position of the switch edge. If the switch edge is drawn as far to the left as possible inside the variable gadget (as seen in x_3 in Figure 4.16a), this represents the assignment of *true* to the variable, if the edge is drawn as far to the right as possible, this represents the assignment of *false*. We later also make sure that the horizontal space inside the variable gadget is never (for any radius)



(a) An occurrence gadget (shown in black) crossing three variable gadgets (shown in gray / green). The variables x_1 and x_2 do not participate in the occurrence, x_3 does.



(b) A negative occurrence of x_4

Figure 4.16.: Occurrence gadgets

large enough such that five vertices could be embedded within it horizontally next to each other.[8]

With this, we can show that for a single variable gadget, the switch edge cannot be positioned such that occurrence gadgets representing a negative occurrence of the respective variable as well as gadgets representing a positive occurrence of the same variable can be laid out without requiring an additional row, i.e. with both gadgets having a height of 3. This situation is depicted in Figure 4.17a: The occurrence gadget drawn at radius r_1 represents a positive occurrence, while the gadget drawn at r_2 represents a negative occurrence. The switch edge is placed in the middle such that both gadgets can be drawn with a height of 3. Consider the distance d_1: Clearly, $d_1 \geq 3 \cdot d(r_2)$ must hold. However, as stated above, we assume that a variable gadget is never large enough for five vertices to be laid out horizontally next to each other inside it, i.e. $\forall r\colon d_{total} < 6 \cdot d(r)$. From this follows $d_2 < 3 \cdot d(r_2)$ and thus $d_2 < d_1$. However, this means that $d_3 > d_4$ must also hold. Since $d_4 > 3 \cdot d(r_1)$ is clearly required, that would result in $d_3 + d_4 > 6 \cdot d(r_1)$, which may not be as per our assumption.

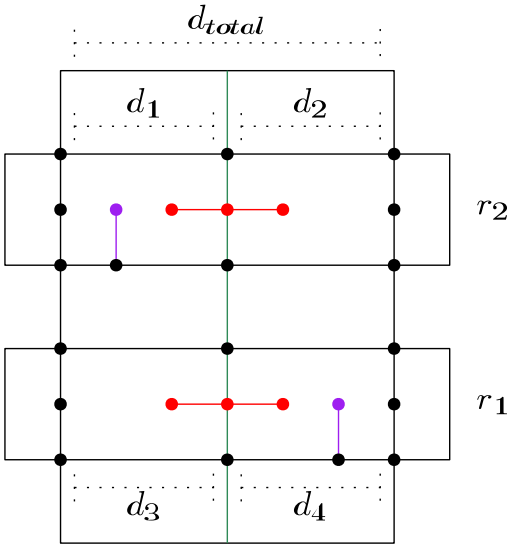
Thus, if our assumption holds, only a drawing as illustrated in Figure 4.17b is possible: One of either positive or negative occurrences must be drawn with at least height 4.

We now demonstrate the clause gadget. It is simply a frame of minimum height 13 (enforced by stacking vertices on its left and right borders), and containing three occurrence gadgets, as depicted in Figure 4.18. Now, if at most two of the occurrence gadgets have to be laid out using a height of 4, the clause gadget can still be drawn with height 13. If, however, all three occurrence gadgets need height 4 to be drawn, then the clause gadget needs at least height 14 to be drawn.

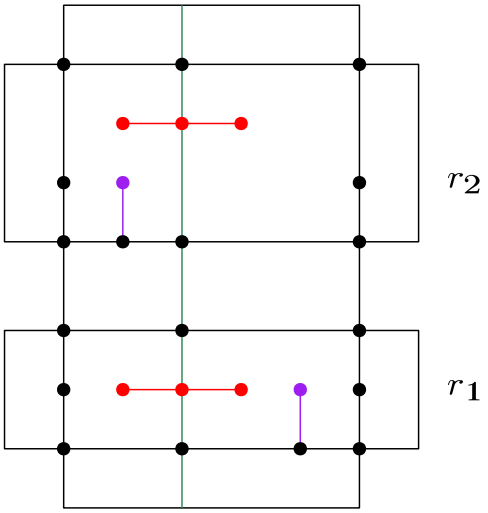
4.4.3. Conclusion

We now show how to combine the gadgets and how to do so in a way that enforces the assumption made above, namely that inside every variable gadget, there is never enough space to lay out five vertices horizontally next to each other. Figure 4.19 shows the complete graph resulting from a 3-SAT instance with three variables and two clauses. The insides of the occurrence gadgets are not shown for clarity reasons. We are now interested in the minimum width and height needed to draw this graph. We first consider the level indicated

[8] Keep in mind that while the gadget might have the same width at every radius, the minimum distance between two vertices decreases with increasing radius.



(a) A situation that cannot arise: A variable gadget is laid out in such a way that both a positive and a negative occurrence gadget can intersect with it without requiring an additional row



(b) A valid layout of a variable gadget with two contrary occurrences

Figure 4.17.: An impossible and a possible drawing of a variable gadget with two contrary occurrences

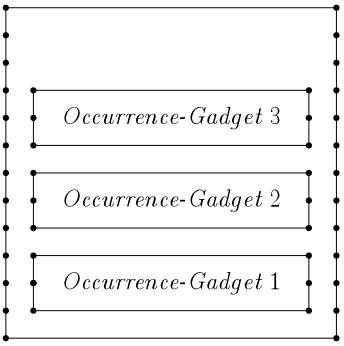


Figure 4.18.: The Clause Gadget containing three Occurrence Gadgets

by the red arrow in Figure 4.19, where the variable gadgets are all connected to each other and are forced to a minimum width of six vertices. As can be seen, on the left and right side of the frame, there are three vertices in this level that do not belong to a variable gadget, indicated by w_{fixed}. The number of vertices in the middle, w_{dyn} is $6n$.

Ignoring the additional brown vertices for now, assume that this level is drawn with radius r_{base}. Then, for all vertices to be embeddable, it must hold:

$$\begin{aligned} & (w_{dyn} + 2w_{fixed} - 1)d(r_{\text{base}}) \leq 2\pi \\ \Leftrightarrow\ & (4n+5)\frac{c}{r_{\text{base}}} \leq 2\pi \\ \Leftrightarrow\ & \frac{(4n+5)c}{2\pi} \leq r_{\text{base}} \end{aligned} \tag{4.1}$$

Unsurprisingly, a larger number of vertices causes a larger r_{base} to be needed to uphold the minimum distance constraint. Thus, by adding w_{add} additional vertices to that level (the brown vertices in Figure 4.19), we can enforce an even larger r_{base}:

$$\frac{(4n+5+w_{add})c}{2\pi} \leq r_{\text{base}} \tag{4.2}$$

We now take a look at the minimum height needed to draw the frame. As can be seen in Figure 4.19, at the top and bottom, there are two rows each independent of the clause gadgets. The two rows needed to draw these make up the height h_{fixed}. Then, we assume that every clause gadget can be drawn with its minimum height of 13, as shown above. Thus, for the height h_{dyn} holds: $h_{dyn} = 13m$, and the total height is thus $2h_{fixed} + h_{dyn} = 13m + 4$.

We now must make sure what we assumed above: That the horizontal space inside a variable gadget is never large enough for five vertices to be embedded next to each other. We first make sure that in every valid solution, the bottom of the frame must really be embedded at radius r_{base} (and not with larger radius) by setting:

$$K = r_{\text{base}} + 2h_{fixed} + h_{dyn} \tag{4.3}$$

Here, r_{base} is the minimum value satisfying inequation 4.2. We now know that at the bottom of the frame, every variable gadget has exactly enough room for four vertices to be placed horizontally next to each other. Consider Figure 4.20: The width of a variable gadget is thus:

$$w = 5d(r_{\text{base}}) = \frac{5c}{r_{\text{base}}}$$

We must thus make sure that at any r_2 still overlapping with the frame (thus $r_2 \leq K$ for any valid solution), it holds that $w < 6d(r_2)$. Thus:

$$\begin{aligned} & w < 6d(r_2) \\ \Leftrightarrow\ & w < \frac{6c}{r_2} \\ \Leftrightarrow\ & \frac{5c}{r_{\text{base}}} < \frac{6c}{r_2} \\ \Leftrightarrow\ & \frac{5}{6}r_2 < r_{\text{base}} \end{aligned} \tag{4.4}$$

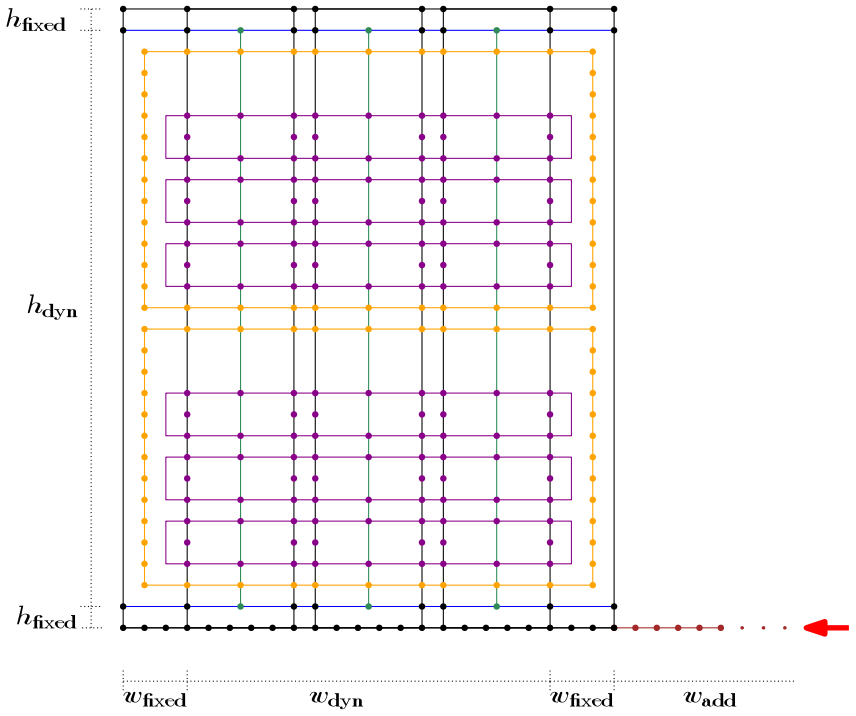


Figure 4.19.: The complete assembled graph with three variables and two clauses. Insides of the occurrence gadgets are not shown. Occurrence gadgets are marked magenta, clause gadgets orange. The brown vertices are added to achieve a certain minimum width.

But since we know that $r_2 < K$ must hold in a valid solution, we can give another lower bound for r_{base}:

$$\begin{aligned} & \frac{5}{6}K && < r_{\text{base}} \\ \Leftrightarrow\ & \frac{5}{6}(r_{\text{base}} + 2h_{fixed} + h_{dyn}) && < r_{\text{base}} \\ \Leftrightarrow\ & 5(2h_{fixed} + h_{dyn}) && < r_{\text{base}} \\ \Leftrightarrow\ & 65m + 20 && < r_{\text{base}} \end{aligned} \tag{4.5}$$

Inequation 4.5 gives us the minimum r_{base} for which the assumption holds. As we have shown above, we can use the w_{add} additional vertices illustrated in Figure 4.19 to force r_{base} to become arbitrarily large. We can thus construct G such that the assumption holds.

Finally, we have to formally show how a valid instance of 3-SAT is being mapped to a valid Dynamic Distance Orthogonal Area Compaction instance, and vice versa. We also have to show that the transformation can be done in polynomial time.

3-SAT → Dynamic Distance Orthogonal Area Compaction

Given a valid 3-SAT instance, it is always possible to draw the graph G (and its associated orthogonal representation) constructed above ortho-radially with maximum radius K: Given a satisfying assignment of the variables x_i, draw the switch edge inside every variable gadget as far to the left as possible if the respective variable is assigned *true*, and as far to the right as possible if the variable is assigned *false*. Since we assume a valid 3-SAT instance, every clause gadget now must contain at least one occurrence gadget that is

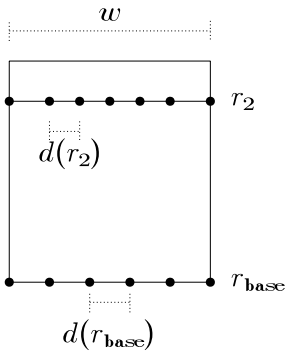


Figure 4.20.: Calculating the width of a variable gadget

satisfied and can be drawn with height 3, as shown in Section 4.4.2. The other two occurrence gadgets may need a height of 4, but then, every clause gadget can still be drawn with height 13. Since K was defined such that the height of the frame is exactly the constant height plus $13m$, a drawing of maximum height K can be achieved.

Dynamic Distance Orthogonal Area Compaction → 3-SAT

Given is a valid drawing of G (and its associated orthogonal representation) with maximum radius K. Then, by the definition of K, every clause gadget can have a maximum height of 13. To achieve this, at least one occurrence gadget per clause gadget must have maximum height 3. For any variable not intersecting[9] with any occurrence gadget, the assignment of a truth value does not matter, since it is not part of any clause, thus we assign *true*. For any variable gadget that intersects only with occurrence gadgets that are drawn with a height of more than 3, assign *true* arbitrarily. For any variable gadget intersecting with at least one occurrence gadget that is drawn with height 3, assign the truth value represented by that occurrence gadget. Now, every clause is satisfied, since at least one of its literals was represented by a occurrence gadget of height 3, and the respective variable was assigned the truth value needed to draw that occurrence gadget with height 3, which is the truth value also satisfying the literal of the clause.

Time Complexity of the Transformation

First, the number of vertices and edges required to form the cage is a constant plus the number of vertices required for w_{add} as illustrated in Figure 4.19. By inequations 4.4 and 4.5, that number is limited by a polynomial. The vertices and edges required for a variable gadget (without any clause or occurrence gadgets) are constant, so the total number of vertices and edges required for all variable gadgets is linear in the number of variables. The number of vertices and edges required to insert a clause gadget and its occurrence gadgets is a constant plus the number of vertices necessary where the occurrence and clause gadgets cross the variable gadgets. However, the number of vertices needed for a single crossing is constant. Thus, the total number of vertices and edges required for all crossings is linear in the product of the number of clauses and the number of variables. Finally, the structure of every gadget and the composition of all gadgets is fixed and thus can be computed in linear time. Therefore, the transformation can be computed in polynomial time.

Finally, from the proof of $\mathcal{NP}$-hardness directly follows this corollary:

Corollary 4.11. Dynamic Distance Orthogonal Area Compaction *is $\mathcal{NP}$-hard in the strong sense.*

[9] Keep in mind the definition of *intersect* from Section 4.4.2: A variable gadget and an occurrence gadget only *intersect* if the variable gadget represents the variable taking part in the occurrence.

Proof. To show $\mathcal{NP}$-hardness in the strong sense, it is sufficient to show that during the transformation from 3-SAT, all numbers in the created Dynamic Distance Orthogonal Area Compaction instance are bounded by some polynomial. The only number appearing in an instance is K, which is determined in equation 4.3 as $K = r_{\text{base}} + 2h_{fixed} + h_{dyn}$. Certainly, h_{dyn} and h_{fixed} are bounded by the number of vertices, which is polynomial. The minimum value we can use for r_{base} is determined in inequation 4.5, which is also polynomial in the size of the input instance. Thus, all numbers in the transformed instance are polynomially bounded, and Dynamic Distance Orthogonal Area Compaction is $\mathcal{NP}$-hard in the strong sense. □

4.5. $\mathcal{NP}$-hardness of Degree Limitation

Since real metro networks have stations that are directly connected to more than four other stations, resulting in a vertex with a degree of more than four, and since an orthogonal (or ortho-radial) layout can never have a vertex of degree more than four, we need a way to handle these stations. Here, we present some plausible rules for doing this, and show that the problem of finding a solution that adheres to these rules and results in a drawing with a minimum number of bends is $\mathcal{NP}$-hard and even $\mathcal{APX}$-hard.

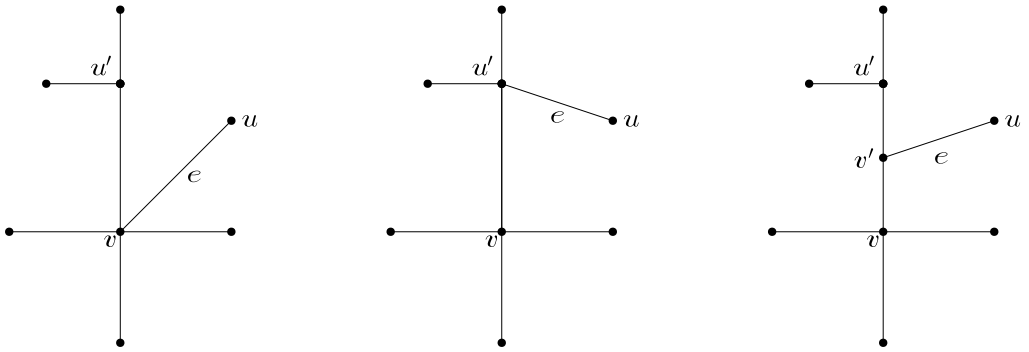
4.5.1. Overview

Generally, the problem of having vertices with a degree of more than four can only be solved by removing edges from these vertices, as already discussed in Section 4.1.3. However, since edges are used to represent metro lines, we may not just omit them. We must therefore move these edges to other, possibly new, vertices. Consider the example in Figure 4.21a, where v has degree 5, and we want to rehang e away from v. Figure 4.21b depicts the situation after e was rehung to the existing vertex u'. Now, every vertex has at most degree 4. However, any metro line running from v to u must now be routed via u', resulting in a drawing that suggests that these metro lines run from v via u' to u, instead of running from v to u directly. Thus, moving edges to an existing vertex results in drawings misrepresenting the metro network we want to draw. Now, consider Figure 4.21c, where the edge was moved to a new vertex inserted into the edge (v, u'). Now, the metro lines running from v to u can be routed without touching any other vertices representing stations. Therefore, we only allow this method of moving edges. We formalize this in the following rule:

Definition 4.12 (Edge-Moving Rule)**.** *Given a graph $G = (V, E)$, a vertex v of at least degree 5, an edge $e = (v, u)$ incident to v and an orientation, which is either* clockwise *or* counterclockwise*, the* Edge-Moving Rule *is:*
Let $e' = (v, u')$ be the edge that is clockwise (respectively counterclockwise, depending on the selected orientation) of e at v, add a new vertex v' to G. Then, replace e' with the edges (v, v') and (v', u'). Finally, replace e with the edge (v', u).

The process of transforming G into a graph where all vertices have degree 4 or less then consists of repeatedly selecting a vertex of degree 5 or more, selecting an edge incident to this vertex, selecting an orientation and finally applying the rule laid out above.

We add one more detail: Additionally to the graph, the overall process takes as input a function $b\colon V \to \{X \mid X \in \mathcal{P}(E) \wedge |X| \leq 4\}$, specifying for every vertex a list of at most four edges (which must be incident to that vertex). The edges in $b(v)$ are treated as not being eligible for being moved away from v. This function can be used to model important incidences, for example for edges carrying a lot of metro lines, and which should not be removed from their vertices. For an example, consider the map excerpt in Figure 4.22,



(a) Example of a vertex of degree 5, with an edge that should be removed

(b) Example of moving an edge to an existing vertex

(c) Example of moving an edge to a new vertex

Figure 4.21.: Two possibilities to lower the degree of v.

taken from a metro map of the London Subway. The station King's Cross St. Pancras has a total of seven adjacent stations: Euston, Euston Square, Russell Square, Farringdon, Angel, Highbury & Islington and Caledonian Road. However, these adjacencies carry different numbers of metro lines each: While there are three lines running to Euston Square and Farringdon, there are only two lines running to Euston, and only one line to all other adjacent stations. Now, if the adjacency between Kings's Cross St. Pancras and Euston Square was to be moved away from King's Cross St. Pancras, and was to be moved in clockwise order, a new vertex would be inserted into the edge between King's Cross St. Pancras and Euston, and a total of five metro lines would be routed over this edge, then to be split up before being routed to Euston and Euston Square, respectively. This could lead to an impaired readability of the metro map at the point where that many metro lines split. Thus, it probably would be preferable to have edges carrying only one metro line moved away from King's cross St. Pancras. To achieve this, the edges between King's Cross St. Pancras and Euston respectively Euston Square could be added to $b(v)$ (with v being the vertex representing King's cross St. Pancras).

While limiting the graph to a degree of 4, we want to minimize the number of bends in a bend-optimal drawing of the resulting graph. We formalize the problem as follows:

Definition 4.13 (MAXDEGREE-4)**.** *Given a graph $G = (V, E)$, a function $b\colon V \to \{X \mid X \in \mathcal{P}(E) \wedge |X| \leq 4\}$, and an integer K, is it possible to transform G into G' such that every vertex in G' has at most degree* 4*, the number of bends in a bend-minimal orthogonal drawing of G' is at most K, by doing the following repeatedly:*

1. *Select a vertex v of degree at least* 5
2. *Select an edge e incident to v, such that $e \notin b(v)$*
3. *Select a direction, either* clockwise *or* counterclockwise
4. *Apply the rule from Definition 4.12*

The operations left unspecified by this approach are how to select the next vertex, how to select an edge for removal, and how to select whether to move that edge clockwise or counterclockwise.

We now show that making these choices such that the minimum number of bends in an orthogonal layout of the resulting graph is minimized is $\mathcal{NP}$-hard. We do so by reducing from PLANAR 3-SAT:

Figure 4.22.: Excerpt from a hexalinear metro map of London's metro by Maxwell J. Roberts. The station King's Cross St. Pancras has may adjacent stations.

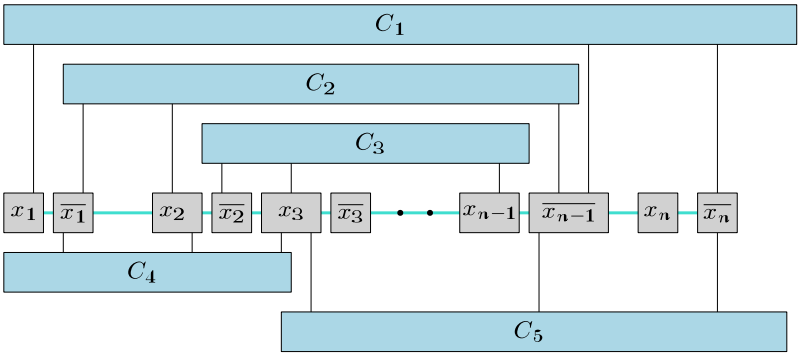


Figure 4.23.: Example drawing of a PLANAR 3-SAT instance

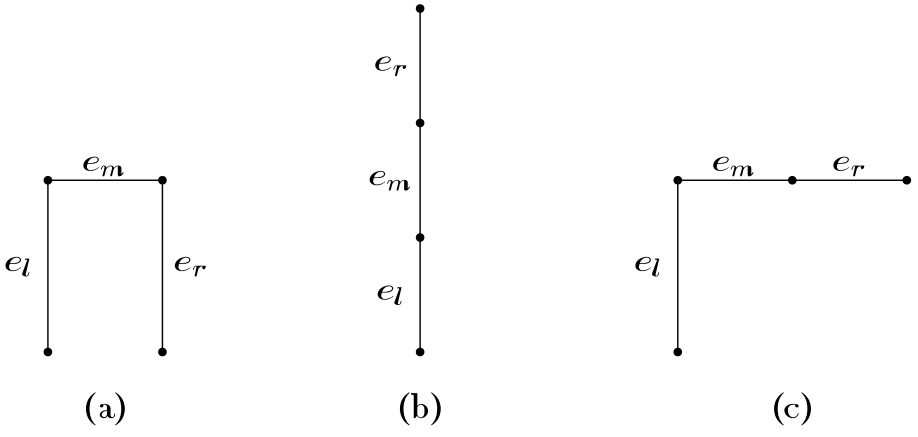


Figure 4.24.: Different possible bends of paths of length 3. The gadget allows possibilities (a) and (b), and disallows e.g. (c) and all others.

Definition 4.14 (Planar 3-SAT)**.** *Given are an instance consisting of a set U of variables and a collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$, where the following graph $G = (V, E)$ is planar:*

- *In V, there is a vertex for every clause $c \in C$*
- *In V, there are two vertices x_i and $\overline{x_i}$ for every variable $x_i \in U$*
- *If the positive literal of x_i appears in clause c_j, then $\{x_i, c_j\} \in E$*
- *If the negative literal of x_i appears in clause c_j, then $\{\overline{x_i}, c_j\} \in E$*
- *For every variable x_i, the two representative vertices are connected: $\{x_i, \overline{x_i}\} \in E$*
- *Every vertex representing a negative literal is connected to the vertex representing the positive literal of the next variable: $\{\overline{x_i}, x_{i+1}\} \in E$*

Is there a satisfying truth assignment for C?

This problem states that a form of 3-SAT remains $\mathcal{NP}$-hard, in which a planar graph, and in fact even a drawing of a planar graph as illustrated in Figure 4.23, can be derived from a problem instance in the following way: The two possible literals of all variables $x_1, x_2, \dots x_n$ become vertices and are drawn as boxes of fixed height. These boxes are all aligned vertically and connected into a path (see the turquoise path in Figure 4.23). The clauses $C_1, C_2, \dots C_m$ also become vertices and are drawn as boxes above or below the row of variables. Clauses then connect to the variables that appear in them via a vertical line segment. Note that this also means that every clause-box connects to all of its variable-boxes either only from above or only from below.

This problem was shown to be $\mathcal{NP}$-hard by Lichtenstein [Lic82] in its basic form, where only one vertex represents each variable. The form used here, where every variable is represented by two vertices, one per literal, and where the variable-representing vertices form a connected backbone (see the turquoise edges in Figure 4.23), was shown to be still $\mathcal{NP}$-hard by de Berg and Khosravi [dBK10].

Big Picture

To prove $\mathcal{NP}$-hardness in our case, we transform a Planar 3-SAT instance into a Maxdegree-4 instance by constructing a graph that can be laid out like the example in Figure 4.23 without any edge-bends if and only if the Planar 3-SAT instance is satisfiable. To this end, we introduce three gadgets: One for the variables, one for the clauses, and one helper gadget that allows us to restrict the way that edges may bend at vertices. Throughout the whole construction, it is important to keep in mind that we only consider completely edge-bend-free drawings as valid (by setting K appropriately).

4.5.2. 0°/180°-Gadget

The first gadget forces a path of length 3 to have one of two possible configurations: either the first and the last edge of that path have an angle of 0°, or they have an angle of 180° between them (or, in terms of rotation: have either rotation 2 or rotation 0). Figure 4.24 illustrates this: While (a) and (b) represent the two permitted shapes of the path, (c) is an example for something that the gadget prevents. The gadget is presented in Figure 4.25. The function b for the vertices of this gadget is defined as follows:

$$b(v) = \begin{cases} \{e_3, e_4, e_5, e_6\} & \text{if } v = v_0 \\ \emptyset & \text{otherwise} \end{cases}$$

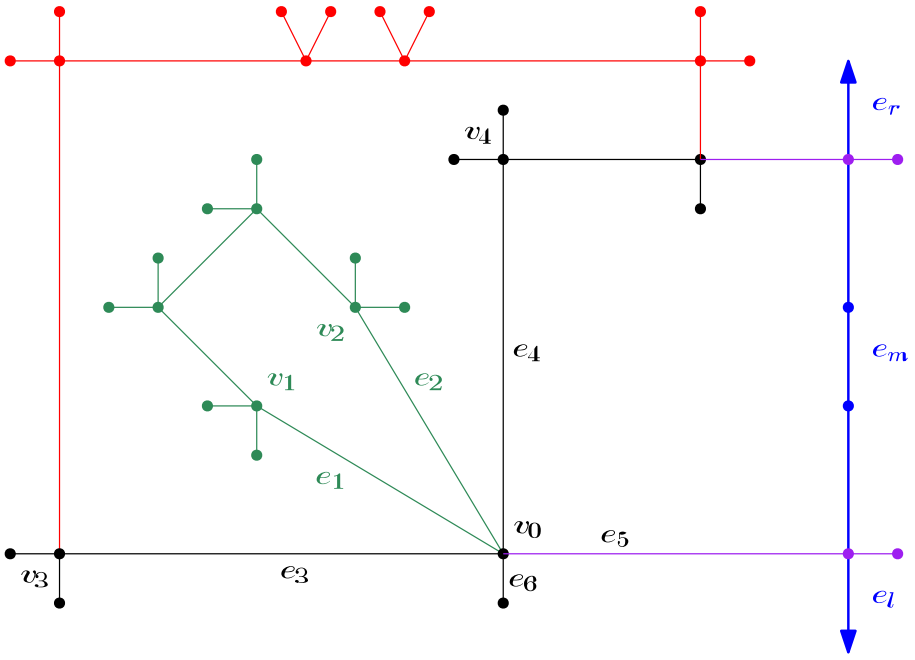


Figure 4.25.: Gadget to enforce 0°/180° angles

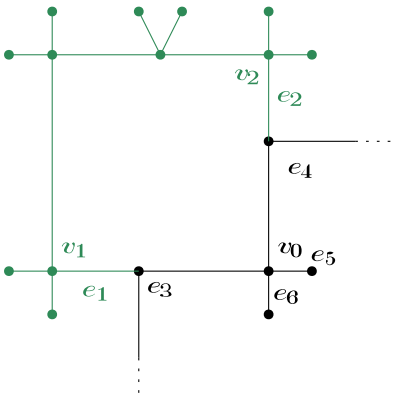


Figure 4.26.: A way in which e_1 and e_2 cannot be removed from v_0. Only an excerpt from the gadget is shown.

The gadget mainly consists of two paths, the black and the red paths, plus some green path connecting to v_0. The green path ends in two edges that must be removed from v_0. Note that all other edges incident to v_0 cannot be moved away from v_0 because of our choice of $b(v_0)$. Also note the two purple edges connecting the rest of the gadget to the path of length 3 (actually length 5, since it is subdivided by two purple vertices, shown in blue) which should be forced into one of the two allowed configurations. The gadget ensures that the only possible layouts, i.e. the only layouts that can be drawn without a bend on an edge, are layouts in which both blue arrows point in the same direction, or layouts in which both blue arrows point away from each other.

With the definition of b, e_1 and e_2 are the only edges that can be removed from v_0, and both edges must be removed to have maximum degree 4 at v_0. Note that the red path and the black path are incompatible in the sense that the face enclosed by the red and black path (ignoring the green path for a moment) would not have rotation 4 in a drawing without bent edges: Simple counting results in a rotation of 6. The broad idea is now that e_1 and e_2 can be moved clockwise or counterclockwise, thereby inserting two new vertices into e_3 or e_4, and thus permitting two more bends (which are outward bends for

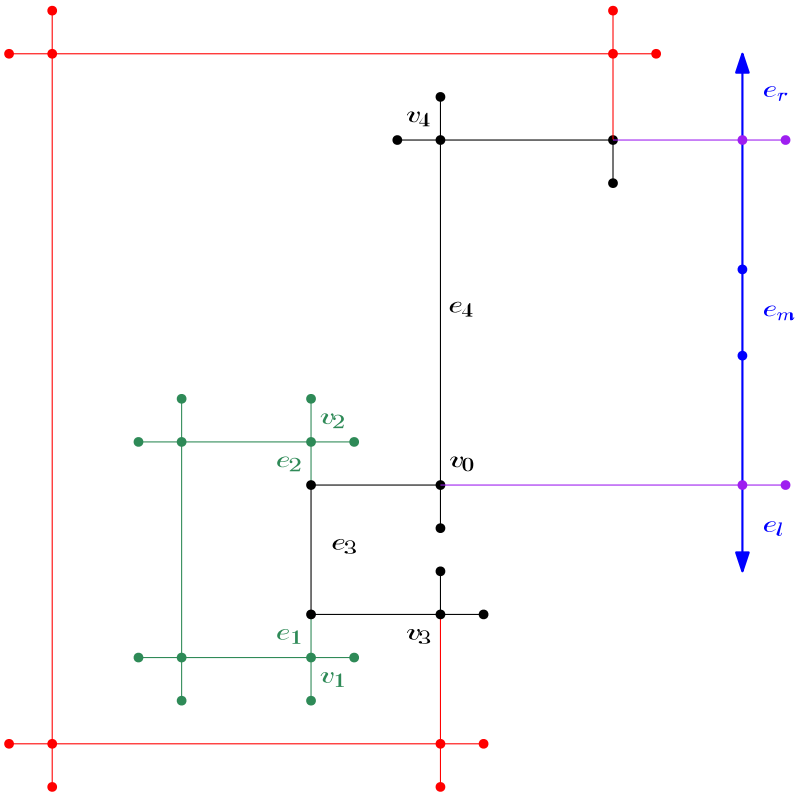


Figure 4.27.: The gadget in the 0° layout

the face enclosed by the black and red path) on the black path without having to bend an edge.

We now evaluate the effect of different orders and directions in which e_1 and e_2 are moved away from v_0. First, if e_1 is selected first, and *clockwise* is selected as direction, then e_1 would be moved to a vertex inserted into e_2. However with this, only at most one additional vertex can be inserted into the black path (by moving e_2 later), which is not enough to facilitate a drawing without an edge-bend. The same situation arises if e_2 is selected first together with the direction *counterclockwise*. Thus, these choices never lead to the desired edge-bend-free drawing.

Now, if e_1 is moved counterclockwise and e_2 is moved clockwise (regardless of the order), the situation depicted in Figure 4.26 arises: Edges e_1 and e_2 can now have at most a 90° angle between them (if no edge is to be bent), which is not sufficient to draw the green path without an edge-bend.

Thus, the only two possible ways of removing e_1 and e_2 from v_0, that can possibly lead to a drawing without any bent edges are:

Case 1 First move e_1 counterclockwise, then move e_2 counterclockwise

Case 2 First move e_2 clockwise, then move e_1 clockwise

It is now important to note that the blue path consisting of e_l, e_m and e_r is forced to be parallel to e_4 by the purple parts of the graph (again under the assumption that no edge is to be bent). Thus, we must now evaluate how the edge e_4 (and maybe a subdivision of e_4) can be shaped in a layout without edge-bends.

Case 1 leads to the situation drawn in Figure 4.27: Here, e_1 and e_2 introduce new possible bends in e_3, i.e. not on e_4. Note how both blue arrows still point away from each other, since e_4 is straight and the blue path must be parallel to it.

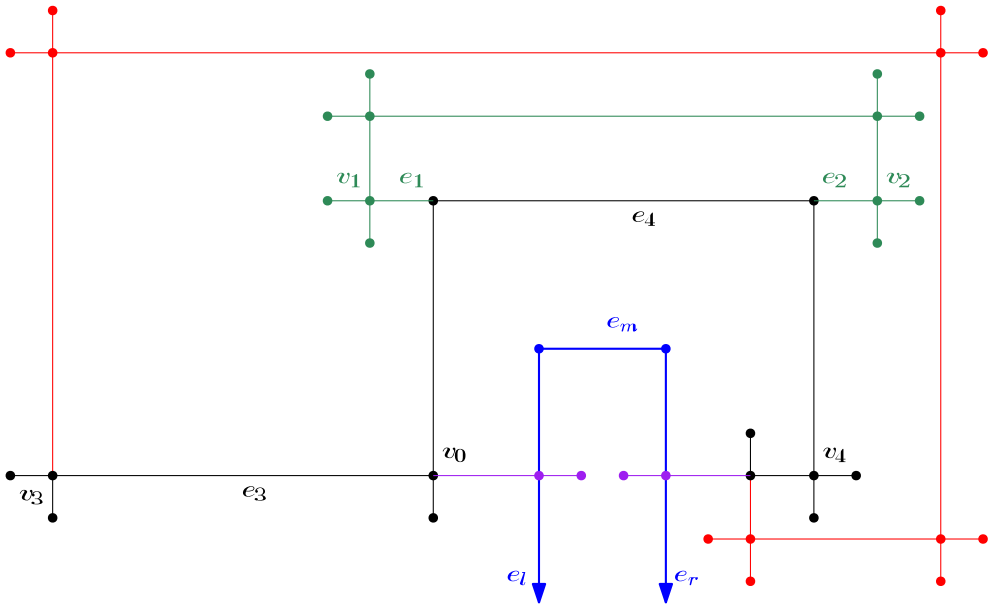
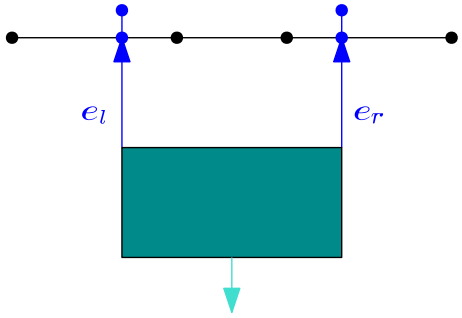
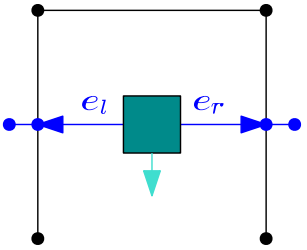


Figure 4.28.: The gadget in the 180° layout



(a) The gadget forcing some path to have rotation 0



(b) The gadget forcing some path to have rotation 2

Figure 4.29.: Illustrations of how to connect the gadget to the rest of the graph. Only the blue path is shown in detail, all other parts of the gadget are hidden inside the green box.

Case 2 leads to what is illustrated in Figure 4.28: Now, e_1 and e_2 introduce the two bends in e_4. Note that now, both blue arrows point in the same direction, again in accordance with the requirement that the blue path must be parallel to the subdivision of e_4.

When using this gadget in the construction of G, we usually hide its complexity and only show the gadget as a box, with the blue path consisting of e_l, e_m and e_r sticking out of the box; see Figure 4.29. Here, only the edges e_l and e_r are shown in detail, and it is connected to some other part of the graph (again a path in this case), forcing it into one of two possible configurations.

In these figures, note the additional turquoise arrow pointing out of the gadget. We call this the *anchor* of the gadget. For use in the variable gadget, we must be able to connect something to the bottom of this gadget, i.e. in the direction that the turquoise arrow points, which is opposite direction of the direction that the blue arrows have when pointing in the same direction. Unfortunately, the gadget itself has no element that reliably is drawn in this way. We circumvent this problem by adding another path of length 3 (see the turquoise part in Figure 4.30) that is forced to be parallel to the blue path. Here, the anchor can be attached as illustrated in Figure 4.30.

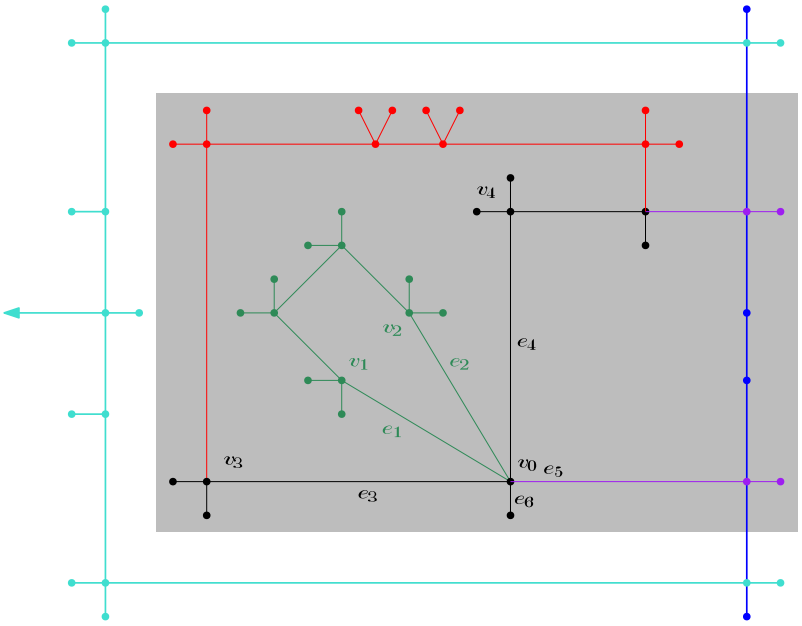


Figure 4.30.: Extension of the 0°/180°-gadget to have an anchor point that always points down. Original gadget shown with gray background.

4.5.3. Variable and Clause Gadgets

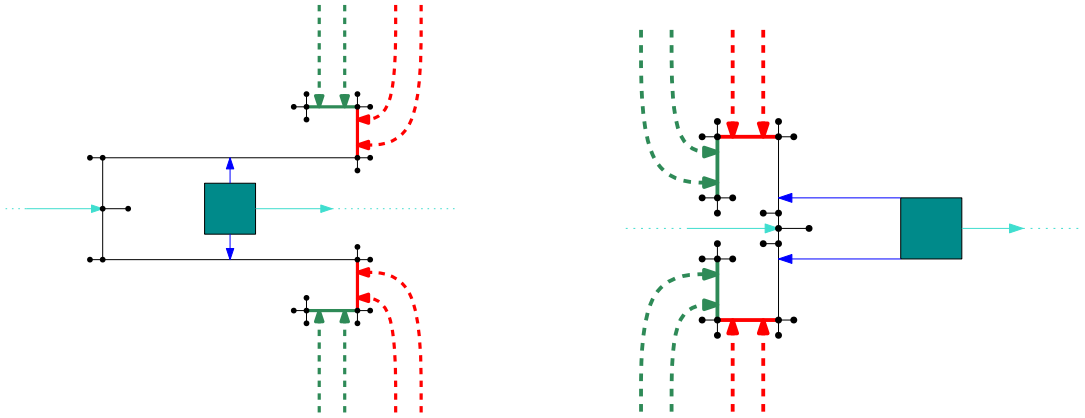
We now show how we represent variables and clauses in our transformed graphs. The gadgets used for variables are depicted in Figure 4.31, where the layout in Figure 4.31a is interpreted as a variable assigned with *true*, and the layout in Figure 4.31b represents *false*. In the center of the gadget is a 0°/180°-gadget hidden in a box that forces the variable gadget into one of these two configurations. The turquoise lines indicate how the variable gadgets are connected to each other: Since we can line them up horizontally (as shown in [dBK10])), we link them into one long line using their anchors. We also define *horizontal* to be the direction of that line.

Please note that although we could, we do not split the variable gadgets into one gadget for the positive and one gadget for the negative literal of each variable. Rather, every variable gadgets has four separate areas where clause gadgets may connect: Two for clauses using the positive literal of the variable, and two for clauses using the negative literal of the variable. The dashed arrows illustrate how the clause gadgets connect to the variable gadgets. Similar to the drawing in Figure 4.23, our clause gadgets explained below have three legs connecting to the respective variable gadgets. Each of the legs represents an occurrence of a variable in a clause, which can either be positive or negative. In Figure 4.31, we colored the legs representing negative occurrences (called *negative legs*) and the points where they connect to the variable gadget in red. Positive legs are colored green. Note how in both possible layouts, either only negative legs or only positive legs connect vertically to the gadget, while the other set of legs must connect horizontally.

The clause gadget follows the idea of drawing a Planar 3-SAT instance as illustrated in Figure 4.23. An example can be seen in Figure 4.32: Here, v_c has degree 6 and must have two edges removed. We also specify b for this gadget:

$$f(v) = \begin{cases} \{e_l, e_m, e_r, e_a\} & \text{if } v = v_c \\ \emptyset & \text{otherwise} \end{cases}$$

The algorithm can thus only move e_1 and e_2 away from v_c. The edges drawn as arrows in Figure 4.32 are used to connect the gadget to other parts of the graph. While e_l, e_m



(a) The gadget representing a variable, in a configuration representing *true*.

(b) The gadget representing a variable, in a configuration representing *false*.

Figure 4.31.: The gadget used to represent a variable, in its two possible configurations. Dashed arrows indicate how the clause gadgets connect their legs. Red legs represent negative usages of this variable, green legs positive usages. Dotted edges indicate how the variable gadgets are chained. The green box hides a 0°/180°-gadget.

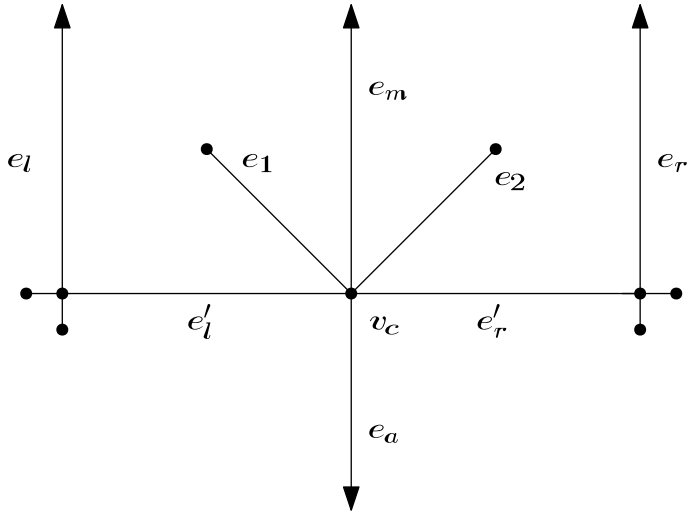
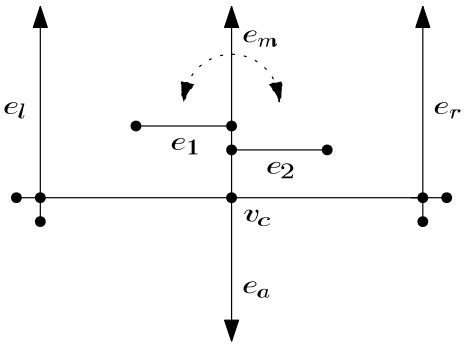
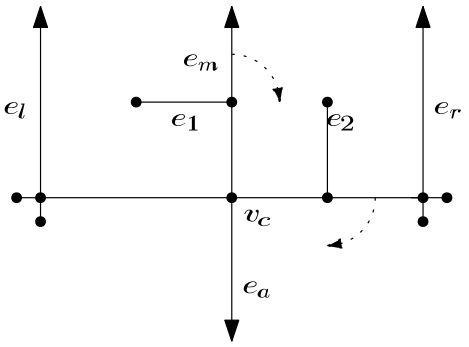


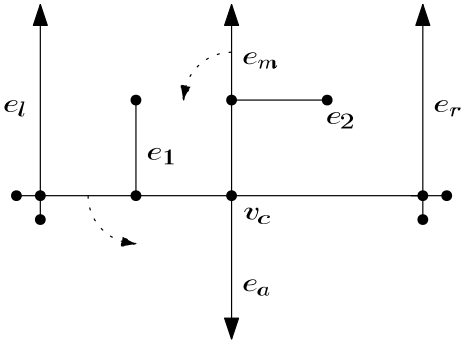
Figure 4.32.: The clause gadget



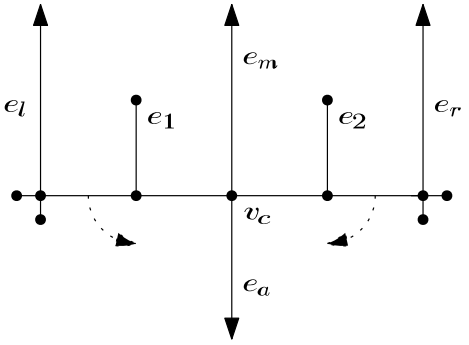
(a) Possible situation in the e_m leg can rotate in both directions.



(b) Possible situation in which the e_m and the e_r legs can rotate clockwise.



(c) Possible situation in which the e_m and the e_l legs can rotate counterclockwise.



(d) Possible situation in which the e_l leg can rotate counterclockwise and the e_r leg can rotate clockwise.

Figure 4.33.: Illustrations of all four possible ways of moving e_1 and e_2 away from v_c in the clause gadget. The dotted arrows indicate in which directions the legs can rotate.

and e_r are the connections to the three variable gadgets belonging to the corresponding clause, e_a is used to enforce a certain alignment of the clause gadgets. Later on, we show that we can force e_a to always be laid out vertically. If we now ignore e_1 and e_2 for a moment, this would mean that e_l, e_r and e_m must also be laid out vertically if we allow for no bends. Now, e_1 and e_2 can be moved to any two of e'_l, e_m and e'_r, thereby inserting two new vertices in at most two of e'_l, e_m and e'_r. We call e_m, e_r together with e'_r, and e_l together with e'_l the three *legs* of this gadget.

As illustrated in Figure 4.33, every one of the four possibilities of moving e_1 and e_2 results in a situation where up to two of the legs can (but do not have to) rotate by 90°, but never all three of them. Together with the fact that we assume e_a to always point down, this means that one leg always has to point up. We now show how to connect the clause and the variable gadgets. As depicted in Figure 4.23, the clauses can be partitioned in a set being drawn above the variables and a set being drawn below the variables, called the *upper* respective *lower* set. For the lower set, we use the clause gadget as illustrated in Figure 4.32, for the upper set we use a vertically mirrored version of it, i.e. with e_a pointing up instead of down.

Assuming that e_a is laid out vertically, this results in at least one leg having to connect to the variable gadget vertically. Together with the fact that at the variable gadget, only positive legs can connect vertically if the variable gadget is laid out in the configuration representing *true*, and only negative legs can connect vertically if the variable gadget is laid out representing *false*, this leg forced to connect vertically represents a satisfied literal in the respective clause.

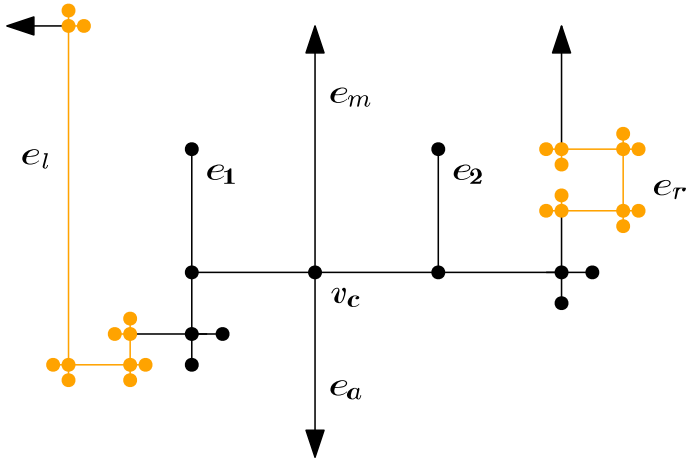


Figure 4.34.: Modification to the clause gadget to facilitate connection to the variables.

We need to make two more modifications to the clause gadget. First, when the e_l or e_r leg are rotated outwards, they have no way anymore of reaching toward the middle of the drawing, where the variable gadgets are. We fix that by inserting a series of U-shaped bends into the e_r leg, and a mirrored version into the e_l leg. You can see the modification in Figure 4.34: The orange parts have been inserted. In e_r, the structure is not used. It can easily be seen that this structure does not change the possible orientations of the legs. In e_l, the structure is used to move the e_l leg towards the middle of the drawing.

The second modification makes sure that any leg that is able to rotate can rotate in such a way that it is able to connect bend-free to its variable gadget. If a leg represents a satisfied literal in a clause, it stays vertical and connects vertically to the variable gadget. If, however, a leg represents an unsatisfied literal, it rotates by 90° and connects horizontally. The problem here is that the direction in which the leg points does not have to match the direction required for connecting to the variable gadget. For example, an e_r leg can only rotate clockwise, and thus point right, while an e_l leg can only rotate counterclockwise and point left. For e_m it even depends on which of e_1 and e_2 were moved to e_m. We thus must make sure that if a leg was rotated, it is able to point in both of the horizontal directions. Here, we use the 0°/180°-gadget shown in Section 4.5.2. With this, we can change a leg that points right to a leg pointing left. Figure 4.35 illustrates this situation: One 0°/180°-gadget has been inserted into e_m and e_r each. In this case, e_r demonstrates how it is now possible to point e_r left without any bends although it was rotated clockwise. At e_m it can be seen that, since neither e_1 nor e_2 was moved to e_m, even with the inserted 0°/180°-gadget, the leg can still only be directed vertically.

4.5.4. Anchoring the Clause Gadgets

So far, we always assumed that we can fix the e_a edge of the clause gadgets to point down (or up, in the case of the upper set). We now show how this is achieved. In a planar layout of a Planar 3-SAT instance as depicted in Figure 4.23, the clauses are nested: Some clauses, like C_2 in the example, are drawn in between two legs of another clause. We now first show how to anchor a clause gadget under the assumption that we can find some edges e'_{a1} and e'_{a2} definitely pointing down between the e_l and the e_m leg as well as between the e_m and the e_r leg. Figure 4.36a illustrates this situation: The blue parts are the anchors we have assumed. To them, e_1 and e_2 are attached such that they must always be laid out vertically. We must now show that this does not interfere with the way that the clause gadget should be laid out. Refer back to Figure 4.33, showing all possible distributions of e_1 and e_2. With both e_1 and e_2 pointing up, the situation illustrated in

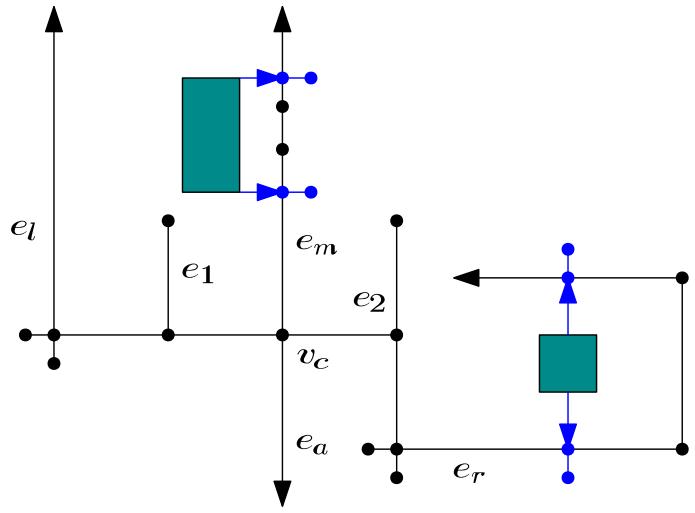


Figure 4.35.: Using the 0°/180°-gadget to enable a clause gadget to point all legs left if they can rotate at all. The 0°/180°-gadget itself is hidden in the green box. Refer to Section 4.5.2 for details on that gadget.

Figure 4.33a is clearly not possible anymore. Also, in the situations depicted in Figure 4.33c and Figure 4.33b, the bend on e_m is not optional anymore.

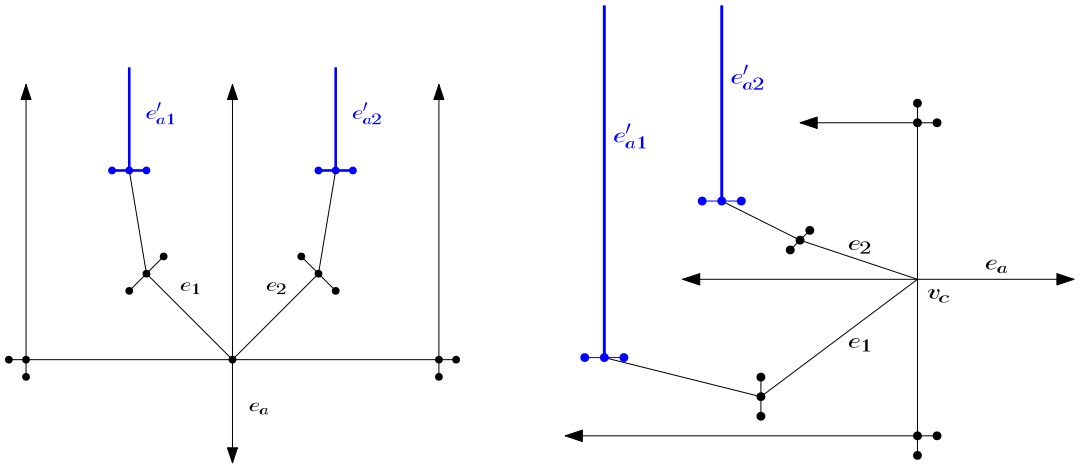
This is not a problem. We only need a valid layout for every set of up to two rotated legs. If no leg should rotate, if one of e_r and e_l or if both e_r and e_l should be rotated, the situation in Figure 4.33d yields a valid layout with both e_1 and e_2 pointing up. In case only e_m should be rotated, the situation in Figure 4.33b yields a valid result with e_1 and e_2 pointing up. If e_m should be rotated together with e_l respectively e_r, the situations in Figure 4.33c respectively Figure 4.33b yield valid layouts and have e_1 and e_2 pointing up. Thus, for every necessary case, we can still find a valid layout.

We now show how connecting e_1 and e_2 to the anchors forces e_a to point down: Assume that e_a points right. We still assume both anchors e'_{a1} and e'_{a2} to point down. Since e_1 must be pointing toward e'_{a1}, the clockwise angle between e_1 and e_a must now be 90°. But at the same time, there are at least e_m and e_r clockwise between e_1 and e_a at v_c. Thus, this could not be embedded without bends. The case that e_a points left follows from symmetry.

It remains to be shown that we can always find the anchors e'_{a1} and e'_{a2}. For any clause gadget that has no further clause gadgets nested in it, we can run an edge down to the string of horizontally connected variable gadgets. This gives us two anchors that always point down. If a clause gadget has further clause gadgets nested between its legs, the e_a edges of these nested gadgets can be used recursively.

4.5.5. Conclusion

To conclude the proof, we must finally show that the transformation is polynomial, and that the resulting MAXDEGREE-4 instance is an accepted yes-instance if and only if the original PLANAR 3-SAT instance was a yes-instance. It is easy to show that the transformation is polynomial: The clause and 0°/180° gadgets each have a constant number of vertices and edges. The variable gadget has a number of edges and vertices that is linear in the number of clauses, which is polynomial. Certainly, the number of gadgets used is polynomial in the number of variables respectively clauses in the PLANAR 3-SAT instance. Thus, the graph in total contains a number of edges and vertices that is polynomial in the size of the PLANAR 3-SAT instance. Also, all gadgets can be constructed in constant time once a planar embedding is computed on the PLANAR 3-SAT instance, which is also polynomial.



(a) Anchoring a clause based on two anchors. The blue parts are the two anchors assumed to always point down. They have e_1 and e_2 attached to them such that they must always be vertical.

(b) Layout made impossible by anchoring e_1 and e_2.

Figure 4.36.: Illustrations for anchoring the clause gadget

If a planar layout without bends is possible, we can look at every variable gadget and assign the truth value it represents (cf. Section 4.5.3). Then, in the layout, every clause gadget must have one leg vertically connecting to a variable gadget. However, as we have shown, a leg can only connect vertically to a variable gadget if the literal that the leg represents is satisfied. Thus, the inferred solution to the PLANAR 3-SAT instance is valid. On the other hand, if the PLANAR 3-SAT instance has a solution satisfying all clauses, we can find a bend-free layout by laying out the variable gadgets according to the truth assignments of the PLANAR 3-SAT instance. Then, every clause gadget can connect at least one leg vertically, while its other two legs may connect horizontally to the variable gadgets, as shown.

In conclusion, we have shown the following theorem:

Theorem 4.15. *$\mathcal{NP}$-hardness of* MAXDEGREE-4
The MAXDEGREE-4 *problem is $\mathcal{NP}$-hard.*

From here, it as also easy to show this corollary:

Corollary 4.16. *$\mathcal{APX}$-hardness of* MAXDEGREE-4
The MAXDEGREE-4 *problem is $\mathcal{APX}$-hard.*

Proof. Assume there is an approximation algorithm for MAXDEGREE-4 with approximation factor α. Transform any PLANAR 3-SAT instance into a MAXDEGREE-4 instance as described above. If the MAXDEGREE-4 instance is solvable for $K = 0$ edge-bends, the approximation algorithm must find a solution with $\alpha K = 0$ edge-bends with polynomial time complexity. From this, we could again derive a solution to the original PLANAR 3-SAT instance as shown above. Thus, no polynomial-time approximation algorithm can exist under the assumption $\mathcal{P} \neq \mathcal{NP}$. □

4.6. Implementation and Experimental Evaluation

We implemented and practically evaluated the approach discussed in Section 4. We first describe the implementation and the input datasets in Section 4.6.1. We then present the

City	Vertices	Edges	Metro Lines	Maximum Degree
Montréal	68	69	4	5
London	209	236	11	7
Berlin	331	380	26	7

Table 4.1.: Properties of the instances used in the experimental evaluation

experimental setup and the metrics we wanted to explore in Section 4.6.2. In Section 4.6.3 we then discuss the measured results. That section is further subdivided into Section 4.6.3.1, which examines the parameters that take effect during the *Shape* step of the framework, and Section 4.6.3.2, examining the parameters that influence the *Metrics* step. We conclude the experimental evaluation with a subjective assessment of our results in Section 4.6.4.

4.6.1. Implementation and Data

Since precise performance measurements were not among the chief objectives of our experiments, Python was chosen as development language.

The software roughly works in four steps: First, data is read, parsed and preprocessed. In the second step, we perform Tamassia's *Shape* step by computing the modified flow network described in Section 4.2. In the third step, we compute actual positions for vertices, thereby executing Tamassia's *Metrics* step. For this, we use the simulated-annealing-based technique described in Section 4.3. We use a cooling schedule with quadratic cooling and a pre-set number of annealing steps (see parameters below).

All experiments were computed on machines with 48 2.1 GHz AMD Opteron CPU cores and 256 GB of RAM. However, since our implementation does not use parallelism, we usually computed 48 instances in parallel on these machines.

As input, the software processes transportation data supplied in *General Transit Feed Specification* (GTFS) format. This format, originally developed by Google, has become somewhat of a standard since it allows transport agencies to supply information about their networks to Google Maps. In fact, there even exist sites collecting large amounts of these GTFS datasets. Unfortunately, most of these datasets require customized filtering and preprocessing to be usable. For example, many datasets do not just include one type of transportation, e.g. a metro transportation network, but include under- and overground railways of different types, buses, etc. Also, since the purpose of these datasets usually is to allow public-transportation routing, many maps have quirks that must be handled. For example, in the official Berlin dataset, many subway stations are modeled not as one station, but as a collection of stations with differing names, positions, and connections.

We cleaned up the datasets of Berlin, London and Montréal to the extent that they became usable for our purposes. While Montréal is a rather small instance, Berlin and London both have large and complex metro networks. Table 4.1 lists the size of the graphs derived from the metro networks before any planarization or degree limitation was done, as well as the number of actual metro lines the respective metro network has.

4.6.2. Experimental Setup

The framework we implemented has a number of parameters, mostly balancing different optimization criteria against each other. The procedure of our experiments is as follows: Depending on what we were interested in examining, we first decided on a parameter space to explore, i.e. a set of values for every parameter. Then, for every possible combination of

these values, multiple instances were computed, i.e. multiple drawings were produced. We then determined the values that our metrics assumed for these instances.

We now briefly describe the parameters as well as the metrics used in our evaluation.

Parameters ω_{bends}, ω_{line}, and ω_{station}

As defined in Definition 2.4 and described in Section 4.2, these parameters control the costs incurred to the flow network used for computing the shape. Here, ω_{bends} specifies the costs of an edge-bend, ω_{line} represents the costs of a metro line bending on an edge, and ω_{station} gives the costs of a metro line bending at a station.

Parameters ω_{area} and ω_{dist}

During the final compaction step, we use a metaheuristic that treats the problem as a black box. We can therefore try to optimize multiple criteria during this step. In our case, we use the simulated annealing process to minimize the maximum height of the drawing (which is the only factor determining the used area in a circular drawing) as well as the standard deviation of the distances between neighboring metro stations. To do so, ω_{area} is the weight (i.e. priority) of minimizing the height, while ω_{dist} is the weight of the standard deviation of the distances between neighboring stations.

Parameters *#anneal* and *reaugment-percentage*

As explained in Section 3.2 and Section 4.3, simulated annealing is an iterative process, which in every iteration generates a solution similar to the last solution. To achieve this in practice, we need to specify how many of these iterations we compute. We call this number *#anneal*. In Section 4.3, we describe that producing a similar solution is done by removing some edges and vertices used for augmentation and then augmenting the graph again. The parameter *reaugment-percentage* specifies what fraction of G's faces are unaugmented.

Please note that the parameters ω_{bends}, ω_{line}, and ω_{station} are rather independent from the parameters ω_{area}, ω_{dist}, *#anneal*, and *reaugment-percentage*, since the two groups take effect during two separate steps of the framework. We do evaluate their interplay, but primarily focus on evaluating the results when varying the parameters within one of those two groups.

Metrics

The metrics we measured are closely connected to quality criteria from Section 2.2 and thus the objective function from Definition 2.4. Our metrics are:

height The maximum radius used in the final circular drawing. Note that in a circular drawing, the maximum radius alone determines the area usage. This corresponds to (O1).

edge-bends The number of times that an edge bends in the drawing. This corresponds to (O2).

line-bends The sum of all edge-bends weighted by the number of metro lines running on the edge of the respective edge-bend. This corresponds to (O2a).

station-bends The number of times that a metro line bends at a metro station. This corresponds to (O3).

standard deviation of station distance Given the distances between all pairs of vertices that have an edge between them, we compute the standard deviation of this set. This corresponds to (O4).

Parameter Name	Values	# of measurements per value
City	Berlin, London	9000
ω_{station}	$0.0, 1.0, 3.0, 5.0, 10.0$	3600
ω_{edge}	1.0	18000
ω_{line}	$0.0, 1.0, 5.0$	6000
ω_{height}	$1500, 2000, 2500$	6000
ω_{dist}	$100, 300$	9000
#anneal	$0, 10, 100, 1000, 5000$	3600
reaugment-percentage	$0.03, 0.02$	9000

Table 4.2.: Parameter Space of Dataset 1. The first column specifies the parameter, the second column lists the values tried for that parameter. The third column indicates how many measurements were taken for every value of the parameter in this row.

Normalization

In our experimental runs, we varied all of the parameters, and, for every combination of parameters, computed multiple instances. When evaluating the effect of one parameter on one metric, it is desirable to eliminate the effects that the other parameters have on the metric. As an example, assume for the moment that we had just two parameters: *#anneal* and the city of the dataset we are using. We want to evaluate the height of the final drawing. Now, since Montréal is far smaller than London and Berlin, and London is somewhat smaller than Berlin (in terms of the metro network's size), it is highly likely that the drawings of Montréal are smaller than the drawings of London or Berlin, and the drawing of London is smaller than the drawing of Berlin - regardless of the value of *#anneal*. Thus, our measurements would always form three separate clusters, which would distort our statistics.

Therefore, we normalize the data in the following way: When evaluating the effect of a parameter on a metric, we take the set of all parameters minus the parameter we are evaluating. For these parameters, we take every possible combination of values. For every such combination, we get a set of measurements for the metric under consideration. Within this set of measurements, we determine the median value and divide all other values in the set by the median.

4.6.3. Results

The first dataset we present constitutes an overall evaluation of the parameter space. Table 4.2 lists the values we tried for each of the parameters. We conducted preliminary experiments to narrow down the range of interesting values per parameter. For each possible configuration, ten instances were computed, resulting in a total of 18000 measurements. Note that since Montréal has a very simple metro network, it did not turn out to be useful for this exploratory type of experiment.

Please also note that the two parameter groups ω_{station}, ω_{edge}, ω_{line}, and ω_{height}, ω_{dist}, *#anneal*, *reaugment-percentage* take effect in two separate steps of the framework. The fact that the parameters ω_{height} and ω_{dist} are far larger than the other weighting factors does thus not influence the results of the *Shape* metrics. The rather large values are technically motivated, since the simulated annealing algorithm needs some scaling to its objective function.

We first look at the parameters taking effect during the *Shape* phase, and then look at the parameters which influence the *Metrics* phase.

4.6.3.1. Parameters influencing Shape

We first take a look at the effect of the parameters concerned with edge- and line-bends. We start by examining the effects of the ω_{line} parameter. Figure 4.37 shows a box-whisker plot of the ω_{line} parameter value against the normalized number of lines bending on edges, while Table 4.3 displays the actual values. We can see that by raising ω_{line} from 0.0 to 5.0 we can reduce the normalized number of lines bending on edges by a factor of about 2. We performed a *Wilcoxon Signed-Rank Test* on the data to determine whether the measurements were pairwise significantly different for different values of ω_{line}. Table A.6 in the appendix lists the p-values of this experiment. Applying the standard significance level of 0.05, we may assume that all measurements were pairwise significantly different.

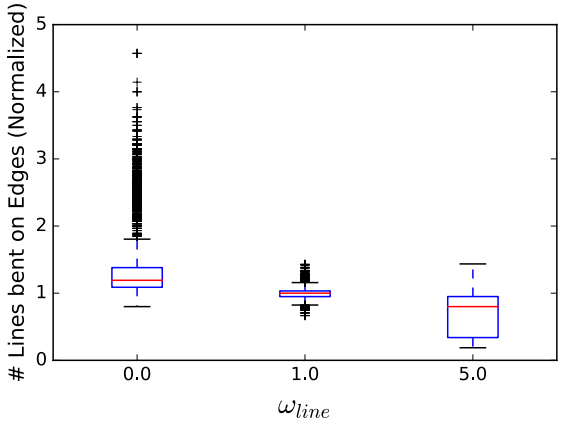


Figure 4.37.: Evaluation of ω_{line}'s influence on the number of lines bending on edges

	No. of line-bends on edges		
ω_{station}	Average	Median	Std. Dev.
0.0	1.43	1.19	0.569
1.0	0.999	1	0.0911
5.0	0.687	0.8	0.309

Table 4.3.: Evaluation of ω_{line}'s influence on the number of lines bending on edges

To further evaluate which values are reasonable for ω_{line}, we computed another set of experiments, this time only varying the ω_{line} parameter. The exact set of parameters used can be found in Table A.1 in the appendix. Table 4.5a shows values for an induced metric: For every instance, we subtract the number of edge-bends from the number of line-bends. Note that a value of 0 implies that every edge-bend happens on an edge carrying only one metro line. As we can see, for a value of $\omega_{\text{line}} = 5.0$, the maximum difference is already 0, implying that across all 1000 computed instances with $\omega_{\text{line}} = 5.0$, no edge-bend happened on any edge carrying more than one metro line. Thus, we assume that (at least for the metro networks under consideration), setting ω_{line} to a value larger than 5.0 is not useful.

Next, we examine whether ω_{station} is similarly successful in reducing the number of metro lines bending at stations. As before, Figure 4.38 is a box-whisker plot of the normalized number of lines bending at stations against the value of ω_{station}. Table 4.4 lists the actual values. We can see that, without any incentive to avoid lines bending at stations (i.e. $\omega_{\text{station}} = 0$), the normalized number of lines doing so is about two times higher than it is even for $\omega_{\text{station}} = 1.0$. However, by raising ω_{station} to 10.0, we can reduce the normalized number of lines bending at a station by another factor of about 1.4.

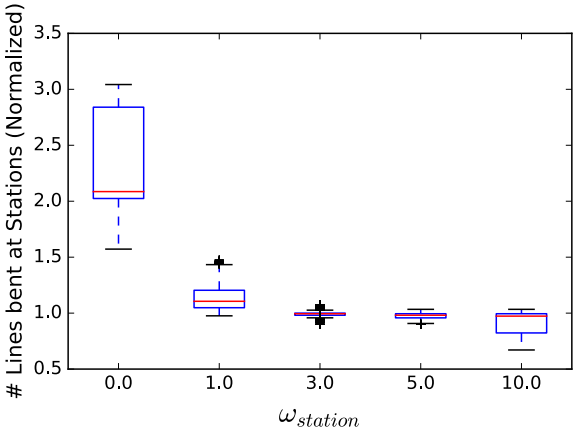


Figure 4.38.: Evaluation of ω_{station}'s influence on the number of lines bending at stations

	No. of line-bends at stations		
ω_{station}	Average	Median	Std. Dev.
0.0	2.27	2.09	0.456
1.0	1.15	1.11	0.129
3.0	0.99	0.995	0.0253
5.0	0.975	0.981	0.0244
10.0	0.91	0.973	0.11

Table 4.4.: Evaluation of ω_{station}'s influence on the number of lines bending at stations

	Line-Edge-Bend Difference	
ω_{line}	Average	Maximum
0.0	5.49	12
1.0	1.34	6
2.0	1.3	5
3.0	0.048	1
4.0	0.024	1
5.0	0	0
7.5	0	0
10.0	0	0

(a) Listing the difference between the number of line-bends and edge-bends by value for ω_{line}.

	# Lines bent at Stations		
ω_{station}	Average	Median	Std.Dev.
0.0	2.42	2.41	0.0492
1.0	1.18	1.17	0.0297
5.0	1.11	1.1	0.0319
6.0	1	1	0.0357
7.0	0.814	0.817	0.023
8.0	0.816	0.817	0.0233
10.0	0.817	0.817	0.023

(b) Evaluation of ω_{station}'s influence on the number of lines bending at stations under the condition $\omega_{\text{line}} = 5.0$

Table 4.5.: Evaluation of metro lines bending on edges

We again performed a Wilcoxon-Test on the data to determine whether the measurements were pairwise significantly different. Table A.5 in the appendix lists the p-values of this experiment. All pairwise p-values are below 10^{-50}, thus we may assume pairwise statistically significant differences.

However, since we determined a desired value of $\omega_{\text{line}} = 5.0$ above, we were especially interested in how the number of lines bending at stations behaves if ω_{line} is chosen as desired. We computed an additional dataset for this case, the parameter space of which is listed in Table A.2 in the appendix. Table 4.5b and Figure 4.40a present normalized number of lines bending at stations for different choices of ω_{station}. We can see that $\omega_{\text{station}} = 7.0$ seems to be the largest useful value of ω_{station}, since the number of lines bending at stations does not fall further for larger values. In fact, for all consecutive pairs of values up to 7.0, the Wilcoxon-Test reports significant difference, while it does not report that for any consecutive pair of values larger than 7.0 (see Table A.8 in the appendix).

However, the value 7.0 is somewhat special insofar as that with the choice of ω_{line}, the flow network's costs for bending a single edge with a single line on it are $\omega_{\text{line}} + \omega_{\text{edge}} = 5.0 + 1.0 = 6.0$. Thus, setting $\omega_{\text{station}} = 7.0$ indicates that if a bend at a station can be avoided by introducing a bend on an edge carrying a single line, it should be done so, while

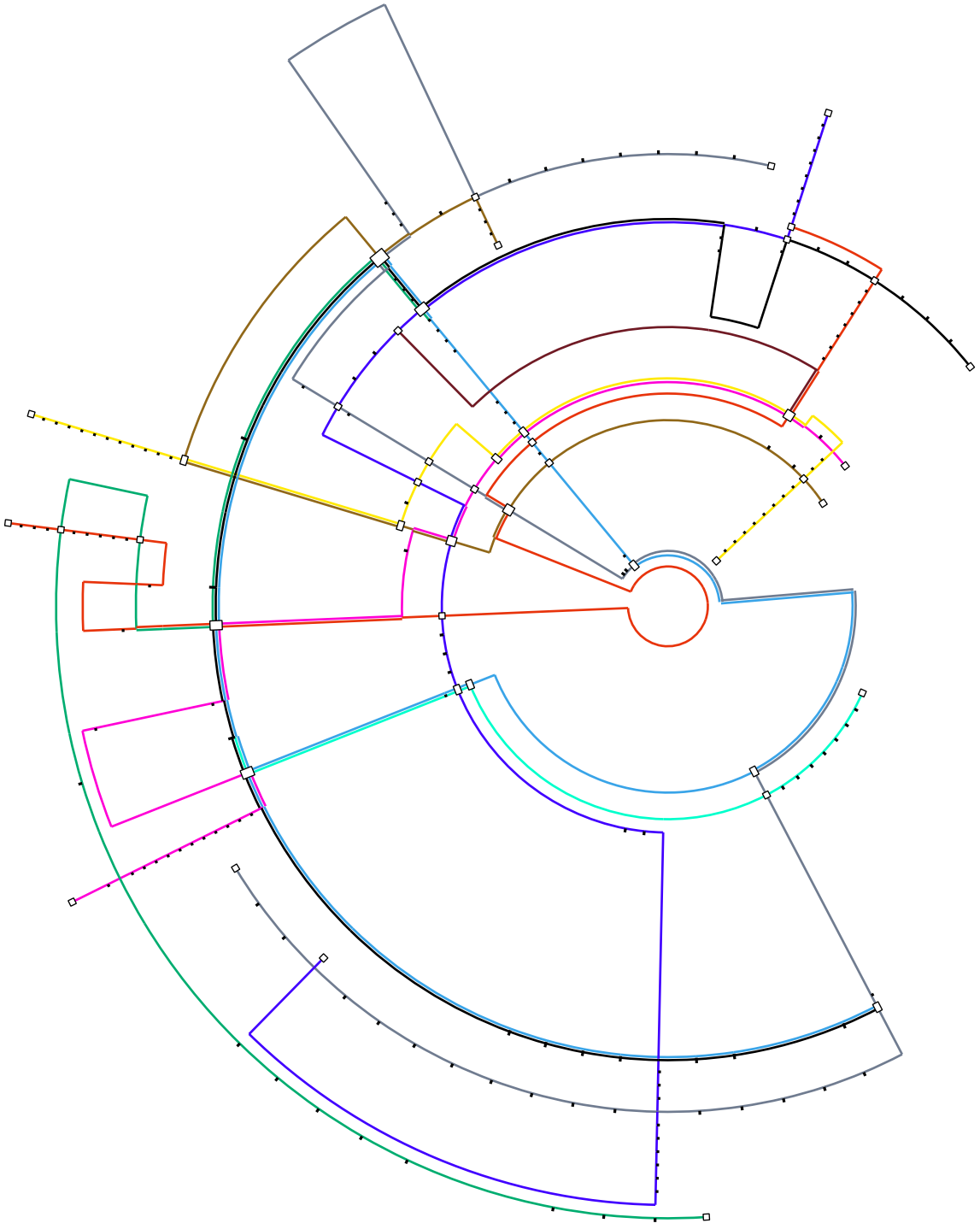
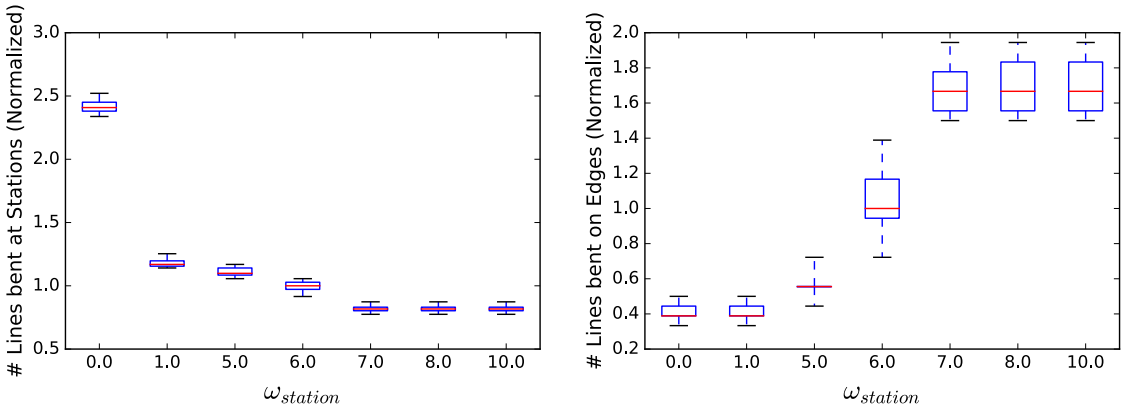


Figure 4.39.: A complete drawing of the London Underground without any manual tuning. Note that assigning colors to the metro lines such that contrast between parallel-running lines is maximized is a problem that is not within the scope of this thesis. Also, minimizing the number of times that metro lines running along the same edges must cross is a separate problem. Thus, these criteria are not optimized in this drawing.



(a) The influence of ω_{station} on the number of lines bending at stations with $\omega_{\text{line}} = 5.0$

(b) The influence of ω_{station} on the number of lines bending on edges with $\omega_{\text{line}} = 5.0$

Figure 4.40.: Plots evaluating good values for ω_{station}

	# Lines bent on Edges		
ω_{station}	Average	Median	Std.Dev.
0.0	0.412	0.389	0.0458
1.0	0.414	0.389	0.0465
5.0	0.562	0.556	0.0527
6.0	1.04	1	0.154
7.0	1.7	1.67	0.128
8.0	1.71	1.67	0.13
10.0	1.71	1.67	0.127

(a) Evaluation of ω_{station}'s influence on the number of lines bending on edges under the condition $\omega_{\text{line}} = 5.0$

	Drawing Height		
ω_{station}	Average	Median	Std.Dev.
0.0	0.908	0.907	0.158
1.0	0.918	0.911	0.151
3.0	1.03	1.02	0.171
5.0	1.04	1.04	0.181
10.0	1.1	1.09	0.198

(b) Evaluation of ω_{station}'s influence on the drawing height. Only instances with $\#anneal = 5000$ are considered.

Table 4.6.

setting $\omega_{\text{station}} = 6.0$ indicates that bends at stations and bends on single-line edges are considered equally bad. Taking this into consideration, we had a look at how the number of metro lines bending on edges in influenced by the choice of ω_{station} (again under the assumption $\omega_{\text{line}} = 5.0$). Figure 4.40b and Table 4.6a show the results, indicating that the number of lines bending on edges in fact rises again significantly when 6.0 or 7.0 is chosen for ω_{station}. It is interesting to note that setting $\omega_{\text{station}} = 1.0$ gets rid of about half of all line-bends at stations without increasing the number of lines bending on edges.

We now first examine the interplay of ω_{station} and the compaction efforts, before looking closer at the parameters influencing the annealing process. Interestingly, the effect described in the following only appears in London's metro map, not in Berlin's drawings. We still consider the effect to be disadvantageous enough to mention it here. Figure 4.41 shows the normalized heights of the produced drawings with different values for ω_{station}, now again from the first dataset as introduced in Table 4.2, but only considering the instances based on the London Underground. While Figure 4.41a includes all measurements, Figure 4.41b only considers those measurements where 5000 annealing steps were made. In Figure 4.41b, the increase in height is clearly visible. Table 4.6b indicates that raising ω_{station} from 0.0 to 10.0 comes with a 12% increase in average normalized drawing height. The p-values resulting from a Wilcoxon-Test can be seen in Table A.7. Clearly, the increase in height when increasing ω_{station} is significant.

	Drawing Height		
#anneal	Average	Median	Std.Dev.
0	1.23	1.22	0.161
10	1.16	1.15	0.153
100	0.998	0.996	0.133
1000	0.859	0.857	0.128
5000	0.815	0.808	0.128

(a) *#anneal*'s influence on the drawing height

	Station-Distance Std.Dev.		
#anneal	Average	Median	Std.Dev.
0	1.97	1.93	0.43
10	1.64	1.61	0.358
100	1.02	0.999	0.198
1000	0.689	0.675	0.13
5000	0.621	0.611	0.12

(b) *#anneal*'s influence on the standard deviation of distances between neighboring stations

Table 4.7.: Evaluation of *#anneal*

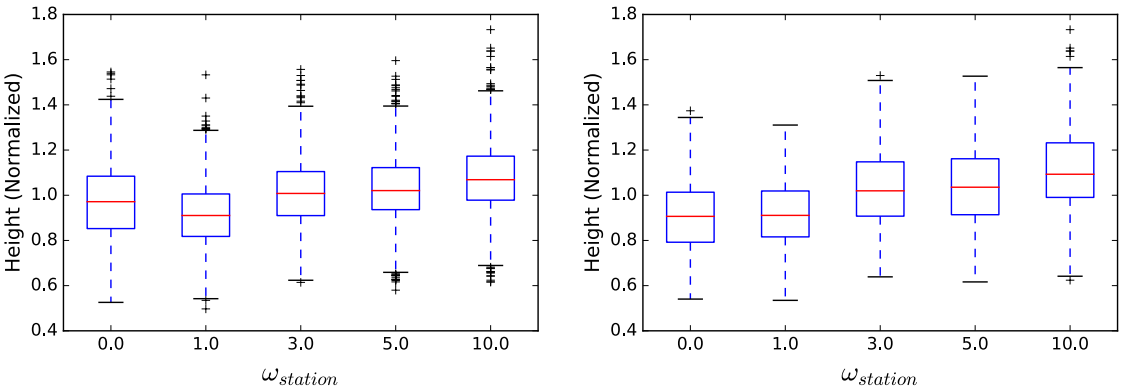
This effect is easily explained with two metro map drawings. Consider Figure 4.42a, which shows a metro map drawing of Montréal. This drawing was created with $\omega_{\text{station}} = 0.0$ (and *#anneal* = 5000). Compare this drawing to the one in Figure 4.42b also showing Montréal, but in a drawing generated with $\omega_{\text{station}} = 7.0$. Clearly, in the latter drawing, sequences of degree-two vertices are drawn in a straight path to avoid bending a metro line at a station. However, by chance some of these sequences, namely the vertices on the red and the blue lines, have been drawn on radial lines. Comparing this to the drawing in Figure 4.42a, it stands out that the same lines tend to 'coil up' in the left part of the drawing. These radial sequences as seen in Figure 4.42b tend to force larger radii when ω_{station} is larger than 0. In fact, the maximum radius in Figure 4.42a is about 10% smaller than the maximum radius in Figure 4.42b, resulting in a drawing with almost 20% less area usage.[10] However, following the recommendations by Maxwell Roberts [Rob12], we assume that a high number of lines bending at stations is worse than a slightly increased area. Thus, we recommend setting $\omega_{\text{station}} \geq \omega_{\text{line}} + \omega_{\text{edge}}$, in our case 7.0.

4.6.3.2. Parameters influencing Metrics

We now take a look at the effects of altering the parameters *#anneal* and *reaugment-percentage*. The first, non-surprising result is that increasing *#anneal*, the number of steps taken during the annealing process reduces the height and the station-distance deviation. Figure 4.43a and Table 4.7a show the (normalized) measurements of the drawings' heights by different values for *#anneal*. Clearly, increasing the number of annealing steps is an effective way of improving the drawing height. With 5000 annealing steps, the normalized height of the drawings is reduced by more than 35% compared to drawings without any annealing. A Wilcoxon-Test reports a p-value of less than 10^{-35} for each pair of measurements, so all differences are statistically significant. A similar picture emerges for the influence of *#anneal* on the standard deviation of distances between neighboring stations, as can be seen in Figure 4.43b and Table 4.7b. Again, increasing the number of annealing steps effectively reduces the standard deviation. In this case, the Wilcoxon-Test reports p-values of less than 10^{-40} for every pair of measurements, thus we may assume significance in every case.

We performed further experiments with 10000 and 20000 annealing steps. However, these experiments did not yield statistically significant differences in terms of height minimization or minimization of the standard deviation of distances between neighboring stations.

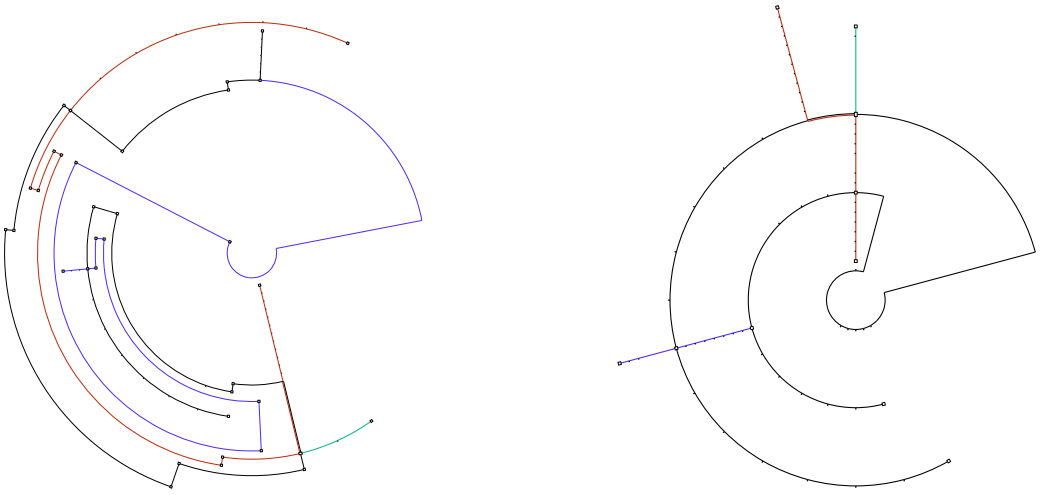
[10] Please note that because of scaling these proportions are not reflected by the depicted drawings.



(a) Considering the full dataset

(b) Considering only runs with 5000 annealing steps

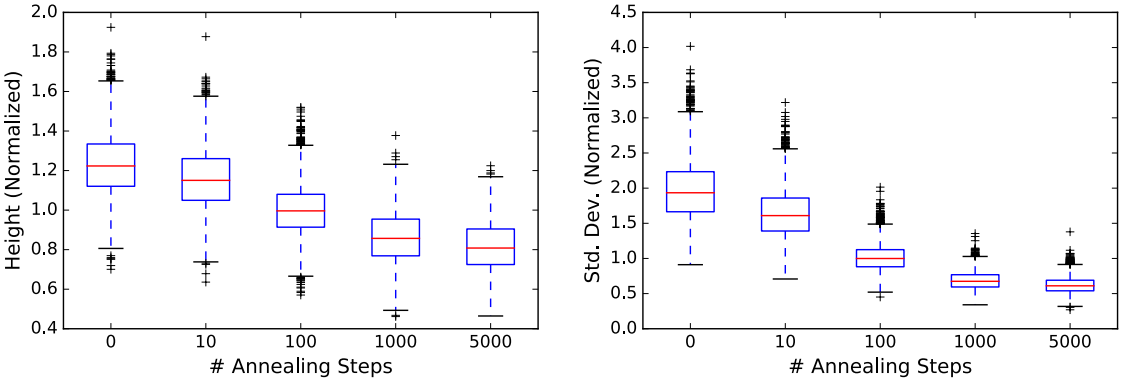
Figure 4.41.: Evaluation of ω_{station}'s influence on the height of the drawing. Only instances based on the London dataset are considered here.



(a) A metro map of Montréal with $\omega_{\text{station}} = 0.0$

(b) A metro map of Montréal with $\omega_{\text{station}} = 3.0$

Figure 4.42.: Example of ω_{station}'s influence on the height of the drawing



(a) *#anneal*'s influence on the drawing height

(b) *#anneal*'s influence on the standard deviation of distances between neighboring stations

Figure 4.43.: Evaluation of *#anneal*

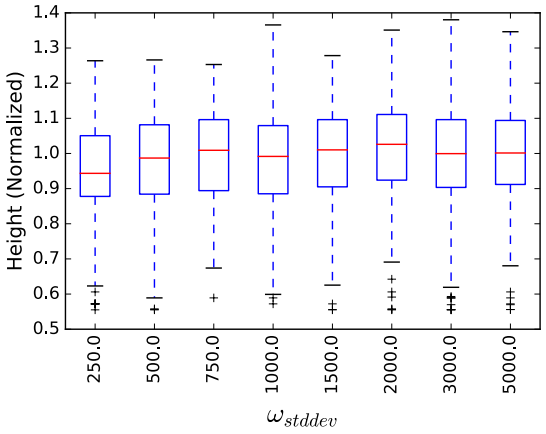


Figure 4.44.: ω_{dist}'s influence on the drawing height

	Drawing Height		
ω_{dist}	Average	Median	Std.Dev.
250	0.948	0.943	0.153
500	0.974	0.987	0.146
750	0.992	1.01	0.137
1000	0.981	0.992	0.15
1500	0.998	1.01	0.138
2000	1.01	1.03	0.151
3000	0.986	1	0.162
5000	0.996	1	0.146

Table 4.8.: ω_{dist}'s influence on the drawing height

The next set of parameters that we evaluate are the weights ω_{height} and ω_{dist}. Since they just weigh up optimizing height against optimizing the standard deviation of distances between neighboring stations, the absolute values of both parameters do not matter. We therefore performed a set of experiments where ω_{height} was set to 2000, and we varied ω_{dist}. The exact parameter space can be found in Table A.3. The first surprising result is that we could not find any statistically significant differences in the normalized standard deviations of distances between neighboring stations when varying ω_{dist}. The exact p-values can be found in Table A.9. Although some of these values are below 0.05, a Dunn-Bonferroni correction must be applied to the data, since we perform pairwise comparisons, resulting in a quadratic number of hypotheses. The corrected p-level of $0.05/32 \approx 0.0016$ is not met by any of the pairs.[11] When considering the height of the drawings, the results are minimally different. Figure 4.44 and Table 4.8 show the normalized drawings' heights for different values of ω_{dist}. On average, there is a 6.5% increase in normalized drawing height when going from $\omega_{\text{dist}} = 250$ to $\omega_{\text{dist}} = 2000$. In fact, this difference is statistically significant even after a Dunn-Bonferroni correction, as can be seen from Table A.10.

Thus, we can conclude that while high values of ω_{dist} seem not to improve optimization of the station distance standard deviation, they might minimally deteriorate height minimization. Therefore, it seems prudent to choose a rather small value for ω_{dist}. This result might indicate that optimizing drawing height and optimizing station-distance standard deviation is not mutually impeditive.

Finally, we examine the influence of the parameter *reaugment-percentage*, specifying for which fraction of the graph's faces the augmentation is changed in every step of the annealing process. For this, we computed an experiment with the parameter space listed in Table A.4. Figure 4.45a and Table 4.9a present the data generated in this experiment. Most of the pairwise differences are not statistically significant, as can be seen from Table A.11. However, the values 0.02, 0.03, 0.04 and 0.06 do produce drawing heights that are significantly lower than the largest value in the set, 0.3. When we compare this with the normalized time needed to perform 1000 annealing steps at the various *reaugment-percentage* values, shown in Figure 4.45b and Table 4.9b, we can see that the

[11] The reader might wonder why we did not apply a Dunn-Bonferroni correction to the other sets of p-values. In fact, one would have to do so. However, we did not want to complicate things by specifying a different significance level in each case. Since the Dunn-Bonferroni correction factor would be below 100 in each of our cases, and in all cases where we would have had to apply such a correction, all significant p-values were below 10^{-5}, all statements about significance still retain their validity.

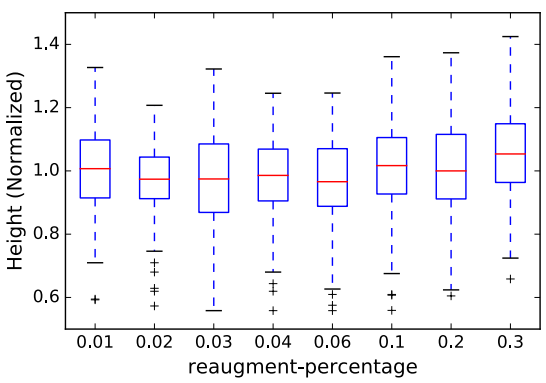
	Drawing Height		
	Average	Median	Std.Dev.
0.01	0.999	1.01	0.132
0.02	0.97	0.974	0.123
0.03	0.967	0.975	0.16
0.04	0.978	0.986	0.139
0.06	0.965	0.966	0.157
0.10	1.01	1.02	0.146
0.20	1	1	0.161
0.30	1.06	1.05	0.149

(a) *reaugment-percentage* influence on the normalized drawing height

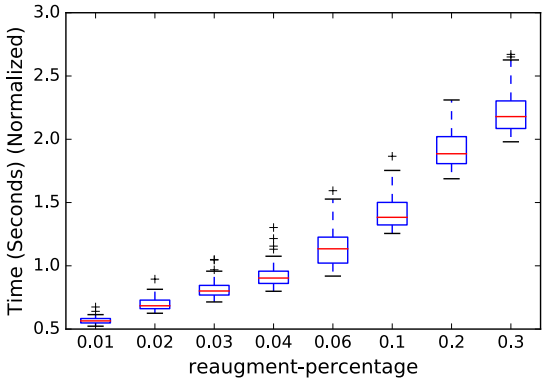
	Time spent in Annealing		
	Average	Median	Std.Dev.
0.01	0.568	0.565	0.026
0.02	0.697	0.684	0.0465
0.03	0.818	0.801	0.0671
0.04	0.919	0.903	0.0825
0.06	1.14	1.13	0.139
0.10	1.42	1.38	0.121
0.20	1.92	1.89	0.158
0.30	2.21	2.18	0.16

(b) *reaugment-percentage* influence on the normalized time spent in simulated annealing at 1000 iterations

Table 4.9.



(a) *reaugment-percentage* influence on the drawing height



(b) *reaugment-percentage* influence on time spent in the annealing process

Figure 4.45.: Evaluating the effect of *reaugment-percentage*

necessary time increases drastically with the value of *reaugment-percentage*. Thus, 0.03 seems a good value for *reaugment-percentage* to us.

At last, we take a quick look at the performance of our implementation. Please note that our implementation is meant as a proof of concept. We did not perform any optimizations, and several data structures (most notably the data structure for holding the graph's planarity information) are implemented rather inefficiently. Optimizing these, respectively implementing complex but efficient data structures, was outside the scope of this thesis.

The time needed to create a metro drawing is dominated by the time spent in the annealing process. In fact, the time spent for all other computations can be neglected for reasonable values of *#anneal*, thus we focus on the time spent in the annealing process.

Table 4.10 lists the time spent in the annealing process as well as the time per annealing iteration for various values of *#anneal* (only considering London instances, with a *reaugment-percentage* of 0.03). The first thing that stands out is the fact that the average time per iteration increases slightly with the number of iterations.[12] Although the increase is small,

[12]With the measurement for 10 iterations being an outlier; it is plausible that the time spent initializing the simulated annealing process dominates these timings.

	Time (in Seconds) spent in Annealing			Time (in Seconds) spent per Iteration		
#anneal	Average	Median	Std.Dev.	Average	Median	Std.Dev.
10	7.2	7.2	0.8	0.72	0.716	0.0763
100	63.9	62.1	5.9	0.639	0.621	0.0585
1000	630.9	627.6	52.2	0.631	0.628	0.0522
5000	3553.8	3441.9	550.5	0.711	0.688	0.11
10000	7154.9	6762.9	1274.3	0.715	0.676	0.127
20000	14497.4	13816.5	2425.3	0.725	0.691	0.121

Table 4.10.: Time requirements of the annealing process for different values of *#anneal* (at a *reaugment-percentage* of 0.03)

it is in fact statistically significant, as can be seen from the p-values in Table A.12.[13] This suggests that the time complexity is superlinear in the number of annealing iterations, a fact that can most likely be explained by the inefficiencies in our implementation. In absolute terms, a drawing with 5000 annealing steps (which is the value we recommended above) can be produced in about one hour.

4.6.4. Subjective Evaluation

We now present our subjective evaluation of the drawing results. Figure 4.39 shows a drawing of the London Metro that was created with $\omega_{\text{station}} = 7.0$, $\omega_{\text{line}} = 5.0$ and 5000 annealing steps. The drawing is passably compact. Note that assigning colors to metro lines such that contrast between parallel-running metro lines is maximized is its own $\mathcal{NP}$-hard problem, the solution of which is not within the scope of this thesis. Aside from such contrast problems, most lines are reasonably easy to follow visually. Figure A.1 is another drawing of the London Metro, with the same set of parameters.

However, there are a number of problems we identified:

Radial Sequences

In Figure 4.39, the center-wards end of the yellow metro line consists of a sequence of degree-two vertices. This sequence is drawn radially, pushing everything else outwards. If this sequence was drawn orbitally, a lot of space could be saved towards the center of the drawing. However, to control how such sequences are drawn, it would be necessary to introduce some global notion of direction into the flow network computing the orthogonal representation. It is not clear how that could be achieved.

Station-Bends

Figure A.2 shows another Metro Map of the London Underground, this time with $\omega_{\text{station}} = 0.0$. The measurements we took suggested that setting low values for ω_{station} might result in more compact drawings. However, the zigzagging metro lines especially in the upper and lower parts create a lot of visual disturbance when trying to follow the lines visually. Also, drawings with $\omega_{\text{station}} = 0.0$ tend to have very long orbital segments, which we feel make it hard to follow the metro lines visually. We therefore do not recommend turning off the station-bend minimization.

[13] Note that even though some of the pairs do not reach pairwise significance, we may assume that the overall increase is significant.

Choice of the Center Face

Something that is not directly reflected by any metric is the effect that the choice of the center face has on the results. In this evaluation, we chose the center face randomly for every instance, hoping that any effect that this choice has averages out over the number of measurements. Looking at many drawings, we got the impression that this choice has in fact a major influence on the quality of the drawings. However, we assume that when creating a circular metro map of an actual metro network, the user might want to select the center of the drawing manually. Thus, this should not present a problem in practice.

Spacing based on the Number of Metro Lines

Something that our approach does not yet take into consideration when computing the minimum space between two vertices or edges is the number of metro lines running via the edges or vertices. However, particularly for stations, the number of metro lines connected to them varies a lot in practice, resulting in metro stations that are drawn with rather large symbols, and stations depicted by rather small symbols. It would be desirable to differentiate between them when computing minimum distances.

5. Topology-Shape-Metrics on a Cylinder

5.1. Introduction

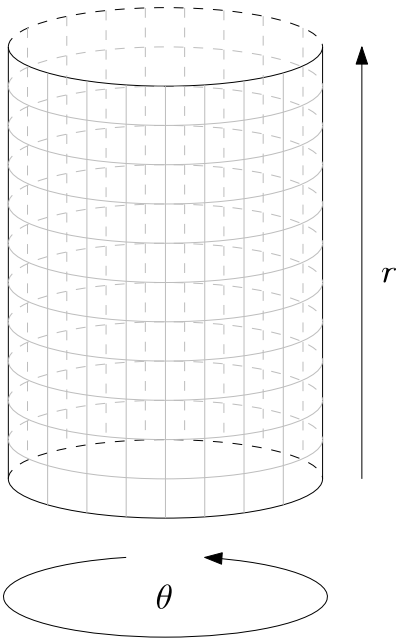
In this thesis, we try to draw graphs ortho-radially. One primary concern in doing so is minimizing the number of bends in the drawing. In Chapter 4, we demonstrated an approach that first uses bend-minimization techniques for an orthogonal drawing and then transforms this orthogonal drawing into an ortho-radial one. However, the obvious question is whether this bend-minimization can also directly be done in an ortho-radial setting.

This chapter briefly looks into this question. We first illustrate differences between orthogonal and ortho-radial bend minimization. Then, we present an intuitive adaption of the *Topology-Shape-Metrics* framework in Section 5.2. In Section 5.3 we then illustrate why these changes are not enough to ensure that the result is always embeddable, presenting some non-trivial problems that must be solved if the *Topology-Shape-Metrics* framework was to be transferred to the ortho-radial case.

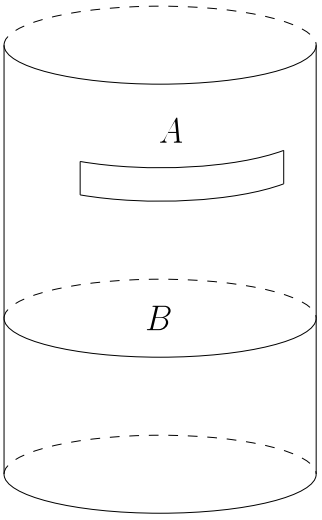
We were able to find surprisingly little research related to bend-minimization for graphs embedded on cylinders. Hasheminezhad et al. [HHT09] not only present something similar to an ortho-radial representation (see below), but also give a way of deciding for paths, cycles and certain combinations of paths whether the graphs can be embedded on a cylinder adhering to a given ortho-radial representation. Hasheminezhad et al. [HHMT10] give a characterization of graphs that have a straight-line embedding on cylinders, but only for cubic rectangulated graphs. Tamassia and Tolis [TT91] give a characterization of graphs admitting a visibility representation on a cylinder, where vertices are represented by intervals along the axis of the cylinder, and edges are represented by overlapping intervals. However, we were not able to find any prior work examining bend-minimization for graphs embedded on cylinders.

Drawing graphs ortho-radially, as defined in Definition 2.3 of Section 2.1, and as already discussed in Chapter 4, is equivalent to drawing graphs on a grid embedded on a cylinder, on which one set of grid lines runs parallel to the cylinder's rims, and one set of grid lines runs parallel to the cylinder's axis. We call this a *cylindrical grid*. Such a cylindrical grid can be seen in Figure 5.1a. Even though this point of view is equivalent to an ortho-radial grid,[1] for some illustrations throughout this chapter, the cylinder model makes matters

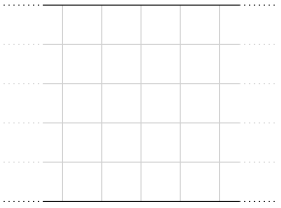
[1] Actually, this is not entirely true, since drawing on an ortho-radial grid allows for edges to be routed through the center S, which would correspond to an edge crossing the rim of the cylinder. However, we did not exploit this possibility, and thus we may assume equivalence for our purposes.



(a) A grid on a cylinder



(b) Two non-homotopic types of cycles on a cylinder



(c) The grid from Figure 5.1a unrolled

Figure 5.1.: Illustrations of embedding graphs on a cylindrical grid

more clearly. Therefore, we sometimes use this drawing style. Occasionally, we simplify drawings even further by unrolling the grid into the plane, as illustrated in Figure 5.1c. Here, the dotted grid lines indicate at what point the grid from Figure 5.1a was cut open.

When drawing a graph on an ortho-radial grid (or a cylindrical grid), there are two special faces: The center face, containing the origin of the ortho-radial grid, and a face that is the outer face. On the cylinder, these two faces contain the upper and lower rims of the cylinder. This differs from the embedding in an orthogonal grid, in which just an outer face is fixed. It is however possible to select the same face for both, thus having a face that contains both the lower and the upper rim of the cylinder. In this case, it would be possible to cut open the cylinder from one rim to the other without cutting the embedded graph, and flattening the resulting surface into a plane. Conversely, the embedding would be equivalent to orthogonally embedding the graph into the plane and then rolling it up on a cylinder. Since this is well researched, in this section we only concern ourselves with embeddings in which two different faces have been selected for upper and lower rim.

More to the point, there are two non-homotopic ways of drawing a cycle on a cylinder: Type A is having both rims on one side, Type B is separating both rims from each other. Figure 5.1b shows examples of both types of cycles. While a cycle of Type A can be embedded in the plane with the same shape it has on the cylinder (i.e. the same bends on its edges, and the same angles between the edges of the respective vertices), Type B cycles cannot. This is easy to see using the example in Figure 5.1b: Clearly, when embedding a cycle in the plane, it must use bends. We therefore only care for graphs that contain Type B cycles.

5.2. Adapting Topology-Shape-Metrics

Since Tamassia's Topology-Shape-Metrics framework is the standard technique for minimizing edge-bends in an orthogonal drawing, the obvious idea is to try to adapt it to the ortho-radial, respective cylindrical, case. In this section, we present the intuitive way of doing so. Later, we show which problems arise in this case.

Because of the different topology of a cylinder (when compared to a plane), the rotation which cycles in the drawing may have changes: While in an orthogonal drawing, all (clockwise) cycles have rotation 4, this is not the case when drawing ortho-radially or on a cylinder. All Type A cycles still have rotation 4, but any Type B cycle has rotation 0 in a valid drawing. See Figure 5.1b again for two examples. If we assume that G is connected, every inner face is always a Type A cycle, thus the idea on which Tamassia's flow network (and the orthogonal representation) is based, namely that every face must be assigned a rotation of 4, still holds for the inner faces. However, the center and the outer face are cycles of Type B. For them, we must adapt the flow network (and the requirements of the orthogonal representation) in such a way that they must be assigned a rotation of 0. With this, we define an *ortho-radial representation* analogous to the orthogonal representation from Definition 3.1:

Definition 5.1 (Ortho-Radial Representation)**.** *Given a graph $G = (V, E)$, and a planar embedding of G resulting in the set of faces F. An ortho-radial representation is a function $H\colon F \to \{E \times \{0,1\}^\star \times \{90, 180, 270, 360\}\}^{|F|}$, assigning to every $f \in F$ a circularly ordered list. All elements of these lists have the form (e, s, a), where*

- *e is an edge of G,*
- *s is a binary string,*
- *a is an integer in the set $\{90, 180, 270, 360\}$.*

Likewise analogous to the definition in Section 3.1, we say such an ortho-radial representation is *valid* if:

(1) The **e** lists for all faces conforms to the planar embedding of G, i.e.

$$\forall f \in F\colon ((e_i, \cdot, \cdot), (e_j, \cdot, \cdot)) \in H(f) \Rightarrow (e_i, e_j) \in f$$

(2) Let $(e, s, a) \in H(f)$ and $(e, s', a') \in H(f')$ (note that e is the same in both entries), then s can be obtained from s' by reversing the string and exchanging 0's and 1's

(3) All faces are properly closed, i.e.

$$\forall f \in F\colon \sum_{(e,s,a)\in H(f)} \mathrm{rot}(s, a) = \begin{cases} 0 & \text{if f is the outer or center face} \\ 4 & \text{if f is an internal face} \end{cases}$$

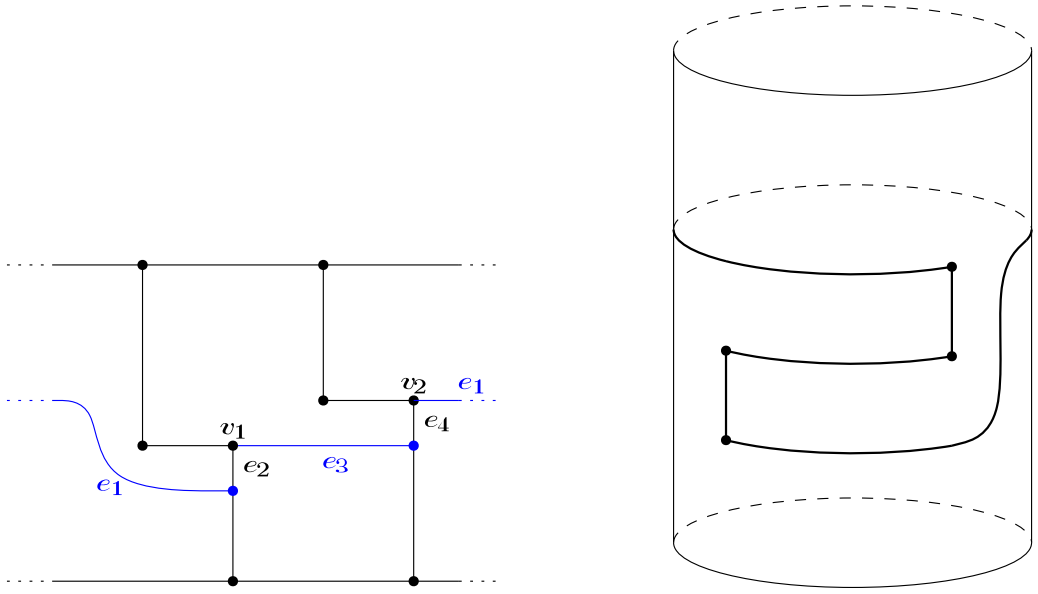
(4) For every vertex, the sum of angles between its incident edges is 360°.

The only change of this definition compared to the definition of an orthogonal representation is condition (3), where the outer and the center face are required to have rotation 0.

To compute such an ortho-radial representation, we again use a flow network similar to the one used to compute an orthogonal representation. Refer to Section 3.1.2 for details on how that flow network was built. The only changes we have to make are the demands of the vertices representing the outer and center face. Instead of having a surplus of 4, as in the orthogonal case, both vertices have a demand of 0.

5.3. Drawability

We now examine the question of whether such an ortho-radial representation is drawable on a cylinder, respectively in an ortho-radial way. Drawability here means that we



(a) A rectangulation that is not drawable. Blue parts are added for rectangulation. One edge is drawn curved because of the impossibility to draw this graph. Dotted edges wrap around the cylinder.

(b) Illustration of a circular radial dependency

Figure 5.2.: Illustrations of drawability problems

can find functions f_V and f_E as defined in Definition 2.3 such that no crossings occur, i.e. $\forall e, e' \in E \colon \forall s \in (0,1) \colon f(e,s) \neq f(e',s)$ and $\forall v, v' \in V : f_V(v) \neq f_V(v')$

In the case of an orthogonal representation, Tamassia shows that the result is always drawable in a constructive way: He first demonstrates a way to subdivide all faces into rectangles (cf. Section 3.1.3), and then refers to a rectangle compaction algorithm that always results in a valid drawing. However, this approach does not necessarily result in a drawable ortho-radial representation, as Figure 5.2a demonstrates: Here, both blue edges e_1 and e_2 and the blue vertices are added to rectangulate the graph. However, because of the circular nature of the θ-dimension, these edges introduce a circular dependency on the radial coordinates of v_1 and v_2. The placement of e_1 forces v_1 to have a larger radial coordinate than v_2, while e_3 forces v_2 to have a larger radial coordinate than v_1.

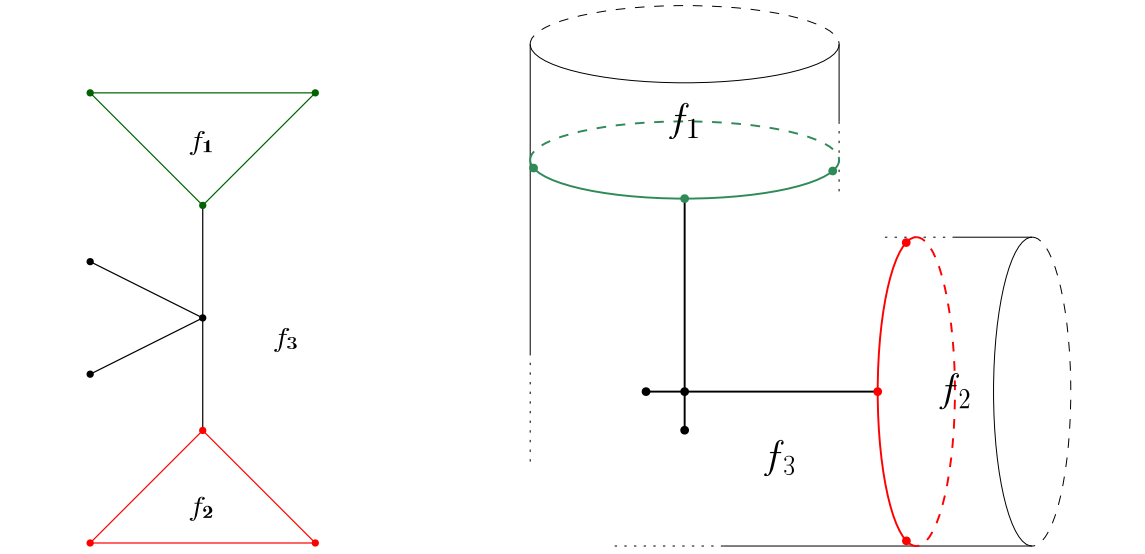
Thus, we must have a closer look at drawability in the ortho-radial case. We demonstrate two problems which can cause ortho-radial representations to be not drawable.

5.3.1. Problems

We now illustrate two kinds of problems that impede drawability in the ortho-radial case.

Problem 1: Finding a wrapping rotation

For the first of the two problems, consider the graph depicted in Figure 5.3a, and let f_1 be the center and f_2 the outer face. The bend-optimal ortho-radial representation of this graph is bend-free: The red and green cycles are wrapped around the cylinder without any bend, and the black path is laid out in between. However, because of the the two additional black vertices cause the path between the two cycles must bend. Consider the drawing in Figure 5.3b, illustrating a bend-free ortho-radial representation where f_1 is the center and f_2 is the outer face. Here, faces f_1 and f_2 have rotation 0, while f_3 has rotation 4. Thus, this is a valid ortho-radial representation. However, this can obviously not be drawn on a cylinder, since the two cycles around f_1 and f_2 are twisted by 90° against each other, and thus at least one of them has no edge running around the cylinder.



(a) A graph the optimal ortho-radial representation of which is twisted and not drawable

(b) Illustration of a non-drawable ortho-radial representation

Figure 5.3.: Illustrations of drawability problems

Problem 2: Circular Radial Dependencies

The second problem we look into is depicted in Figure 5.2b: Here, the curvy line should actually be a straight edge. Looking at the cycle, it has two bends right, and two bends left, thus both the upper and the lower face have a valid rotation of 0. However, since when traversing all edges of the cycle, one never moves toward the lower rim (or never toward the upper rim, depending on the direction of traversal), no valid r-coordinates can be assigned here.

5.4. Conclusion

We have presented the intuitive way of transferring the bend-minimization technique used in Tamassia's Topology-Shape-Metrics framework to an ortho-radial setting and have shown that in this setting problems arise which are not existent in the orthogonal case.

We pointed out two problems that we perceive to be the main problems impeding drawability, and we think that overcoming these problems results in an ortho-radial representation that is always drawable. We have some ideas as to mitigate the problems. Roughly speaking, Problem 1 presented above corresponds to a circulation in the flow network that is used to compute the ortho-radial representation. We hope that by either prohibiting circulations in this flow network, or by searching the residual flow network for an (bend-optimal) circulation in the opposite direction, we can solve this problem. However, further research is needed in this area.

6. Conclusion

This thesis examines ways to draw metro maps using an ortho-radial drawing style, resulting in a drawing employing concentric circles and radial spokes algorithmically. Prior to this thesis, the only known algorithm was limited to rather restricted instances (cf. Fink et al. [FLW14]).

We formalized the problem in a way that models the optimization problem according to the quality criteria for metro maps identified by research in the information visualization and psychology communities.

We then presented an algorithmic framework based on the Topology-Shape-Metrics framework by Roberto Tamassia, which is capable of producing metro map drawings even for complex instances, as we demonstrated in the experimental evaluation. The framework is able to compute an orthogonal representation that is tailored to the user's preferences regarding the question in which places bends should be located if necessary. We not only modified the framework to account for the peculiarities of drawing usable metro maps, but also added steps to transform the orthogonal representation into an ortho-radial drawing that actually takes advantage of the circular dimension. As part of our modification of Tamassia's framework, we presented a compaction technique that improves on the technique suggested by Tamassia in terms of area minimization as well as flexibility.

In our experimental evaluation, not only did we show that it is possible to efficiently produce reasonable metro map drawings, but we were also able to describe the effects that different combinations of weighting parameters have on the drawings. With this knowledge, we are able to adapt our drawings to different requirements and give recommendations on how to parameterize our framework.

Furthermore, we were able to show $\mathcal{NP}$-hardness for the subproblems of compacting an ortho-radial representation on the one hand and to reduce the maximum degree of the graph to 4 in such a way that the number of bends in the resulting drawing is minimal on the other hand. This justifies using non-optimal approaches for solving these subproblems.

While the drawings produced by our framework might not yet be suitable for being used as a metro map without any further manual modification, we are positive that drawings produced by our framework can serve as support for human designers trying to produce circular metro maps. We also hope that future research, as outlined below, might further improve the drawings up to the point where they can immediately compete with hand-made drawings.

We are positive that the ability to rapidly produce circular metro maps with certain desired properties supports research efforts such as user studies looking into the usability of circular metro maps.

6.1. Future Work

We think that future work in improving the framework presented in Chapter 4 should primarily focus on finding ways to globally control the relative layout of long sequences of degree-two vertices. As discussed in Section 4.6.4, the fact that many such sequences are drawn radially in the center of the drawing is the most severe impediment to producing compact drawings. Also, a more in-depth evaluation of good cooling schedules for the simulated annealing process would most likely be worthwhile, together with a more efficient implementation of the supporting data structures; We see a lot of room for improvement in this step. Since we are using a black-box metaheuristic that includes a way of exploring a solution neighborhood at this step, it would also be thinkable to swap the simulated annealing technique for other metaheuristics that are based on exploring a solution neighborhood in a similar way, e.g. evolutionary algorithms.

Also, while we could identify the interactions of weighting parameters in Section 4.6.3, we did not have sufficient information on the optimal relation between the different ways that metro lines can bend. It would be beneficial to conduct a user study to evaluate the usability of metro map drawings resulting from different parameter configurations in practice.

Aside from further work on the presented framework, the field of carrying out bend minimization in a drawing embedded on a cylinder, as briefly outlined in Chapter 5, seems to present a lot of open questions. We intend to focus future research on finding ways of mitigating the drawability problems discussed.

An interesting problem that came up, but which went beyond the scope of this thesis, is whether it is possible to identify which face should be chosen as center face to optimize a given criterion in a smarter way than just trying out all faces.

Finally, to actually produce complete metro maps, additional problems must be solved, most notably the problem of placing station labels in the drawing. While there are some approaches of doing so for drawing styles working with straight lines at various angles (cf. Section 1.2), these techniques are probably not applicable to a circular drawing style. To produce ready-to-use metro map drawing algorithmically, a solution to this problem must be found.

Bibliography

[AH06] Silvania Avelar and Lorenz Hurni. On the design of schematic transport maps. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 41(3):217–228, 2006.

[AVL85] E.H.L. Aarts and P.J.M. Van Laarhoven. A new polynomial-time cooling schedule. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 206–208, 1985.

[Bar80] DJ Bartram. Comprehending spatial information: The relative efficiency of different methods of presenting information about bus routes. *Journal of Applied Psychology*, 65(1):103, 1980.

[BETT98] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR, 1998.

[Bor89] Robert F Bornstein. Exposure and affect: Overview and meta-analysis of research, 1968–1987. *Psychological bulletin*, 106(2):265, 1989.

[dBK10] Mark de Berg and Amirali Khosravi. Optimal binary space partitions in the plane. In *Computing and Combinatorics*, pages 216–225. Springer, 2010.

[FHN+13] Martin Fink, Herman Haverkort, Martin Nöllenburg, Maxwell Roberts, Julian Schuhmann, and Alexander Wolff. Drawing metro maps using bézier curves. In *Graph Drawing*, pages 463–474. Springer, 2013.

[FLW14] Martin Fink, Magnus Lechner, and Alexander. Wolff. Concentric metro maps. In Maxwell J. Roberts and Peter Rodgers, editors, *Abstracts of the Schematic Mapping Workshop 2014*, 2014.

[FP13] Martin Fink and Sergey Pupyrev. Metro-line crossing minimization: Hardness, approximations, and tractable cases. In *Graph Drawing*, pages 328–339. Springer, 2013.

[Gar94] Ken Garland. *Mr Beck's Underground Map*. Capital Transport, 1994.

[GJ79] Michael R Garey and David S Johnson. *Computers and intractability: a guide to NP-completeness*. WH Freeman New York, 1979.

[GPB15] Riccardo Gallotti, Mason A. Porter, and Marc Barthelemy. Information measures and cognitive limits in multilayer navigation. *arXiv preprint arXiv:1506.01978*, 2015.

[GT01] Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*, 31(2):601–625, 2001.

[HHMT10] Mahdieh Hasheminezhad, S. Mehdi Hashemi, Brendan D. McKay, and Maryam Tahmasbi. Rectangular-radial drawings of cubic plane graphs. *Computational Geometry*, 43(9):767–780, November 2010.

[HHT09] Mahdieh Hasheminezhad, S. Mehdi Hashemi, and Maryam Tahmasbi. Ortho-radial drawings of graphs. *Australasian Journal of Combinatorics*, 44:171–182, 2009.

[HMdN06] Seok-Hee Hong, Damian Merrick, and Hugo AD do Nascimento. Automatic visualisation of metro maps. *Journal of Visual Languages & Computing*, 17(3):203–224, 2006.

[HN15] Jan-Henrik Haunert and Benjamin Niedermann. An algorithmic framework for labeling network maps. *arXiv preprint arXiv:1505.00164*, 2015.

[Hsu80] Min-Yu Hsueh. Symbolic layout and compaction of integrated circuits. 1980.

[HW02] William C Hahn and Robert Allan Weinberg. *A subway map of cancer pathways*. Nature Publishing Group, 2002.

[Law76] Eugene L Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 1976.

[Lic82] David Lichtenstein. Planar formulae and their uses. *SIAM journal on computing*, 11(2):329–343, 1982.

[NW11] M. Nollenburg and A. Wolff. Drawing and Labeling High-Quality Metro Maps by Mixed-Integer Programming. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):626–641, May 2011.

[Nö05] Martin Nöllenburg. Automated drawing of metro maps. Technical Report 2005-25, Fakultät für Informatik, Universität Karlsruhe, http://digbib.ubka.uni-karlsruhe.de/volltexte/1000004123, 2005.

[Nö14] Martin Nöllenburg. A survey on automated metro map layout methods. In *Schematic Mapping Workshop*, Essex, UK, April 2014.

[OPL08] Mark Ovenden, Julian Pepinster, and Peter B Lloyd. *Paris Metro style in map and station design*. Capital Transport Pub., 2008.

[Pat99] Maurizio Patrignani. On the complexity of orthogonal compaction. In *Algorithms and Data Structures*, pages 56–61. Springer, 1999.

[RNC16] Maxwell J Roberts, Elizabeth J Newton, and Maria Canals. Radi (c) al departures: Comparing conventional octolinear versus concentric circles schematic maps for the berlin u-bahn/s-bahn networks using objective and subjective measures of effectiveness. *Information Design Journal*, 2016.

[RNL+13] Maxwell J. Roberts, Elizabeth J. Newton, Fabio D. Lagattolla, Simon Hughes, and Megan C. Hasler. Objective versus subjective measures of Paris Metro map usability: Investigating traditional octolinear versus all-curves schematics. *International Journal of Human-Computer Studies*, 71(3):363–386, 2013.

[Rob12] Maxwell J. Roberts. *Underground Maps Unravelled - Explorations in Information Design*. Roberts, Maxwell J., 2012.

[SGSK01] Elmer S Sandvad, Kaj Grønbæk, Lennert Sloth, and Jørgen Lindskov Knudsen. A metro map metaphor for guided tours on the web: the webvise guided tour system. In *Proceedings of the 10th international conference on World Wide Web*, pages 326–333. ACM, 2001.

[SRB+05] Jonathan M Stott, Peter Rodgers, Remo Aslak Burkhard, Michael Meier, and Matthias Thomas Jelle Smis. Automatic layout of project plans using a metro map metaphor. In *Information Visualisation, 2005. Proceedings. Ninth International Conference on*, pages 203–206. IEEE, 2005.

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

[Tei02] Steven L Teig. The x architecture: not your father's diagonal wiring. In *Proceedings of the 2002 international workshop on System-level interconnect prediction*, pages 33–37. ACM, 2002.

[TT91] R. Tamassia and I. Tollis. Representations of Graphs on a Cylinder. *SIAM Journal on Discrete Mathematics*, 4(1):139–149, February 1991.

[Wol96] Alexander Wolff. The map-labeling bibliography. *http://i11www. ira. uka. de/map-labeling/bibliography/*, 1996.

[WTH+13] Hsiang-Yun Wu, Shigeo Takahashi, Daichi Hirono, Masatoshi Arikawa, Chun-Cheng Lin, and Hsu-Chun Yen. Spatially efficient design of annotated metro maps. In *Computer Graphics Forum*, volume 32, pages 261–270. Wiley Online Library, 2013.

[WTLY12] Hsiang-Yun Wu, Shigeo Takahashi, Chun-Cheng Lin, and Hsu-Chun Yen. Travel-route-centered metro map layout and annotation. In *Computer Graphics Forum*, volume 31, pages 925–934. Wiley Online Library, 2012.

Appendix

A. Experimental Results

Parameter Name	Values	# of measurements per value
City	London	8000
ω_{station}	3.0	8000
ω_{edge}	1.0	8000
ω_{line}	$0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 7.5, 10.0$	1000
ω_{height}	1500	8000
ω_{dist}	100	8000
#anneal	0	8000
reaugment-percentage	0.03	8000

Table A.1.: Parameter Space of the dataset used to evaluate reasonable values for ω_{line}. Note that since the number of line bends is fixed before any simulated annealing takes place, the parameters pertaining simulated annealing are irrelevant.

Parameter Name	Values	# of measurements per value
City	London	6000
ω_{station}	$0.0, 1.0, 5.0, 6.0, 7.0, 12.0, 17.0$	1000
ω_{edge}	1.0	6000
ω_{line}	5.0	6000
ω_{height}	1500	6000
ω_{dist}	100	6000
#anneal	0	6000
reaugment-percentage	0.03	6000

Table A.2.: Parameter Space of the dataset used to evaluate reasonable values for ω_{station} under the condition $\omega_{\text{line}} = 5.0$. Note that since the number of line bends is fixed before any simulated annealing takes place, the parameters pertaining simulated annealing are irrelevant.

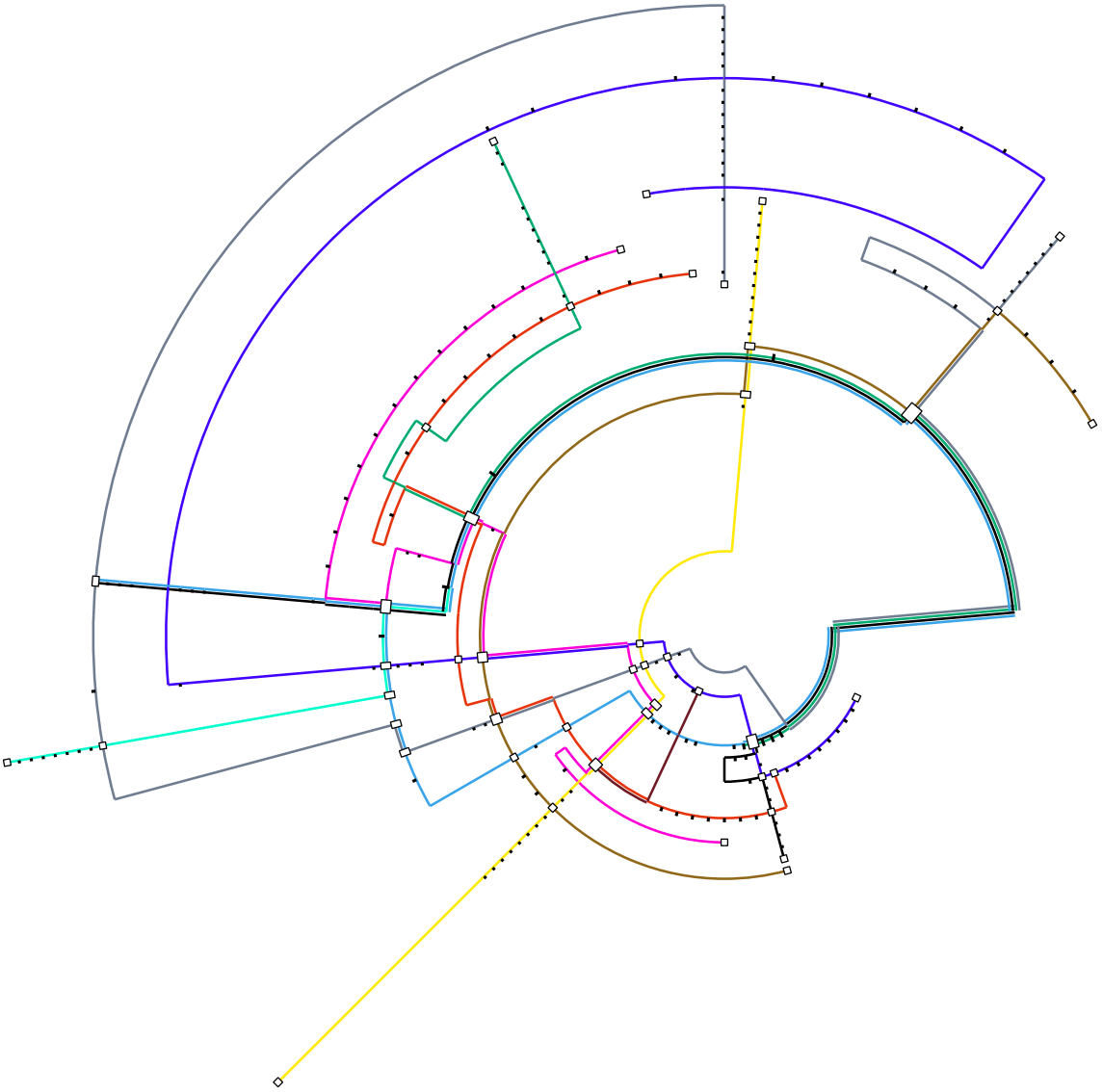


Figure A.1.: A complete drawing of the London Subway without any manual tuning. Note that assigning colors to the metro lines such that contrast between parallel-running lines is maximized is a problem that is not within the scope of this thesis. Also, minimizing the number of times that metro lines running along the same edges must cross is a separate problem. Thus, these criteria are not optimized in this drawing.

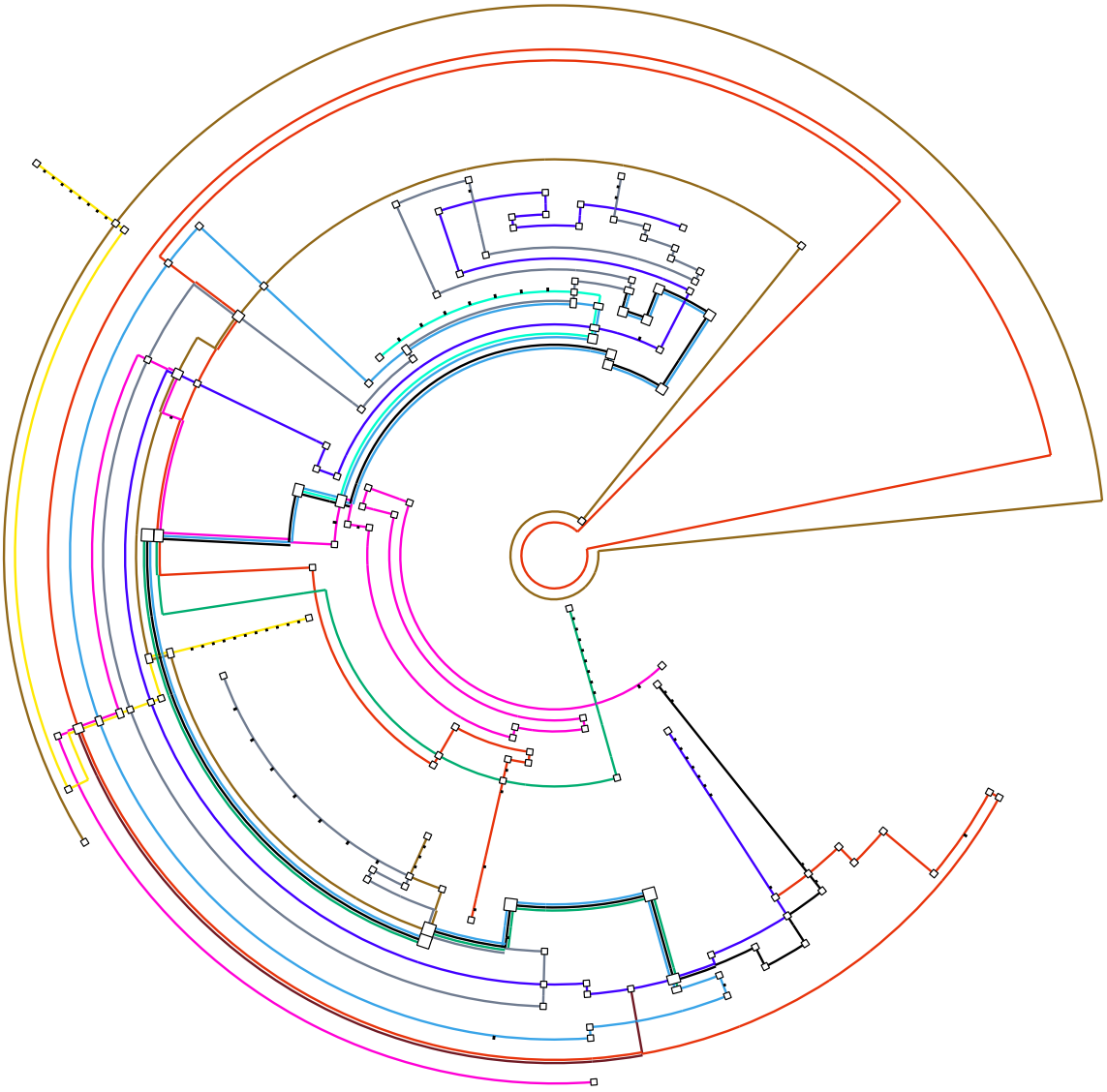


Figure A.2.: A complete drawing of the London Subway without any manual tuning. In this drawing, we chose $\omega_{\text{station}} = 0.0$.

Parameter Name	Values	# of measurements per value
City	London	1600
ω_{station}	7.0	1600
ω_{edge}	1.0	1600
ω_{line}	5.0	1600
ω_{height}	2000	1600
ω_{dist}	$250, 500, 750, 1000, 1500, 2000, 3000, 5000$	200
#anneal	1000	1600
reaugment-percentage	0.03	1600

Table A.3.: Parameter Space of the dataset used to evaluate reasonable values for ω_{dist}. Note that since only the relation between ω_{height} and ω_{dist} influences the simulated annealing process, it is reasonable to fix ω_{height} to one value.

Parameter Name	Values	# of measurements per value
City	London	1700
ω_{station}	7.0	1700
ω_{edge}	1.0	1700
ω_{line}	5.0	1700
ω_{height}	2000	1700
ω_{dist}	200	1700
#anneal	1000	1700
reaugment-percentage	$0.01, 0.015, 0.02, 0.025, 0.03, 0.035, 0.04,$ $0.05, 0.06, 0.08, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6$	100

Table A.4.: Parameter Space of the dataset used to evaluate reasonable values for *reaugment-percentage*.

	0.0	1.0	3.0	5.0	10.0
0.0	—	$< 10^{-50}$	$< 10^{-50}$	$< 10^{-50}$	$< 10^{-50}$
1.0	$< 10^{-50}$	—	$< 10^{-50}$	$< 10^{-50}$	$< 10^{-50}$
3.0	$< 10^{-50}$	$< 10^{-50}$	—	0.38	0.12
5.0	$< 10^{-50}$	$< 10^{-50}$	0.38	—	0.038
10.0	$< 10^{-50}$	$< 10^{-50}$	0.12	0.038	—

Table A.5.: p-values for the evaluation of ω_{station}'s influence on the number of lines bending at stations

	0.0	1.0	5.0
0.0	—	$< 10^{-50}$	$< 10^{-50}$
1.0	$< 10^{-50}$	—	$< 10^{-50}$
5.0	$< 10^{-50}$	$< 10^{-50}$	—

Table A.6.: p-values for the evaluation of ω_{line}'s influence on the number of lines bending on edges

	0	1	3	5	10
0	—	0.23	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$
1	0.23	—	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$
3	$< 10^{-5}$	$< 10^{-5}$	—	0.48	$< 10^{-5}$
5	$< 10^{-5}$	$< 10^{-5}$	0.48	—	$< 10^{-4}$
10	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$	—

Table A.7.: p-values for the evaluation of ω_{station}'s influence on the height of the drawing when doing 5000 annealing steps

	0.0	1.0	5.0	6.0	7.0	8.0	10.0
0.0	—	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$
1.0	$< 10^{-100}$	—	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$
5.0	$< 10^{-100}$	$< 10^{-100}$	—	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$
6.0	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	—	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$
7.0	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	—	0.093	0.014
8.0	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	0.093	—	0.4
10.0	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	$< 10^{-100}$	0.014	0.4	—

Table A.8.: p-values for the evaluation of good values for ω_{station} at $\omega_{\text{line}} = 5.0$

	250	500	750	1000	1500	2000	3000	5000
250	—	0.25	0.15	0.42	0.23	0.25	0.22	0.37
500	0.25	—	0.72	1	0.82	0.88	0.79	0.64
750	0.15	0.72	—	0.65	0.72	0.63	0.78	0.3
1000	0.42	1	0.65	—	0.84	0.86	0.76	0.53
1500	0.23	0.82	0.72	0.84	—	0.7	0.97	0.42
2000	0.25	0.88	0.63	0.86	0.7	—	0.5	0.49
3000	0.22	0.79	0.78	0.76	0.97	0.5	—	0.74
5000	0.37	0.64	0.3	0.53	0.42	0.49	0.74	—

Table A.9.: p-values for the evaluation of good values for ω_{dist} at $\omega_{\text{height}} = 2000.0$ if the goal is to minimize neighboring stations' distances' standard deviation. Note that no Dunn-Bonferroni correction has been applied to this data. Doing so would mean that for statistic significance at 95% confidence, any value would have to be less than $\frac{0.05}{32} \approx 0.0016$. This requirement is not met by any of the above values.

	250	500	750	1000	1500	2000	3000	5000
250	—	0.085	0.005	0.025	0.0016	2.1e-05	0.01	0.0029
500	0.085	—	0.28	0.76	0.052	0.026	0.28	0.14
750	0.005	0.28	—	0.46	0.67	0.096	0.72	0.63
1000	0.025	0.76	0.46	—	0.29	0.037	0.4	0.21
1500	0.0016	0.052	0.67	0.29	—	0.49	0.45	0.9
2000	2.1e-05	0.026	0.096	0.037	0.49	—	0.32	0.39
3000	0.01	0.28	0.72	0.4	0.45	0.32	—	0.64
5000	0.0029	0.14	0.63	0.21	0.9	0.39	0.64	—

Table A.10.: p-values for the evaluation of good values for ω_{dist} at $\omega_{\text{height}} = 2000.0$ if the goal is to minimize the drawings height. Note that no Dunn-Bonferroni correction has been applied to this data. Doing so would mean that for statistic significance at 95% confidence, any value would have to be less than $\frac{0.05}{32} \approx 0.0016$. This requirement is only met by the pairs $(250, 1500)$ and $(250, 2000)$.

	0.01	0.02	0.03	0.04	0.06	0.10	0.20	0.30
0.01	—	0.13	0.085	0.36	0.08	0.6	0.95	0.0061
0.02	0.13	—	0.92	0.6	0.46	0.043	0.3	$< 10^{-3}$
0.03	0.085	0.92	—	0.58	0.74	0.056	0.092	$< 10^{-3}$
0.04	0.36	0.6	0.58	—	0.61	0.26	0.52	$< 10^{-3}$
0.06	0.08	0.46	0.74	0.61	—	0.022	0.067	$< 10^{-3}$
0.10	0.6	0.043	0.056	0.26	0.022	—	0.8	0.077
0.20	0.95	0.3	0.092	0.52	0.067	0.8	—	0.021
0.30	0.0061	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	0.077	0.021	—

Table A.11.: p-values for the evaluation of good values for *reaugment-percentage* if the goal is minimizing the drawings height. Note that no Dunn-Bonferroni correction has been applied to this data.

	10	100	1000	5000	10000	20000
10	—	$< 10^{-5}$	$< 10^{-5}$	0.41	0.66	0.78
100	$< 10^{-5}$	—	0.38	0.0014	0.0049	0.00024
1000	$< 10^{-5}$	0.38	—	$< 10^{-5}$	0.00028	$< 10^{-5}$
5000	0.41	0.0014	$< 10^{-5}$	—	0.65	0.18
10000	0.66	0.0049	0.00028	0.65	—	0.85
20000	0.78	0.00024	$< 10^{-5}$	0.18	0.85	—

Table A.12.: p-values for the evaluation of the time per annealing iteration for different values of *#anneal*

.