Julius-Maximilians-Universität Würzburg

Lehrstuhl für Informatik I

Masterarbeit

im Fach Informatik

Realtime Linear Cartograms using Least-Squares Optimisation

Autor:
Dieter Lutz

Betreuer:
Thomas van Djik
Prof. Dr. Alexander Wolff

September 5, 2014

Sommersemester 2014

Zusammenfassung

Wir stellen einen effizienten Algorithmus für das Zeichnen von räumlich-informativen, linearen Kartogrammen vor: Es wird ein geometrischer Graph transformiert, so dass gegebene Kantenlängen realisiert werden, während die Ausrichtung der Kanten so wenig wie möglich verändert wird. Anwendungen sind beispielsweise das Zeichnen von Reisezeit-Karten und U-Bahn-Plänen. Unser Algorithmus basiert auf einer linearen Optimierung mit Hilfe der Methode der kleinsten Quadrate und arbeitet mit weichen Nebenbedingungen für die realisierten Kantenlängen und Ausrichtungen; wir erreichen einen schnellen Algorithmus, indem wir diese Nebenbedingungen linearisieren. Diese Linearisierung kann auch gegebene Kantenausrichtungen heuristisch approximieren; wir benutzen dies auf geometrischen Graphen um die Winkelauflösung zu verbessern und Oktilinearität zu approximieren. Wir demonstrieren, wie diese Techniken zur Erstellung von U-Bahn-Plänen benutzt werden können. Schließlich zeigen wir experimentell die Echtzeit-Tauglichkeit des Algorithmus und seine Fähigkeit gute Realisierungen für praxisnahe Instanzen zu finden.

Abstract

We introduce an efficient algorithm for drawing spatially-informative linear cartograms: transforming a geometric graph such that given edge lengths are realised, while distorting edge directions as little as possible. This has applications in drawing travel-time maps and metro maps, for example. Our algorithm is based on linear least-squares optimisation and works by having soft constraints on the realised edge lengths and edge directions; we obtain a fast algorithm by linearising these constraints. This linearisation can also be used as a heuristic to approximate given edge directions; we utilized this on geometric graphs to improve angular resolution and approximate octilinearity. We demonstrate how these techniques can be applied to drawing metro maps. Lastly, we show experimentally that the algorithm has realtime performance and that it is able to find good realisations of practical instances.

Contents

Chapter 1

Introduction

A linear cartogram (or: edge-value cartogram) is a drawing of a network such that every edge is drawn with a prescribed length. In general this is not possible: consider for example a triangle with two short edges and one very long edge. Therefore, compromise between the edges may be required.

One application of linear cartograms is the so-called travel-time map [7], in which a geographic network is deformed so that Euclidean distance can be used to read off (approximate) travel time. This can be used as a visual aid for planning public transport journeys or as visualisation of traffic jams. Being able to compute such maps in realtime enables personalised maps and realtime traffic information. With runtimes in the order of tens of milliseconds and low memory usage due to utilising sparse matrices our algorithm is ideally suited for interactive mobile applications. Another application of linear cartograms is in the construction of metro maps, where it is often desirable to draw the stops with uniform spacing.

When drawing a travel-time map, we should not care solely about the lengths of edges: in many applications, the original map must be somewhat recognisable in order for the cartogram to be useful. This is true even for metro maps: though they are heavily stylised, with an emphasis on the topological rather than geographic aspects of the network, they too should ideally not contradict the reader's mental map. To facilitate this, we say a *spatially-informative* linear cartogram should, as much as possible, not change the direction of edges. Our algorithm is able to enhance the readability of metro maps by improving angular resolution and realising approximate octilinearity; still under the premise of changing edge directions as little as possible.

We use least-squares optimisation to construct our drawings. This well-known mathematical framework can be interpreted as attempting to satisfy a contradictory set of *soft* constraints as well as possible. We briefly review it in Chap. 2.

Our contribution

Firstly, we give an algorithm for spatially-informative linear cartograms (Chap. 3). It is based on soft constraints for edge length and edge direction. These constraints are non-linear, so we develop a linear approximation. Instead of asking the optimisation to leave the edge directions unchanged, we can also ask it to realise given edge directions (Chap. 4). We use this as a heuristic for improving angular resolution and approximating octilinearity in graph drawings. Linear cartograms and the realisation of given edge directions can be used independently or together; both use least-squares optimisation and both work well for the drawing of metro maps. We demonstrate this in our case study (Chap. 5). Lastly, we show experimentally that our algorithm is efficient and effective in computing linear cartograms as well as realising given edge directions (Chap. 6). On real data, it has runtimes in the order of tens of milliseconds.

In the literature on linear cartograms, the issue of edge intersections has been largely ignored. Being based on least-squares optimisation, our algorithm can ensure that the a crossing-free input drawing remains crossing-free after transformation by using the *event constraints* from [22]. We extend this approach for metro map drawing and propose two methods that prevent branches of metro maps from getting placed in close proximity.

Chapter 2

Related work

The concept of a travel-time map dates back to Clark [7], who draws a single-source time cartogram by starting at a source vertex and building the network outward in a breadth-first-search manner. An algorithm for travel-time maps of a single source that also deforms the underlying map along with the network was given by Bies and van Kreveld [1]. Buchin et al. [5] propose another approach to maintain the underlying map by keeping the vertex positions fixed and replacing straight lines between vertices with sinusoid curves of desired length.

The abstract problem of drawing a set of points such that a given set of point pairs has prescribed distance is known as *graph realisation.* Saxe [20] and Cabello et al.[6] have shown that this problem is $\mathcal{NP}$-hard in many settings and variants. Still, the problem is well studied, for example in work on sensor network localisation [9, 10]. Specifically in the context of graph drawing, there is a relation to multidimensional scaling, as noted for example by Gansner et al. [11]. In fact, multidimensional scaling has been applied to time-distance maps by Kaiser et al. [13]. Such approaches are computationally somewhat expensive: they typically require an iterative solver. Inoue and Shimizu [21] use least-squares optimisation with linearised constraints depending on edge directions for computing linear cartograms, similar to us. Their linearisation, however, is less flexible and they also require an iterative solver.

Of particular interest to us are the focus-and-context metro maps of Wang and Chi [23], based on non-linear least-squares optimisation. They use a course-to-fine approach, whereby they first find a solution to a much simplified version of the network and then later reintroduce the omitted vertices. This helps with convergence and runtime. In contrast, our algorithm is easily fast enough to be run on an entire network at once, and since our model is linear we always find the unique optimal solution (to our linear approximation). The runtime of our algorithm cannot be directly compared to that of Wang and Chi, however, because their algorithm attempts some things that are outside the scope of this thesis; for example, they label the stations on the focused path. For general metro maps Nöllenburg and Wolff [19] propose an algorithm based on mixed-integer programming; their problem design also includes the labeling of stations. Since mixed-integer programming is computationally expensive, they use a network of reduced size and then reintroduce omitted vertices, similarly to Wang and Chi.

In this thesis we consider a drawing to be just the positions of the vertices. Böttger et al. give an algorithm to morph additional information along with the network [3].

Least-squares optimisation

Here we briefly review the basic framework of least-squares optimisation. Least-squares optimisation is a standard approach to find approximate solutions to overdetermined systems of equations. A system of equations is considered overdetermined if there are more equations than variables. Given a set of variables and a overdetermined system of equality constraints, least-squares optimisation asks for values of the variables such that the constraints are satisfied "as well as possible," in the following sense. Given the variables, the *discrepancy* of a constraint is the difference between its left-hand and right-hand side. Least-squares optimisation asks for a solution such that the sum of squares of discrepancies is minimised, hence the name. Since any constraint is allowed to have some discrepancy, such constraints are called *soft.* [1] It is possible to assign different weights to each of the constraints by multiplying every equality constraint on both sides with the square root of their prescribed weight.

If all constraints are linear, we can represent the system of constraints using a matrix equation $Ax = b$, where x is the vector of variables, A the coefficient matrix and b the vector of the constraints' right-hand sides. Then the least-squares solution x minimises the norm of $Ax - b$. It is well known that this minimum is achieved by solving $A^TAx = A^Tb$. If the constraints are (over)determined, this has a unique solution that can be readily found using standard linear-algebra software. This is particularly efficient if A and A^TA are sparse [15].

If the system of constraints is not linear, a common approach is to use a sequence of linear approximations in a hill-climbing-like approach. Such algorithms are computationally more expensive, since they have to potentially solve many instances of linear least squares. They can also have problems with convergence and can be sensitive to the required *initial* linear approximation used [2], potentially finding a bad local optimum even if the algorithm converges.

[1]It is possible to have hard equality constraints by substituting variables, but we do not use this.

Chapter 3

Linear cartograms

In this section we model the problem of drawing a linear cartogram in the framework of least-squares optimisation. In particular, we model it using a linear model. This is in contrast to for example Wang and Chi [23] who have a similar approach, but use a non-linear model and therefore require an iterative solver.

First we define the objective function that we would actually like to optimise. Unfortunately, this objective function is non-linear, so we develop a linear approximation. In particular, we don't use the common approach of iterated linear approximations using Jacobian matrices; instead, we are more careful about what linear approximation we use.

Preliminaries

By convention, we use uppercase characters for constants and lowercase characters for variables. Let $G = (V, E)$ be a connected graph, possibly representing a road network or a set of metro lines. We are furthermore given the original position $P_v = (X_v, Y_v)$ of every vertex $v \in V$ as input: this is the ground truth in relation to which we want to be spatially informative. For geographic networks these would be real-world positions. We also get a desired length L_e for every edge $e \in E$.

As variables of our optimisation problem, we take the coordinates of the vertices: x_v and y_v for every $v \in V$. For notational convenience, let $p_v = (x_v, y_v)$. Then ideally we want a drawing of G that realises the requested lengths exactly and leaves the edge directions unchanged: a spatially-informative linear cartogram. In a perfect drawing, we have for every edge $e = \{u, v\}$ that

$$p_u + D_e \; = \; p_v, \qquad \text{where } D_e = L_e \cdot \frac{P_v - P_u}{\|P_v - P_u\|} \; . \tag{3.1}$$

Note that this equation is linear in the variables, since D_e is a constant.

These constraints do not fully determine the coordinates of the vertices: the entire drawing can be translated without affecting the constraints. Therefore we arbitrarily fix the position of one of the vertices. When displaying the map, the size of the computed drawing follows from the requested lengths given in the input; if the drawing needs to fit a certain extent and only the relative length of the edges is important, a uniform scale factor can be applied as postprocessing.

3.1 Non-linear objective

The stated objective of a spatially-informative linear cartogram is to realise certain edge lengths and additionally care about the direction of edges. Our optimisation objective is therefore to find a drawing with small error in length and direction. Consider soft constraints that set $\|p_v - p_u\| = L_{\{u,v\}}$ and the direction of $(p_v - p_u)$ equal to the direction of $(P_v - P_u)$ for every edge $\{u, v\} \in E$. We call these the length constraints and the direction constraints respectively. Note that they are not linear in our variables.

For every edge $e = \{u, v\}$, let Δr_e be the discrepancy of its length constraint, that is, $\Delta r_e = \|p_v - p_u\| - L_e$. Similarly, let $\Delta\alpha_e$ be the discrepancy of e's direction constraint. We weight the constraints by factors depending on the requested length L_e. Call these weights $W_\mathrm{r}(L_e)$ and $W_\alpha(L_e)$ respectively. Then a least-squares solution to this set of constraints minimises

$$\sum_{e \in E} f_\mathrm{polar}(\Delta\alpha_e, \Delta r_e, L_e), \qquad \text{where} \tag{3.2}$$

$$f_\mathrm{polar}(\alpha, r, L) = W_\alpha(L) \cdot \alpha^2 + W_\mathrm{r}(L) \cdot r^2 \ . \tag{3.3}$$

Our choice of weights is led by the "visual impact" of errors: two errors with the same value according to f_polar, should have approximately the same visual impact on the network. This, of course, is always a subjective matter. However, it is agreeable that the same *absolute* length error has more impact on smaller edges. Similarly, we consider the same *relative* length error to have more impact on longer edges. We decide to compromise between the two and set

$$W_\mathrm{r}(L_e) = \frac{1}{L_e} \ . \tag{3.4}$$

Figure 3.1 shows our weight $W_\mathrm{r}(L_e)$ in contrast to absolute and relative weighting.

This leaves the choice of weight $W_\alpha(L_e)$ for the direction discrepancy. Here we decide that a direction error of $\pi/4$ radians is equally as bad as being too long or too short by 50%. Setting this equality and using $W_\mathrm{r}(L_e) = \frac{1}{L_e}$, we get

$$W_\alpha(L_e) \cdot \left(\frac{\pi}{4}\right)^2 = \frac{1}{L_e} \cdot \left(\frac{L_e}{2}\right)^2 \quad \Longrightarrow \quad W_\alpha(L_e) = \frac{4L_e}{\pi^2} \ . \tag{3.5}$$

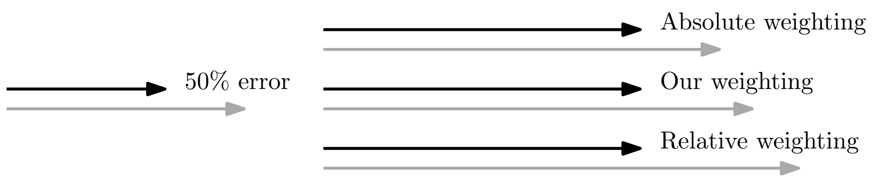


Figure 3.1: Our weight of length errors in contrast to absolute and relative weighting: The black edges show the prescribed length and the grey edges the actual length. The edges on the left side serve as reference and the edges on the right side show equally bad errors according to the respective weighting.

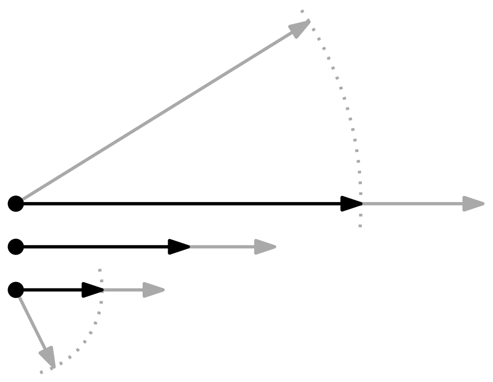


Figure 3.2: Weighting of length error to direction error: each of the grey edges is considered equally bad according to the black edges as prescribed lengths.

Note that the proposed length weight and the chosen equality also lead to lower weight on direction discrepancy on short edges. The effect of our weights is illustrated in Fig. 3.2.

The output of our algorithm should be independent from the scale of the input network; this makes it reasonable to change the scale of the output to fit a certain extent. Hence, we must confirm that the objective function f_{polar} is scale invariant with our weights W_α and W_{r}.

Theorem 1. *The objective function*

$$f_{polar}(\alpha, r, L) = W_\alpha(L) \cdot \alpha^2 + W_r(L) \cdot r^2$$

is scale invariant for weights $W_\alpha(L) = \frac{4L}{\pi^2}$ *and* $W_r(L) = \frac{1}{L}$.

Proof. Let $f_{\text{polar}}(\alpha, r, L)$ be the objective value of the original input network. Let $s \in \mathbb{R}^+$ be the constant scale factor between the original and a scaled input network. Notice that only r and L are affected by scaling; α stays the same. Hence, $f_{\text{polar}}(\alpha, s \cdot r, s \cdot L)$ is the objective value for the scaled input network. The following holds:

$$\begin{aligned} f_{\text{polar}}(\alpha, s\cdot r, s\cdot L) &= W_\alpha(s\cdot L)\cdot\alpha^2 + W_{\text{r}}(s\cdot L)\cdot(s\cdot r)^2 \\ &= \frac{4s\cdot L}{\pi^2}\cdot\alpha^2 + \frac{1}{s\cdot L}\cdot(s\cdot r)^2 \\ &= s\cdot\left(\frac{4L}{\pi^2}\cdot\alpha^2 + \frac{1}{L}\cdot r^2\right) \\ &= s\cdot f_{\text{polar}}(\alpha, r, L) \end{aligned}$$

Therefore the objective value just gets scaled by the constant factor s: both optimisations lead to the same result. In other words, the objective function $f_{\text{polar}}(\alpha, r, L)$ is scale invariant. □

3.2 Linear approximation

As noted, we cannot minimise (3.2) using linear least squares. Consider (3.1) for some specific edge. It actually represents two constraints: one for the x coordinate and one for y. We could instead express these constraints in some other basis, if the transformation is linear. Considering that we would like to penalise direction error and length error differently, we rotate the coordinate system such that the edge is aligned with one of the axes. Then there is a 'perpendicular' and a 'parallel' discrepancy $\delta_\perp$ and $\delta_{||}$, which for small values are similar to direction and length. These discrepancies can be calculated as follows for an edge $e = \{u, v\}$:

$$\delta_\perp = (p_v - p_u)\cdot\frac{(P_v - P_u)^\perp}{\|P_v - P_u\|} \quad \text{and} \quad \delta_{||} = (p_v - p_u)\cdot\frac{P_v - P_u}{\|P_v - P_u\|} - L_e$$

We set them to zero with soft constraints. Notice that the dot product is with a constant vector and therefore linear. See Fig. 3.3 for the geometry involved. Now that we have separate constraints for the perpendicular and parallel error, we can weight them differently; call these weights $W_\perp(L_e)$ and $W_{||}(L_e)$. Then the linear soft constraints will optimise the following, as a function of α, r and L (Compare (3.3)):

$$f_{\text{lin}}(\alpha, r, L) = W_\perp(L)\cdot(\sin(\alpha)(L+r))^2 + W_{||}(L)\cdot(\cos(\alpha)(L+r) - L)^2 \quad (3.6)$$

Note that matrix A that encodes these constraints is sparse, since every constraint involves only a constant number of variables. Experiments show that on practical instances A^TA is also sparse. Linear least-squares optimisation with such sparse matrices A and A^TA is particularly efficient [15].

Solving the optimisation with objective function (3.6) may in general result in a drawing with edge crossings. Edge crossings are undesirable, because they make drawings harder to read and also often contradict the reader's mental map. Therefore we always use the event constraints from [22] to prevent intersections, unless noted otherwise.

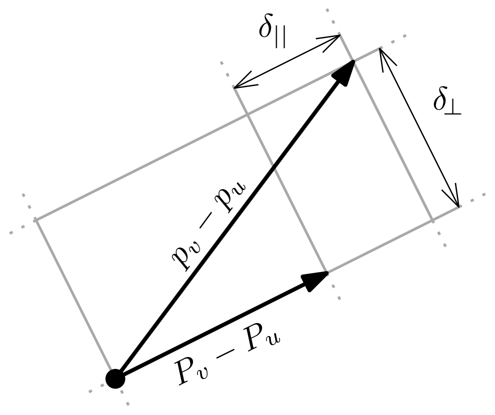


Figure 3.3: The geometry of parallel and perpendicular discrepancies.

3.3 Parameter values

This leaves the choice of $W_\perp$ and $W_{||}$. Consider the difference between our actual objective (f_{polar}) and the objective we can efficiently optimise (f_{lin}): Let $f_{\text{diff}} = f_{\text{polar}} - f_{\text{lin}}$. We want f_{diff} to be small for all reasonable values of α and r: then the objective that we actually optimise is close to the objective we *want* to optimise. We assume that all errors in a reasonable range are equally as likely; to that end, we pick $W_\perp$ and $W_{||}$ so as to minimise

$$\int_{-L/2}^{L/2} \int_{-\frac{\pi}{4}}^{\frac{\pi}{4}} f_{\text{diff}}(\alpha, r, L)^2 \, \mathrm{d}\alpha \, \mathrm{d}r \quad .$$

The bounds $\alpha \in [-\frac{\pi}{4}; \frac{\pi}{4}]$ and $r \in [-\frac{L}{2}; \frac{L}{2}]$ have been chosen because we rarely observe worse discrepancy, so we concentrate on f_{diff} for relevant parameter values. Solving this for the weights gives the following:

$$W_\perp(L_e) = \frac{0.413051}{L_e} \quad \text{and} \quad W_{||}(L_e) = \frac{1.0039}{L_e} \tag{3.7}$$

The calculation is standard, but rather unwieldy, so we present Mathematica code. Tedious manual calculation confirms these results.

First of all, we define the error functions f_{polar} (`fPolar`), f_{lin} (`fLin`) and their difference `fDiff`.

```
In: fPolar[α_, r_, L_] := Warc ∗ α^2 + Wrad ∗ r^2;
    fLin[α_, r_, L_] := Wperp∗(Sin[α](L+r))^2+Wpar∗(Cos[α](L+r)−L)^2;
    fDiff[α_, r_, L_] := fPolar[α, r, L] − fLin[α, r, L];
```

Then we integrate `fDiff` squared over $\alpha \in [-\frac{\pi}{4}; \frac{\pi}{4}]$ and $r \in [-\frac{L}{2}; \frac{L}{2}]$ respectively.

```
In: Integrate[fDiff[α, r, L]^2, {α, −Pi/4, Pi/4}, {r, −L/2, L/2}]
```

Next we minimise this integral over `Wperp` and `Wpar`. We specify appropriate bounds for the weights and L and state that they must be real numbers. Applying `Simplify` and `N` convinces Mathematica to display a numeric result.

In:
```
Simplify[N[Simplify[
    Minimize[{%, Element[Wperp, Reals], Wperp>=0,
                 Element[Wpar, Reals], Wpar>=0,
                 Element[L, Reals], L>0,
                 Element[Warc, Reals], Warc>=0,
                 Element[Wrad, Reals], Wrad>=0},
              {Wperp, Wpar}]
]]]
```

This leads to approximately the following values. Notice the dependence on `L`; this is a constant for our optimisation problem.

Out:

$\texttt{Wperp} \rightarrow 0.115959\,\texttt{Wrad} + \frac{0.733046}{L^2}\,\texttt{Warc}$

$\texttt{Wpar} \rightarrow 0.694059\,\texttt{Wrad} + \frac{0.764513}{L^2}\,\texttt{Warc}$

Following (3.4) and (3.5), we set $\texttt{Wrad} = \frac{1}{L}$ and $\texttt{Warc} = \frac{4L}{\pi^2}$. This gives the following values for $W_\perp$ and $W_{||}$:

In:
```
Wperp = 0.115959 * 1/L + 0.733046/(L^2) * 4L/(Pi^2)

Wpar = 0.694059 * 1/L + 0.764513/(L^2) * 4L/(Pi^2)
```

Out:

$\frac{0.413051}{L}$

$\frac{1.0039}{L}$

Then we are done: these are the values in (3.7).

Notice that our proof of Theorem 1 works similarly for the objective function $f_{\text{lin}}(\alpha, r, L)$ and weights $W_\perp(L) = \frac{0.413051}{L}$ and $W_{||}(L) = \frac{1.0039}{L}$.

Corollary 1. *The objective function*

$$f_{lin}(\alpha, r, L) = W_\perp(L) \cdot (\sin(\alpha)(L + r))^2 + W_{||}(L) \cdot (\cos(\alpha)(L + r) - L)^2$$

is scale invariant for weights $W_\perp(L) = \frac{0.413051}{L}$ *and* $W_{||}(L) = \frac{1.0039}{L}$.

3.4 Comparison to one Jacobian step

We briefly introduce the textbook way of doing least-squares optimisation with a non-linear system of constraints. This uses a sequence of linear approximations with Jacobian matrices in a hill-climbing-like approach.

Each iteration of the hill-climbing process takes some approximate solution for the prescribed constraints. It performs a least-squares optimisation on changes that improve the given solution; the improved solution is then used for the next iteration until a local optimum for this procedure is reached.

The optimisation linearises the length and direction constraints from Sect. 3.1. The constraints consist of functions f_{len} and f_{dir} calculating the length or direction of an edge on the left-hand side and the expected result on the right-hand side. Let A be the Jacobian matrix of these functions at the values of the current solution. Call b the vector of differences between the desired lengths and directions of edges and their values in the current solution. Then $Ax = b$ encode the desired linearised constraints for the optimisation.

Let $p_c = (x_c, y_c)$ and $p_d = (x_d, y_d)$ be the coordinates of two vertexes c and d. We used the following length and direction functions for an edge $\{c, d\}$:

$$f_{\text{len}}(x_c, y_c, x_d, y_d) = \|p_d - p_c\| \quad \text{and}$$

$$f_{\text{dir}}(x_c, y_c, x_d, y_d) = \text{atan2}(y_d - y_c, x_d - x_c) \quad \text{where}$$

$$\text{atan2}(y, x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if } y \geq 0, x < 0 \\ \arctan(\frac{y}{x}) - \pi & \text{if } y < 0, x < 0 \\ \frac{\pi}{2} & \text{if } y > 0, x = 0 \\ -\frac{\pi}{2} & \text{if } y < 0, x = 0 \\ \text{undefined} & \text{if } y = 0, x = 0 \end{cases}$$

Let $P_c = (X_c, Y_c)$ and $P_d = (X_c, Y_c)$ be the coordinates of c and d in the current solution. Then we get the following Jacobian matrix A at P_c and P_d:

$$A = \begin{pmatrix} \frac{X_c - X_d}{\|P_d - P_c\|} & \frac{Y_c - Y_d}{\|P_d - P_c\|} & \frac{X_d - X_c}{\|P_d - P_c\|} & \frac{Y_d - Y_c}{\|P_d - P_c\|} \\ \frac{Y_d - Y_c}{\|P_d - P_c\|^2} & \frac{X_c - X_d}{\|P_d - P_c\|^2} & \frac{Y_c - Y_d}{\|P_d - P_c\|^2} & \frac{X_d - X_c}{\|P_d - P_c\|^2} \end{pmatrix}$$

The proposed method of this thesis only solves one instance of linear least squares, so we only do the first Jacobian step to get a comparable approach.

We still need an initial approximate solution for this first step. In default of a better alternative, it is reasonable to take the original position $P_v = (X_v, Y_v)$ of each vertex $v \in V$ as the initial solution.

Chapter 4

Realisation of given edge directions

Here we present a relatively straightforward modification to the above algorithm; we present it separately because it can be applied independently, outside of linear-cartogram applications. The idea is to realise given edge directions instead of the original ones. Two use cases of this are improving angular resolution and approximating octilinearity of the network. The combination of these two things with linear cartograms give the most pleasing results when drawing metro maps (see Chap. 5).

Note in (3.1) that the definition of D_e uses the normalised direction $P_v - P_u$. We can simply use a different direction for each edge and do the optimisation for those directions instead.

4.1 Angular resolution

One application of realising given edge direction is the improvement of angular resolution. This increases the readability of the drawing. A possible drawback on metro maps, for example, is higher Euclidean distance between unconnected stations. This may lead to the impression that station are quite far away, even if the distance between them is easily walkable.

Similar to Brandes et al. [4], we pick for every vertex a set of edge directions that has perfect angular resolution and is most similar to the original directions: we set them to have least-squares discrepancy compared to the directions in the input drawing. Then we ask our optimisation algorithm to realise these directions in our straight-line drawing; in contrast, Brandes et al. use these balanced directions as tangents for cubic splines. Note that these constraints do not force a path of degree-2 vertices to be realised as a straight line, but it does smooths the path. Figure 4.1 shows an example of improving angular resolution on a public transport system.

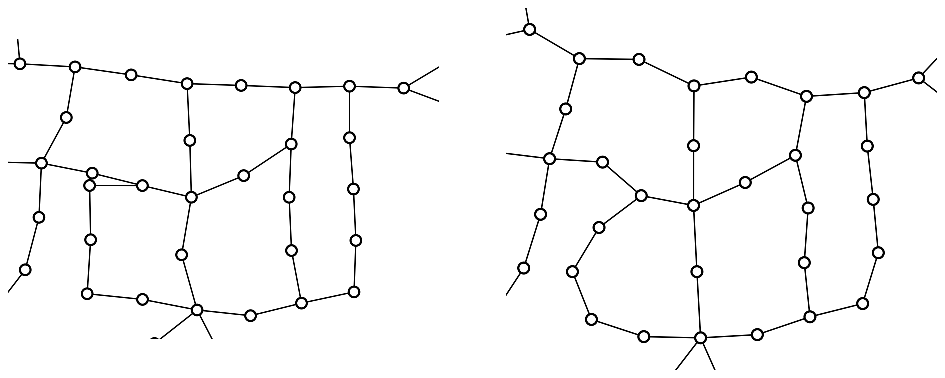

(a) Without angle resolution (b) With angle resolution

Figure 4.1: Crop of a public transport network (city centre of the Karlsruhe public transport system). Notice that the angular resolution has improved and that this benefits readability.

4.2 Octilinearity constraints

Octilinearity is a typical trait for metro map drawings. In an octilinear drawing every vertex has 8 available directions called *ports* in 45 degree margins. By assigning each edge to a port and asking our algorithm to realise these ports we can approximate octilinearity in our drawings. We call the respective constraints *octilinearity constraints.*

Similar to angular resolution, we pick for every vertex an assignment of edge directions to distinct ports that is most similar to the original directions in a least-squares sense. Then we ask our optimisation algorithm to realise these ports. The assignment of edges to distinct ports is a discrete least-squares problem; it is solved as preprocessing for our main least-squares optimisation.

Figure 4.2 shows the minimum-cost flow network leads to solution of this problem. Every segment in the network has capacity 1. The segment from an edge e to a port p has the squared discrepancy between p and the original direction as cost $c(e, p)$. The capacities ensure that every edge gets assigned to exactly one port; the set of assignments has minimum cost in a least-squares sense, since we use a minimum-cost flow network. An optimal solution with integer capacities for this network can efficiently obtained using standard techniques [8].

Note that this minimization also assures that the embedding of the vertexes is preserved.

Theorem 2. *Every assignment of edges to distinct ports for a vertex that minimizes the discrepancy between the assigned ports and the original edge directions in a least-squares sense, preserves it's embedding.*

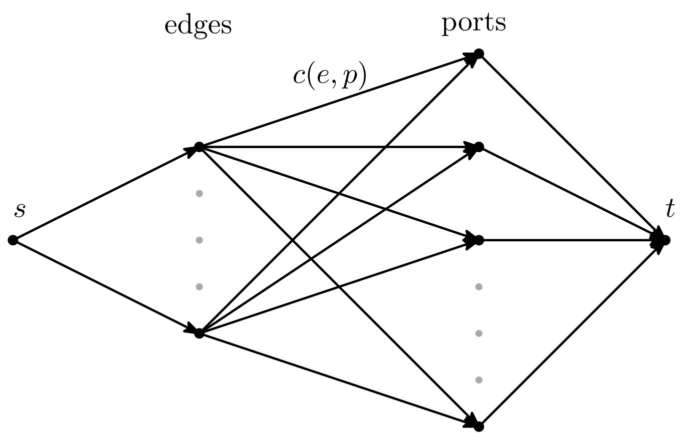


Figure 4.2: Minimum-cost flow network for octilinear edge directions for one vertex. Every segment has capacity 1. The segment from an edge e to a port p has the squared discrepancy between p and the original direction as cost $c(e,p)$.

Proof. We do a proof by contradiction. Assume that there is a minimal solution which does not preserve the embedding. We say an edge moves clockwise if the clockwise angle between it's original direction and assigned port is smaller than π radians, otherwise it moves counter-clockwise. Further we say an edge a overtakes an edge b if

- they move in the same direction and the sector induced by a's movement completely covers the sector of b
- they move in different directions and their sectors overlap at any point. Let a be the edge that moves counter-clockwise in this case.

Figure 4.3 illustrates overtaking edges. Since our minimal solution does not preserve the embedding, there must be such edges a overtaking b.

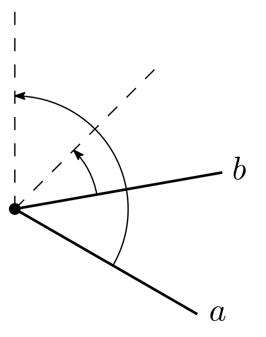
Let d_a be the original direction of edge a and d_b the original direction of edge b. Furthermore be p_a and p_b the directions of the assigned ports of these edges. We define $\angle(d, d')$ as the absolute discrepancy of two directions d and d'. Note that $\angle(d, d') = \angle(d', d)$ holds. We assume without loss of generality that a moves counter-clockwise.

Case 1. Edges a and b move in the same direction. We want to show that switching the assigned ports of a and b improves the solution. The squares discrepancies of a and b amount to

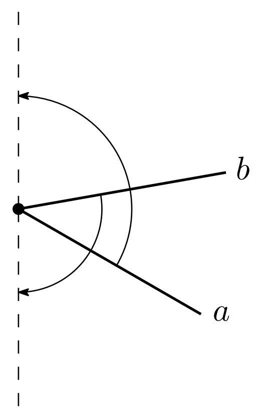
$$D_{\text{norm}} = \angle(d_a, p_a)^2 + \angle(d_b, p_b)^2 \ . \tag{4.1}$$

If we swap their ports, we get

$$D_{\text{swap}} = \angle(d_a, p_b)^2 + \angle(d_b, p_a)^2 \ . \tag{4.2}$$



(a) Edges move in the same direction.



(b) Edges move in different directions.

Figure 4.3: Edge a overtakes edge b while moving the same and different directions.

Since the sector induced by a's movement completely cover the sector of b, it follows

$$\angle(d_a, p_a) + \angle(d_b, p_b) = \angle(d_a, p_b) + \angle(d_b, p_a) \tag{4.3}$$

as well as

$$\angle(d_a, p_a) > \angle(d_a, p_b) \geq \angle(d_b, p_a) > \angle(d_b, p_b) \ . \tag{4.4}$$

Equation (4.3) leads to

$$\begin{aligned}[\angle(d_a, p_a) + \angle(d_b, p_b)]^2 &= [\angle(d_a, p_b) + \angle(d_b, p_a)]^2 \\ D_{\text{norm}} + 2 \cdot \angle(d_a, p_a)\angle(d_b, p_b) &= D_{\text{swap}} + 2 \cdot \angle(d_a, p_b)\angle(d_b, p_a) \ .\end{aligned}$$

So to prove $D_{\text{norm}} > D_{\text{swap}}$, we can also show that

$$2 \cdot \angle(d_a, p_a)\angle(d_b, p_b) < 2 \cdot \angle(d_a, p_b)\angle(d_b, p_a)$$

$$\angle(d_a, p_a)\angle(d_b, p_b) < \angle(d_a, p_b)\angle(d_b, p_a) \ . \tag{4.5}$$

Rearranging (4.5) leads to

$$\begin{aligned}&0 < \angle(d_a, p_b)\angle(d_b, p_a) - \angle(d_a, p_a)\angle(d_b, p_b) \\ &0 <^{(4.3)} \angle(d_a, p_b)\angle(d_b, p_a) - [\angle(d_a, p_b) + \angle(d_b, p_a) - \angle(d_b, p_b)]\,\angle(d_b, p_b) \\ &0 < \angle(d_a, p_b)\angle(d_b, p_a) - \angle(d_a, p_b)\angle(d_b, p_b) - \angle(d_b, p_a)\angle(d_b, p_b) + \angle(d_b, p_b)^2 \\ &0 < [\angle(d_b, p_a) - \angle(d_b, p_b)] \cdot [\angle(d_a, p_b) - \angle(d_b, p_b)] \ . \qquad (4.6)\end{aligned}$$

Inequality (4.6) holds, since both factors in square brackets are truly positive due to (4.4). So switching the ports of a and b does indeed improve the solution. This contradicts the solution's optimality.

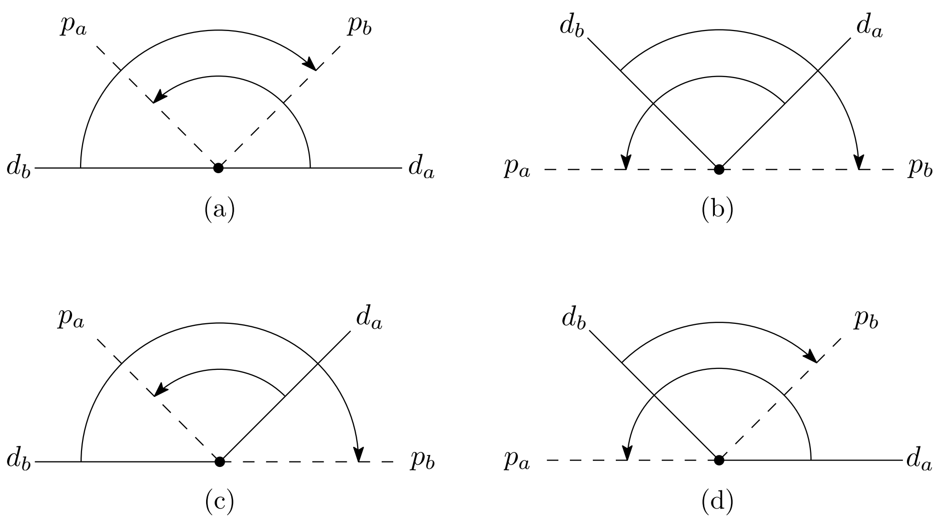


Figure 4.4: The four possible cases of overtakes if edges a and b move in different directions.

Case 2. Edges a and b move in different directions. We again want to show that swapping their ports improves the solution. The discrepancies D_{norm} and D_{swap} are the same as in (4.1) and (4.2). Figure 4.4 shows the four possible cases of overtakes. The following inequalities hold for Case (a):

$$\angle(d_a, p_a) > \angle(d_a, p_b)$$
$$\angle(d_b, p_b) > \angle(d_b, p_a)$$

For Case (c) the inequality

$$\angle(d_b, p_b) > \angle(d_a, p_b) + \angle(d_b, p_a)$$

holds. In both cases $D_{\text{norm}} > D_{\text{swap}}$ follows directly. Cases (b) and (d) work analogous to Cases (a) and (c) respectively. So switching the assigned ports also improves the solution in this case. This is in contradiction to the optimality of the solution.

It follows that every optimal solution must preserve the embedding. □

Note that the above proof works analogously if the ports are not octilinear but arbitrary; that is as long as we minimize the direction discrepancy in a least-square sense. Therefore the preservation of the embedding also applies to angular resolution.

Remember that we calculated a set of distinct octilinear directions for every vertex. We can now ask our optimization algorithm to realise these directions; this gives approximately octilinear drawings. Figure 4.5 shows that our algorithm leads to good approximations of octilinear drawings.

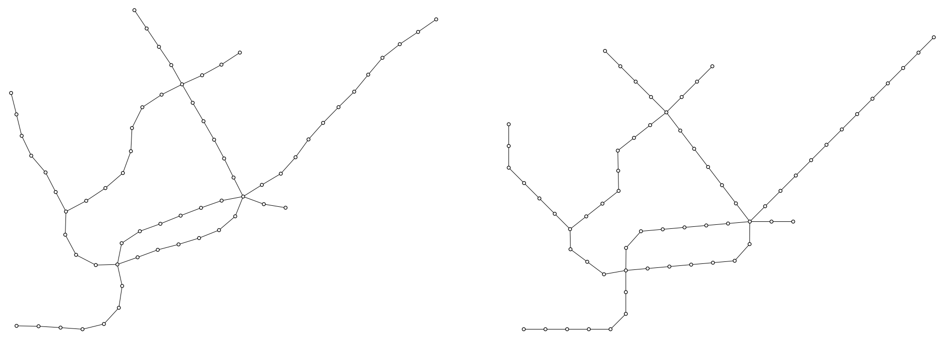

(a) Input drawing (b) Output drawing using octilinearity constraints

Figure 4.5: The input drawing for the calculation of the octilinear edge directions and the output drawing of the algorithm using octilinearity constraints with these directions.

The octilinear edge directions get calculated based on the input drawing in Fig. (a). The optimisation is then applied with these directions and produces the drawing in Fig. (b).

There are mainly two reasons we do not get fully-octilinear drawings: Firstly, the least-squares optimisation is a trade-off between length and direction errors. So often it will make small direction errors to satisfy length constraints better; especially since realising edge directions is not the main focus of our weights. Section 4.4 proposes a modification of our weights for improved realisation of edge directions. Secondly, two endpoints of an edge can be in disagreement about the port of the edge, since we optimise the directions separately for each vertex. This will most likely lead to a non-octilinear placement of the edge. This issue could be avoided by running the port optimisation on the whole graph; however, we are not sure if this problem is solvable in polynomial time.

Open problem. *Find a polynomial-time algorithm which assigns every edge of a graph a distinct octilinear port, while minimising the direction discrepancies in a least-squares sense; or, prove that the problem is $\mathcal{NP}$-hard.*

Additionally, the event constraints from [22] and the later introduced proximity constraints (Sect. 5.1) may interfere with the octilinear placement of edges.

4.3 Runtime improvement

We propose a simple tweak to our angular resolution and octilinearity constraints to improve the runtime our algorithm. Consider the calculation

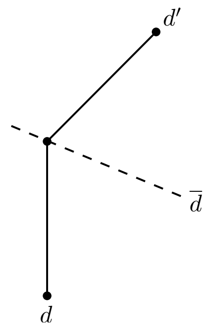


Figure 4.6: Mean direction $\overline{d}$ for an edge which vertexes prefer two different directions d and d'.

of edge directions for angular resolution as well as octilinearity constraints. Both work vertex-wise, calculating two, possible different, directions d and d' for every edge. Our first approach was to just use two sets of constraints for each edge; this doubles the number of constraints.

A better way to handle this problem is to combine those two directions into one, so we only get a single set of constraints per edge. The straightforward choice is the "mean" direction $\overline{d}$ of d and d'; where $\overline{d}$ is the direction which has (the shorter) equal discrepancy to both d and d' (Fig. 4.6). Not only does this speed up our algorithm, but we will also show that it results in an equivalent optimisation problem according to the non-linear error function f_{polar} (see (3.3)).

Theorem 3. *Optimising an edge e for two directions d and d' in a least-squares sense according to the non-linear error function f_{polar} is equivalent to optimising for the mean $\overline{d}$ of these directions.*

Proof. We calculate the length and direction errors for both optimisations and show their equivalence.

Length error. Let Δr_e be the discrepancy of the length constraint for edge e. Then we get the error term

$$E_{\text{len,two}} = 2 \cdot W_{\text{r}}(L_e)\, \Delta {r_e}^2$$

for the two directions d and d' and the error term

$$E_{\text{len,mean}} = W_{\text{r}}(L_e)\, \Delta {r_e}^2$$

for the mean direction $\overline{d}$. Note that both error terms are the same apart from a constant factor of 2 of $E_{\text{len,two}}$.

Direction error. Let $\Delta\alpha_e$ be the discrepancy of e's direction constraint for direction $\overline{d}$ and δ the absolute discrepancy between $\overline{d}$ and d. Note δ is also

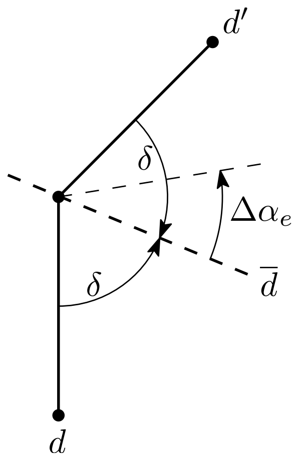


Figure 4.7: Geometry for calculating the direction error terms.

the absolute discrepancy between $\overline{d}$ and d'. Then the error term with two directions amounts to

$$\begin{aligned} E_{\text{dir,two}} &= W_\alpha(L_e)\,(\delta + \Delta\alpha_e)^2 + W_\alpha(L_e)\,(\delta - \Delta\alpha_e)^2 \\ &= 2 \cdot W_\alpha(L_e)\,\delta^2 + 2 \cdot W_\alpha(L_e)\,{\Delta\alpha_e}^2 \end{aligned}$$

and the error term for the optimisation on $\overline{d}$ is

$$E_{\text{dir,mean}} = W_\alpha(L_e)\,{\Delta\alpha_e}^2 \ .$$

Figure 4.7 shows the geometry involved in the calculation. Notice that the error term for two directions is independent of the sign of $\Delta\alpha_e$; it does not matter if the error is clockwise or counter-clockwise of $\overline{d}$. Furthermore the term $2 \cdot W_\alpha(L_e)\,\delta^2$ in $E_{\text{dir,two}}$ is constant and can be ignored for the optimisation. The remaining error terms again only differ by a constant factor of 2 on $E_{\text{dir,two}}$.

Since the error terms of both, the length and direction error, only differ by the same, constant factor on the side of the two directions, it follows that both optimisation problems are equivalent. □

It is to be noted that the Theorem 3 only holds if we exclusively use the proposed set of edge constraints. If we want to use different types of constraints, we would have to multiply them by factor 2 to get an equivalent optimisation problem in the two directions case. However, we only use the constraints with mean direction for every edge, so we do not have to worry about that.

4.4 Weight adjustments

Here we introduce adjustments for our weights to focus on realising edge directions. There are instances where we mostly care about realising prescribed directions and not so much about the length of edges; this is especially the case if we ask for an octilinear drawing. We can achieve this behaviour by changing the weights $W_{||}$ and $W_{\perp}$, so that they put more emphasis on the direction discrepancy when computing octilinear drawings. For the new weights we decide that a direction error of $\pi/30$ radians is already as bad as a length error of 50% (instead of $\pi/4$ radians earlier). This leads to the following weight $W_\alpha(L)$ for direction discrepancies:

$$W_\alpha(L) \cdot \left(\frac{\pi}{30}\right)^2 = \frac{1}{L} \cdot \left(\frac{L}{2}\right)^2 \qquad \Longrightarrow \qquad W_\alpha(L) = \frac{225 L_e}{\pi^2}$$

The weight $W_{\mathrm{r}}(L) = \frac{1}{L}$ stays the same.

We then calculate $W_{||}$ and $W_{\perp}$ analogous to the values in (3.7). Since the adjusted weights will put more emphasis on the direction discrepancy, we expect direction errors to be smaller than before. Therefore we also redo the integral with the bounds $\alpha \in [-\frac{\pi}{30}; \frac{\pi}{30}]$ and $r \in [-\frac{L}{2}; \frac{L}{2}]$. Next we minimize this integral over $W_{\perp}$ and $W_{||}$ for $W_\alpha(L) = \frac{225L}{\pi^2}$ and $W_{\mathrm{r}}(L) = \frac{1}{L}$. This gives us the following adjusted weights:

$$W_{\perp}(L) = \frac{15.2343}{L} \quad \text{and} \quad W_{||}(L) = \frac{1.13797}{L}$$

We scale these weights to make them behave similarly to the original weights. We decide that the weight for the length discrepancy should be the same for both sets of weights; this gives us a scale factor of 1.0039/1.13797. The square root of this factor is then applied to every set of edge constraints that use the adjusted weights.

Figure 4.8 compares the approximately octilinear drawing with original weights of Fig. 4.5(b) to the drawing calculated with the adjusted weights. We can see that our adjusted weights lead to better realisations of the octilinear directions; so that the drawing computed with adjusted weights almost reaches full octilinearity. Therefore we will use the adjusted weights for all future computations that include octilinearity constraints. Note that we can also get weights $W_{\perp}$ and $W_{||}$ for other compromises between direction and length discrepancy by analogous calculations.

Of course the improvement in direction realisation comes with the cost of higher length discrepancy, since we always have a trade-off between direction and length error. However, often the exact realisation of the edge length is not needed, but rather more precise edge directions; for example on octilinear drawings of metro maps.

The chosen ports for the edges can have crucial impact on the length realisation, especially when we use the adjusted weights. Figure 4.9 illustrates

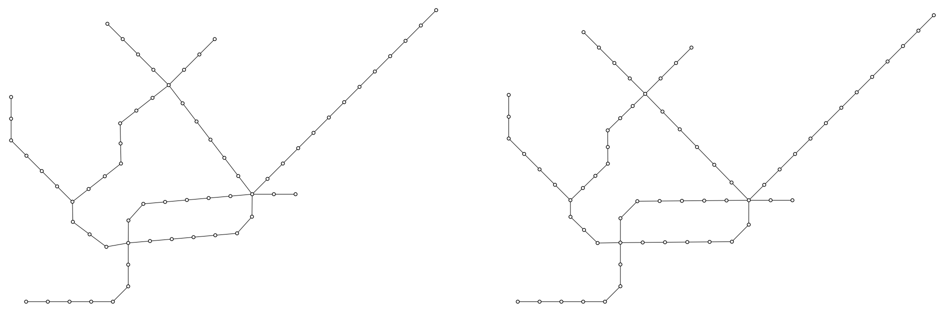
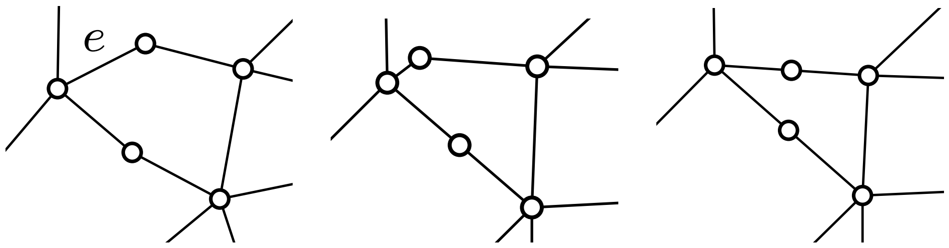


Figure 4.8: Comparison of drawings computed using octilinearity constraints with original and adjusted weights.



(a) Input network (b) Drawing using octilinearity constraints with original ports (c) Drawing using octilinearity constraints with modified ports

Figure 4.9: Input drawing for port calculation. Drawing using octilinearity constraints with these ports. Drawing using octilinearity constraints with eastern port for edge e.

this. The octilinear drawing (b) is computed by our algorithm with the network (a) as input; the algorithm assigns edge e to the north-eastern port. If we give the hint that the eastern port is better for e, we get drawing (c) with much better compromise on length discrepancy. This shows that the ports of minimal discrepancy to the original directions do not always lead to the best drawings. We leave the research for algorithms that resolve such situations to future work.

Chapter 5

Application: metro maps

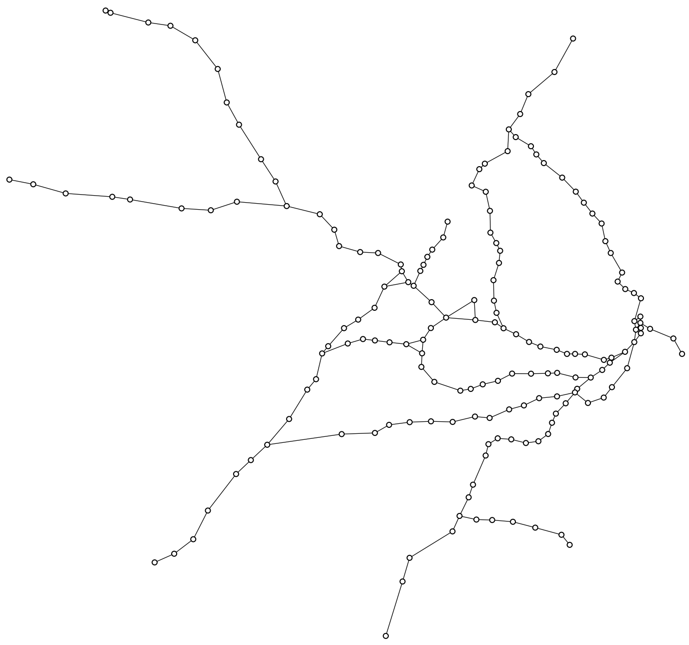
Now we apply our algorithm to compute metro map drawings. It is typically part of the design of a metro map that edges have approximately uniform length; this is, for example, *design rule (R5)* of Nöllenburg and Wolff [19]. Wang and Chi [23] use a small set of different edge lengths to create focus on a specific route through the network. Another approach is to use focusmap algorithms to scale part of the map (see for example Merrick and Gudmundsson [17]). This is used to enlarge the city centre and compress the suburbs, which is often necessary for a good metro map. A unit-length linear cartogram is a more direct way to achieve this: it automatically does what needs to be done (using more space where the network needs it) instead of using a roundabout step with a scale factor.

Now we combine both of our contributions: we draw a linear cartogram of the metro map with uniform edge length while using the angular-resolution improvement. Figure 5.1 shows the geographic input network and the computed metro map of Sydney as an example. The magnified circle in Fig. (b), where vertex v gets very close to edge e, shows an issue with our approach. This happens, because e and the edge of v do not intersect just yet, so the algorithm does not add a constraint for intersection prevention. To fix the issue we introduce a new set of constraints called *proximity constraints.*

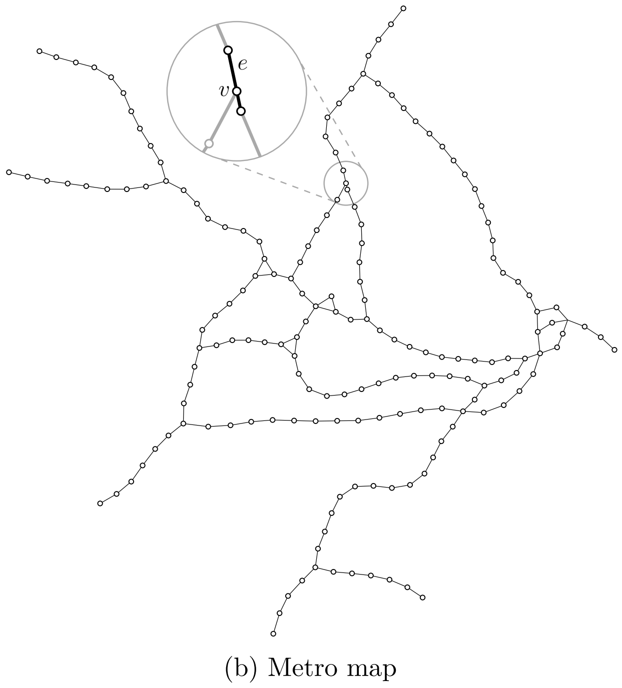
5.1 Proximity constraints

As seen in Fig. 5.1(b), one issue of linear cartograms with event constraints from [22] is that vertexes can get arbitrarily close to edges. For edges that are nearby in the network the angular resolution deals with this. The now introduced proximity constraints will do the same for vertexes and edges that are further apart in the network; they will also be used as a replacement for the event constraints.

We add a set of proximity constraints to the algorithm if a vertex v is in close proximity to an edge e. Similar to event constraints the optimisation



(a) Geographical input network



(b) Metro map

Figure 5.1: Geographical input network and metro map of Sydney computed with our algorithm using uniform edge length and angular resolution.

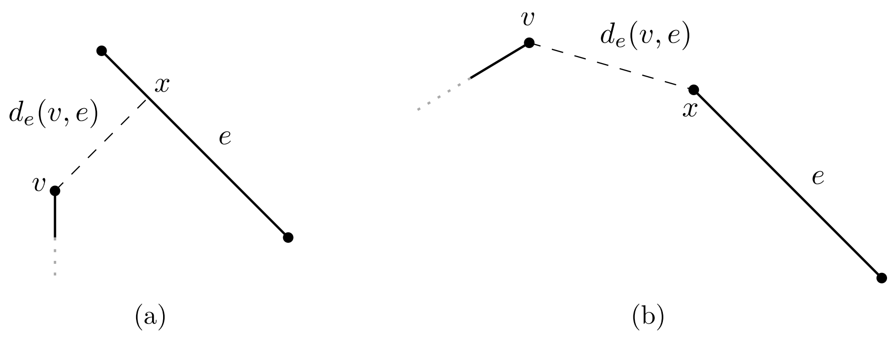


Figure 5.2: Euclidean distance $\text{dist}(v,e)$ between a point v and an edge e as dashed line.

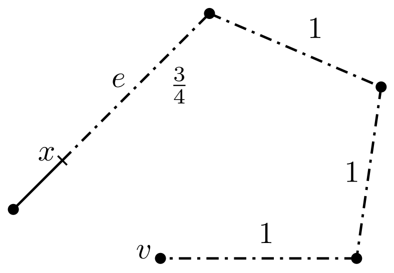


Figure 5.3: Network distance $\text{dist}_{\text{net}}(v,e)$ between a point v and an edge e as dash dotted line. The numbers indicate the contribution of the edges to the network distance.

will be rerun if proximity constraints were added, for up to 15 iterations. We consider v and e to be in close proximity if:

- they intersect
- their Euclidean distance $\text{dist}(v,e)$ is smaller than the uniform edge length and the following holds for $\text{dist}(v,e)$ and their network distance $\text{dist}_{\text{net}}(v,e)$:

$$\frac{\text{dist}(v,e)}{\text{dist}_{\text{net}}(v,e)} < 0.05 \tag{5.1}$$

The Euclidean distance $\text{dist}(v,e)$ is naturally the distance between the point v and closest point x of edge e. We define the network distance $\text{dist}_{\text{net}}(v,e)$ as the minimal number of hops from v to one of the endpoints of e plus the fraction of e from this endpoint to x. Figures 5.2 and 5.3 show examples for Euclidean and network distance.

The set of added proximity constraints demands that vertex v should stay on a parallel of edge e with distance of the uniform edge length. The chosen parallel is the one on the side of v (see Fig. 5.4).

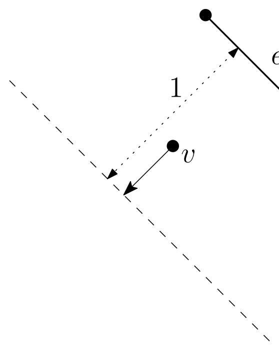


Figure 5.4: Parallel to edge e for the proximity constraint between point v and e.

The constraints are a variation of our edge constraints. We just take the parallel part of the constraints for an edge between v and x with uniform edge length. This is sufficient; v has to stay on the proposed parallel to e if we do not have a parallel error on that (imaginary) edge.

Figure 5.5 shows our metro map of Sydney when we add proximity constraints to the linear cartogram with uniform edge length and angle resolution. We can see that the proximity constraints did push v and e apart, just as anticipated.

5.2 All-pairs metro maps

We introduce another variation of our algorithm for metro map drawing. It keeps the uniform edge length for every edge in the network, but there are also edge constraints put in for every other pair of vertices in the network. We set the prescribed length of every edge induced by such a pair of vertices to the minimal number of hops between the vertices in the network; note that the number of hops between edges in the network is 1. This approach is known from general graph drawing [14]. Similarly, Inoue and Shimizu [21] use the travel times between all pairs of vertices of Japan's railway network to compute complete network linear cartograms. We call our approach *all-pairs metro maps.*

Our intent behind the additional edge constraints for every pair of vertices is to keep the branches of the network apart; they act as an alternative to angular resolution and proximity constraints. They should also help preserving the mental map of the network. Figure 5.6 shows the geographical input network and the all-pairs metro map of Karlsruhe computed by our algorithm. The encircled area highlights the main flaw of this approach: the realised length of network edges can be quite bad.

To handle this problem, we apply a weighting function to the edge constraints to put less weight on longer edges. Inoue and Shimizu take a

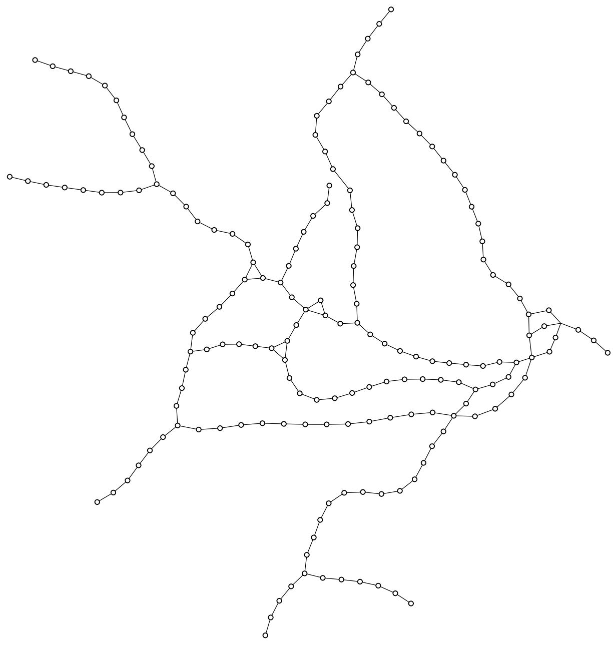


Figure 5.5: Metro map of Sydney computed as linear cartogram with uniform edge length, angular resolution and proximity constraints.

similar approach with their combined linear cartograms. They, however, use fixed weights, whereas we choose weights dependent on the prescribed length L_e for every edge e. We tested the following weighting functions:

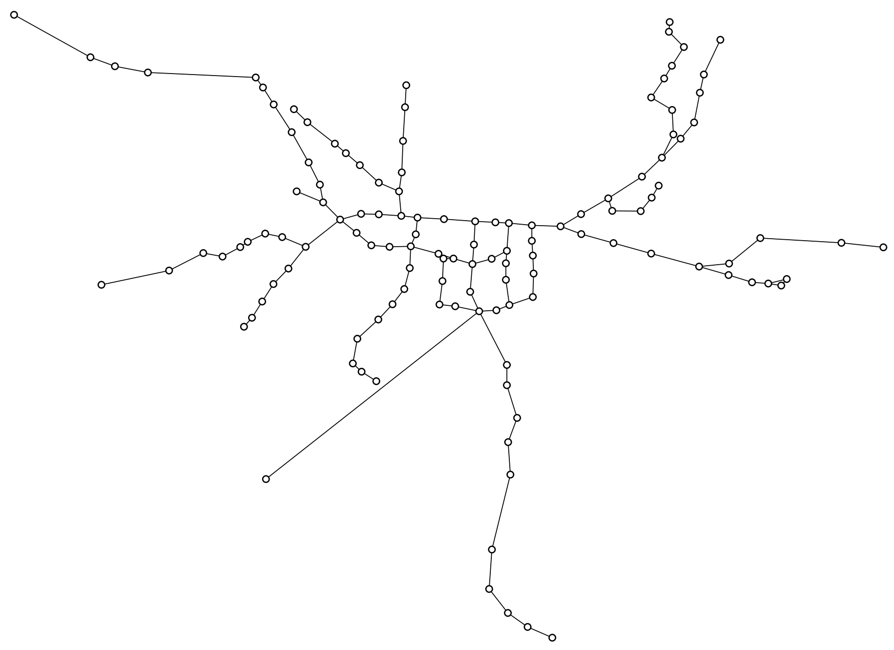
$$f_{\text{linear}}(L_e) = 1 - \frac{L_e - 1}{\max_e L_e}$$

$$f_{\text{reciprocal},x}(L_e) = (\frac{1}{L_e})^x \text{ with } x \in \mathbb{N}$$

$$f_{\text{delayedRec}}(L_e) = \begin{cases} 1 & \text{if } L_e \leq 3 \\ \frac{1}{L_e - 2} & \text{if } L_e > 3 \end{cases}$$

Only $f_{\text{reciprocal},x}$ for $x \geq 2$ led to good length realisations for network edges. In our opinion, the best metro maps of this approach are achieved for $x = 3$.

Figure 5.7(a) shows the all-pairs metro map of Karlsruhe computed with $f_{\text{reciprocal},3}$ as weighting function. This metro map has approximately the quality of our metro map using angular resolution and proximity constraints (Fig. 5.7(b)). However, the all-pairs approach has two downsides: Firstly, we get a significant increase in runtime, even without detecting intersections, since we use edge constraints for all-pairs of vertices in the network. Secondly, it is not clear, how exactly the all-pair edge constraints affect the layout of the drawing. Therefore we prefer the approach using angular resolution and proximity constraints.


(a) Geographical input network

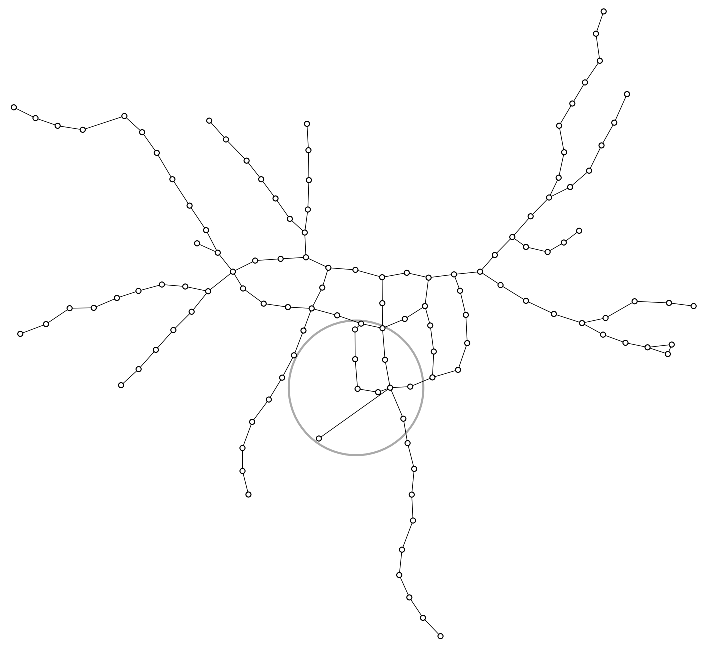
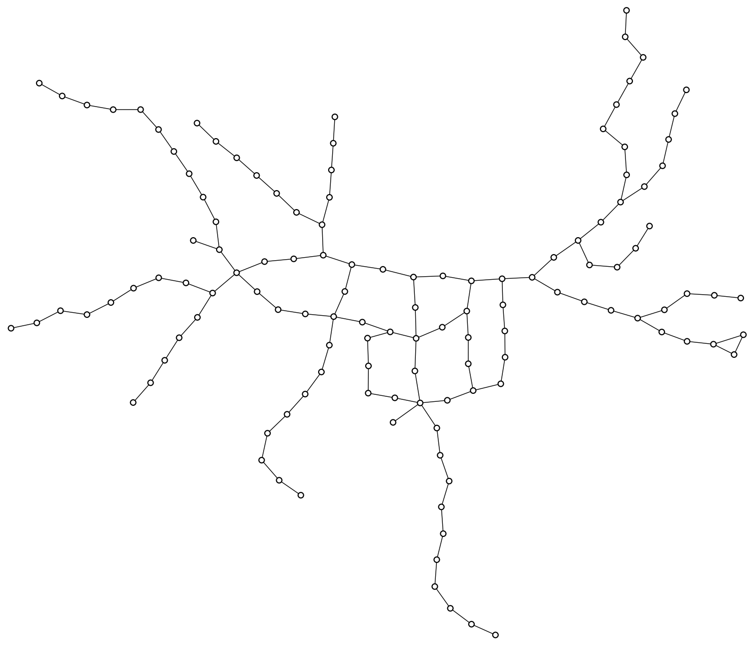

(b) All-pairs metro map of Karlsruhe

Figure 5.6: Geographical input network and all-pairs metro map of Karlsruhe computed by our algorithm.


(a) All-pairs metro map with weighting function $f_{\text{reciprocal},3}$

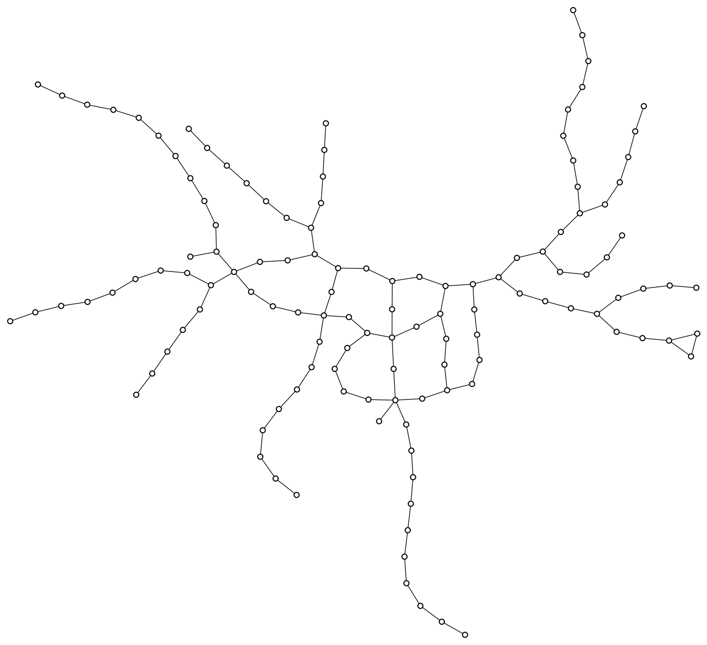

(b) Metro map using angle resolution and proximity constraints

Figure 5.7: All-pairs metro map of Karlsruhe computed with $f_{\text{reciprocal},3}$ as weighting function and metro map of Karlsruhe using angular resolution and proximity constraints.

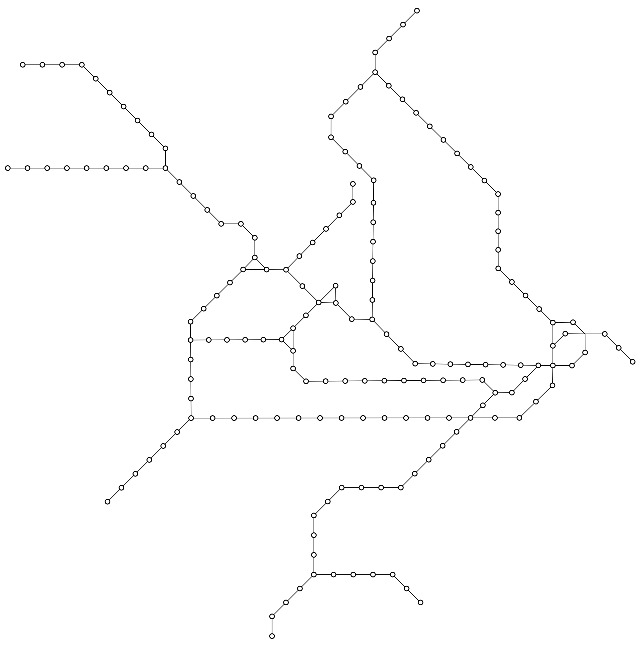
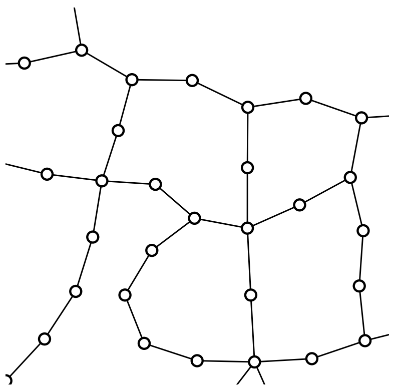


Figure 5.8: Metro map of Sydney computed by our algorithm using uniform edge length, proximity and octilinearity constraints. The port assignments were calculated based on the network of Fig. 5.5.

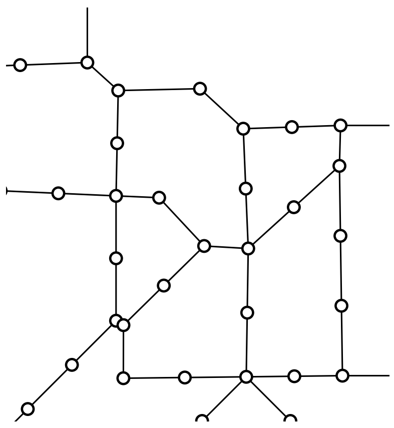
5.3 Octilinearity

We now combine our metro map algorithm with octilinearity constraints (Sect. 4.2) to get approximately octilinear metro maps. Note that deciding if a network can be drawn as an exactly-octilinear, planar metro map with preserved embedding is $\mathcal{NP}$-hard, as shown by Nöllenburg [18]. Figure 5.8 shows the metro map of Sydney computed by our algorithm using uniform edge length, proximity and octilinearity constraints. The network of Fig. 5.5 was used to compute the port assignments. Our algorithm achieves almost full octilinearity, while the drawing still resembles the geographic input network very closely. For the metro maps of London computed by our algorithm with and without octilinearity constraints see Appendix A.

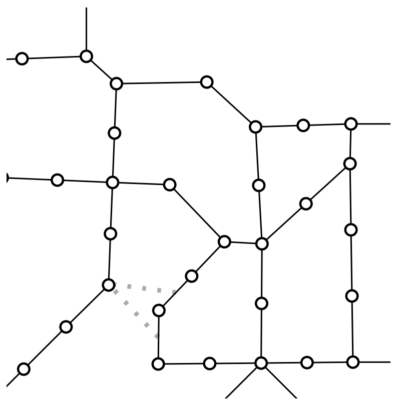
Next, we want to demonstrate how well proximity and octilinearity constraints with adjusted weights interact. Figures 5.9(b) and 5.9(c) shows two clippings of metro maps; both were calculated with uniform edge length and octilinearity constraints and both used 5.9(a) as input for port assignments. However, the second one does use proximity constraints, while the first does not. Without proximity constraints some vertices get very close to realise the octilinear directions in Fig. (b). To prevent this our algorithm inserts two proximity constraints in Fig. (c) (dotted lines); they keep these vertices apart. We still get a reasonable approximation of an octilinear drawing: due to the



(a) Input network



(b) Metro map using octilinearity constraints



(c) Metro map using octilinearity constraints and proximity constraints

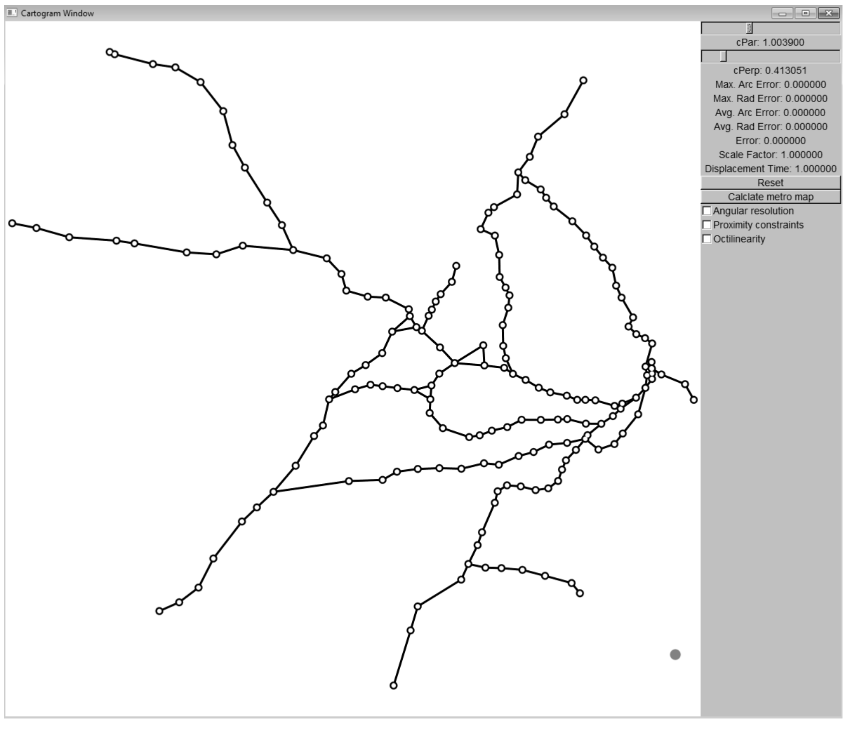
Figure 5.9: Input network for port calculations. Metro maps calculated with uniform edge length and octilinearity constraints. One with and the other without proximity constraints (dotted lines in (c)).

adjusted weights of octilinearity constraints towards direction realisation the algorithm is able to change the length of the appropriate edges.

Chapter 6

Experimental evaluation

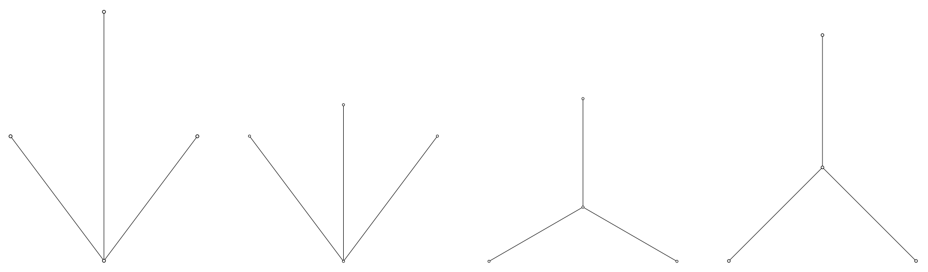
We have implemented the described algorithm in *C++*, using the library *Eigen* [12] for sparse linear algebra. Figure 6.1 shows a screen shot of our graphical user interface on Windows 7. All experiments have been run on a desktop PC with an AMD Phenom^{TM} II X6 1090T CPU at 3.20GHz. Memory usage is not an issue in these experiments, since we use a sparse representation of the matrices.



cPar: 1.003900
cPerp: 0.413051
Max. Arc Error: 0.000000
Max. Rad Error: 0.000000
Avg. Arc Error: 0.000000
Avg. Rad Error: 0.000000
Error: 0.000000
Scale Factor: 1.000000
Displacement Time: 1.000000

☐ Angular resolution
☐ Proximity constraints
☐ Octilinearity

Figure 6.1: Graphical user interface of our implementation on Windows 7.



(a) Original (b) Uniform edge length (c) Plus angular resolution (d) Plus octilinearity

Figure 6.2: The results of our algorithm applied on a network of three lines with uniform edge length, angular resolution and octilinearity constraints getting cumulatively added.

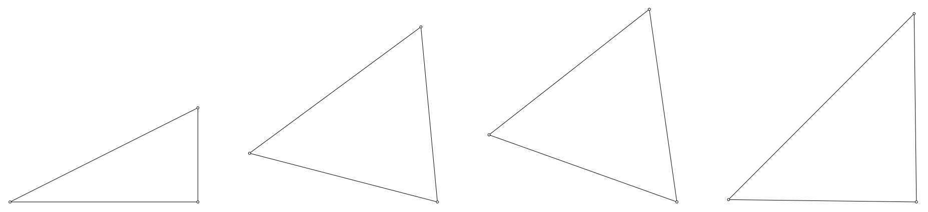
6.1 Simple examples

Firstly, we test the behaviour of our algorithm on simple examples. Uniform edge length, angular resolution and octilinearity constraints get examined. We use the event constraints from [22] to prevent intersections if necessary. Proximity constraints are not relevant here, since the network distance is not high enough in these examples.

The first network we try are three edges of different length sharing one common vertex. Figure 6.2 shows the original network and the resulting networks, when we cumulative add uniform edge length, angular resolution and octilinearity constraints. The results are as expected: the algorithm and the additions made work just as intended.

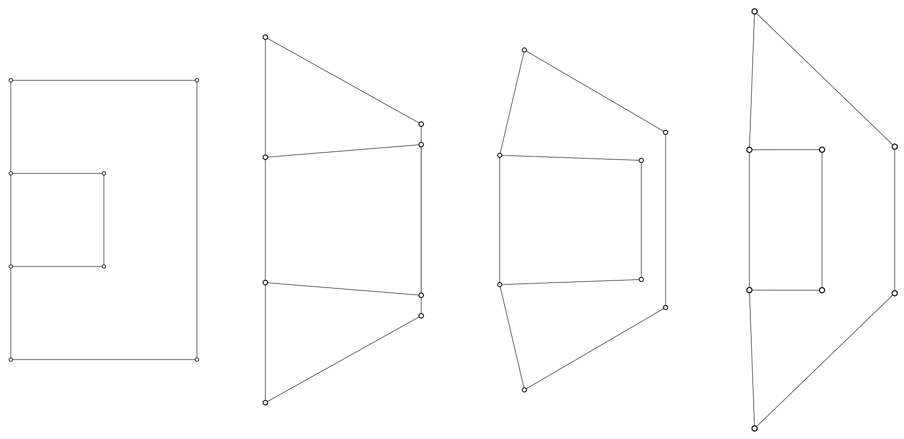
Next, we test a simple triangle similarly to the three lines (Fig. 6.3). When only uniform edge length is applied, the edge are only approximately equally long, since the edge directions have to be distorted to achieve this goal. However, when we add angular resolution, we practically get an equilateral triangle. Also, the drawing using octilinearity constraints is good, even if it is not fully octilinear: this is expected, since octilinearity and uniform edge length are contradicting demands for a triangle.

Now we examine a rectangle embedded in another rectangle (see Fig. 6.4). In this network uniform edge length is not really possible: the achieved result is reasonable, even though two edges are practically overlapping. Angular resolution helps avoiding this. We get a decent drawing using octilinearity constraints; the inner rectangle is squeezed horizontally due to intersection prevention.



(a) Original (b) Uniform edge length (c) Plus angular resolution (d) Plus octilinearity

Figure 6.3: The results of our algorithm applied on a triangle with uniform edge length, angular resolution and octilinearity constraints getting cumulatively added.



(a) Original (b) Uniform edge length (c) Plus angular resolution (d) Plus octilinearity

Figure 6.4: The results of our algorithm applied on a network of a rectangle embedded in another rectangle with uniform edge length, angular resolution and octilinearity constraints getting cumulatively added.

6.2 Experimental setup

Here we describe our experimental setup. We have used real-world road networks[1] and metro maps.[2] In these maps, we chose a set of *affected* edges and ask for their length to change; for the other edges we ask for the length to remain the same. Unless otherwise noted, we have chosen to affect all edges and set each L_e to the original length multiplied by a random factor between 1 and 4.

We first examine the impact of the parallel and perpendicular weights

[1]OpenStreetMap-based shapefiles from `http://download.geofabrik.de/`.

[2]We would like to thank Martin Nöllenburg for providing the geographic public-transport networks of various cities.

on the resulting direction and length discrepancies. Next we compare our approach to one iteration of by-the-book non-linear least squares. Finally we test the runtime on maps of different complexity.

We measure the direction error E_{dir} and length error E_{len} as the average of their respective part of the original non-linear objective function f_{polar} in (3.2). That is:

$$E_{\text{dir}} = \frac{\sum_{e \in E} W_\alpha(L_e) \cdot {\Delta\alpha_e}^2}{|E|}$$

$$E_{\text{len}} = \frac{\sum_{e \in E} W_{\text{r}}(L_e) \cdot {\Delta r_e}^2}{|E|}$$

The *overall* error is determined with

$$E_{\text{overall}} = \frac{\sum_{e \in E} f_{\text{polar}}(\Delta\alpha_e, \Delta r_e, L_e)}{|E|} \quad .$$

Notice that those errors are not scale invariant, because here they are not used as objective functions. However, this is not a problem, since we only use them for comparisons based on identical input maps. For Sect. 6.5, where we do not do comparisons, we measure the direction error and the (relative) length error as follows:

$$E^*_{\text{dir}} = \frac{\sum_{e \in E} |\Delta\alpha_e|}{|E|}$$

$$E^*_{\text{len}} = \frac{\sum_{e \in E} (|\Delta r_e| / L_e)}{|E|}$$

All reported average errors are calculated over 100 runs with different random lengths.

6.3 Direction and length weights

Here we test the ability of our perpendicular weight $W_\perp$ and parallel weight $W_{||}$ to actually effect a priority for correct directions or correct length. We also verify that the values of $W_\perp$ and $W_{||}$ in (3.7) are near optimal for our linear approximation f_{lin} of the non-linear objective function f_{polar} with weights W_α and W_{r} (see Chap. 3). We do this by changing the ratio of $W_{||}$ to $W_\perp$ by factors $\frac{1}{3}$ to 3 and measuring the effect on direction error, length error and the overall error. Notice that our basic algorithm only depends of the ratio between those weights, not their actual values. Hence, if we alter the ratio with a factor greater than 1 the length error should go down; for factors less than 1 the direction error should go down. The overall error should be minimal for our chosen weights $W_\perp$ and $W_{||}$.

Figure 6.5 shows the results of the experiment for the public transportation network of Sydney and London. The normalised values of direction, length and overall errors are shown on the y-axis and the factor with which the ratio of $W_{||}$ (W_{par}) and $W_{\perp}$ (W_{perp}) was altered on the x-axis. Every data point was averaged over 100 runs with random edge lengths, as specified in the experimental setup. We see that for both maps the direction error and length error behave as expected: if we increase their weight, the respective error goes down. Also, the overall error is near the minimum for factor 1 in both cases: the chosen values of $W_{\perp}$ and $W_{||}$ lead to good approximation of f_{polar}.

6.4 Comparison to non-linear least squares

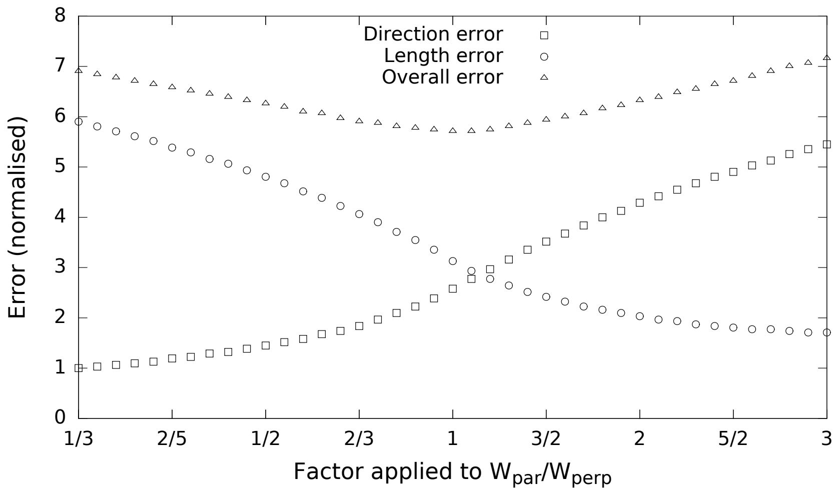
We compare our approach to one iteration of standard non-linear least squares on f_{polar}, using a Jacobian linear approximation (see Sect. 3.4). We take 100 runs with random edge lengths for the metro map of Sydney and London and calculate the error E_{overall} of our algorithm as well as the error E_{NLLS} of one Jacobian step.

Figure 6.6 shows histograms of the ratio of the error E_{NLLS} of one iteration of standard non-linear least squares to the error E_{overall} of our algorithm over 100 runs. We can see that our approach has smaller errors in all cases: For Sydney we get improvements from 20 to 100 percentage points. For London the improvements are in the range of 27.5 to 60 percentage points. A Wilcoxon test [16] showed in both cases that the overall error is significantly lower for our algorithm (p-value < 0.01, one-tailed test, $W(100) = 0$). We noticed that non-linear least squares often still has lower direction error, in exchange for much worse length error.

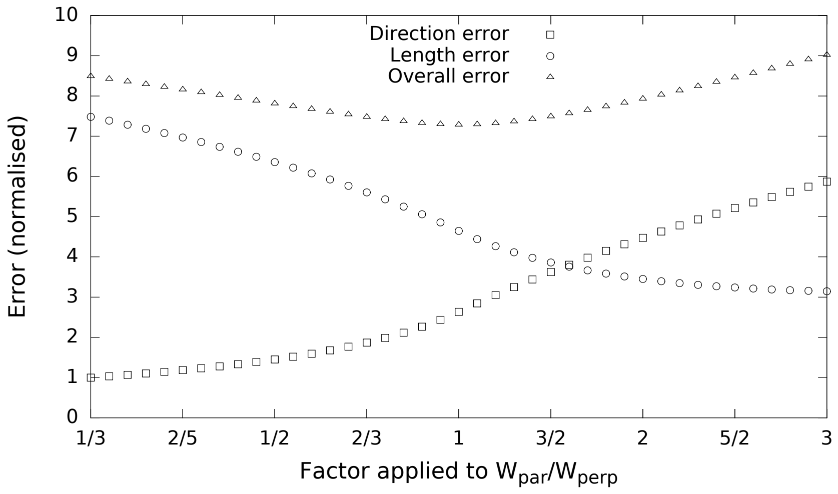
The average runtime of our approach is 0.5 ms for Sydney and 0.8 ms for London as opposed to 0.4 ms and 0.7 ms for one iteration of non-linear least squares. We can conclude that our method is an improvement to standard non-linear least squares, at the cost of a small increase in runtime.

6.5 Quality on realistic inputs

Next, we test the quality of our algorithm on more realistic inputs. We run two different experiments on the road network of the German city of Würzburg, consisting of 2511 vertices and 2995 edges (Fig. 6.7(a)). In the first experiment (“point”), we pick a random point on the map and select every edge that is no further away than 5% of the extent of the map. The random point is rejected if fewer than 10 segments are selected. Then we ask for these edges to be lengthened by a factor of 2 and for the other edges to be unaffected. This might represent the travel-time effect of an accident. Figure 6.7(b) shows an exemplary drawing for this experiment. In the the

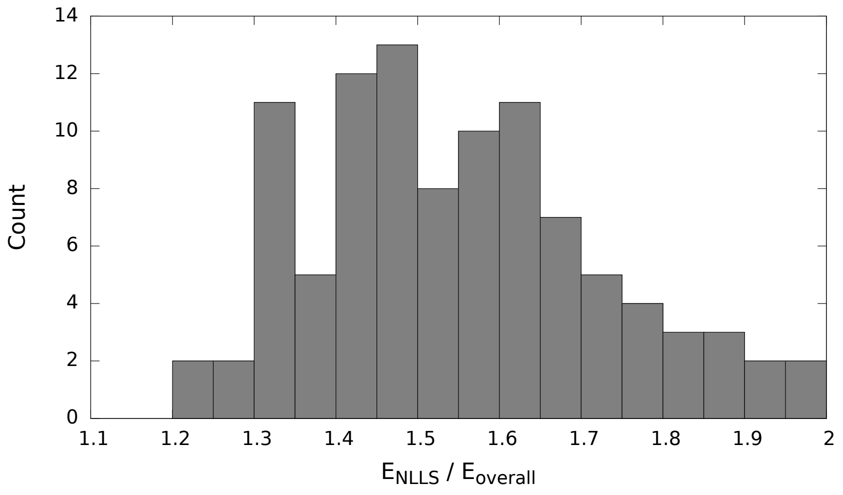


(a) Errors for Sydney's metro map

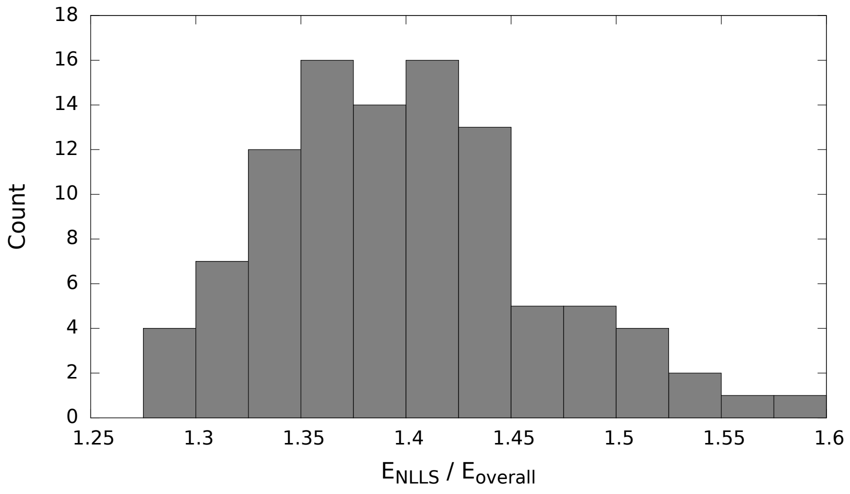


(b) Errors for London's metro map

Figure 6.5: Normalised direction, length and overall error over 100 runs, as a function of weight ratio factors, measured on the metro network of Sydney and London.



(a) Histogram for metro map of Sydney



(b) Histogram for metro map of London

Figure 6.6: Histograms for the ratio of the error E_{NLLS} of one iteration of standard non-linear least squares to the overall error E_{overall} of our algorithm over 100 runs for the metro maps of Sydney and London.

Experiment	Affected edges	Direction error	Rel. length error
Point	No	0.79°	0.007
	Yes	6.33°	0.106
Strokes	No	1.20°	0.012
	Yes	5.73°	0.216

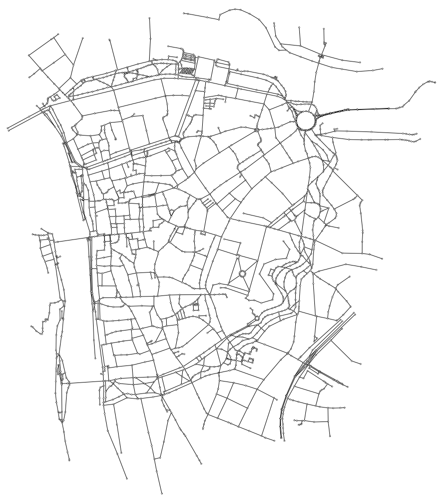
Table 6.1: Average direction error and average relative length error over 100 runs with edge length congested by factor 2 around a random point or on a subset of 16 strokes of the road map of Würzburg with 2511 nodes and 2995 edges.

second experiment (“strokes”), we select a random subset of 16 strokes in the road network and lengthen those in the same way, where a stroke is a path in the network where all edges have the same real-world street name. This represents congested traffic. We run both experiments 100 times. Event constraints are used to prevent intersections.

Table 6.1 shows the results of these experiments. We report the average direction error E^*_{dir} and the average relative length error E^*_{len} as defined in the experimental setup. This is done separately for the affected and unaffected edges, since the errors would always be low if we include many unaffected edges. We see that our algorithm effectively realises the unaffected edges with average direction errors of around 1° and relative length errors of around 1%. The effected edges are expectedly more difficult, we get direction errors of around 6° and relative length errors of around 11% for the experiment “point” and 22% for the experiment “strokes”. The reason for these high errors is that many parts of the input network are very dense, so that vertices are not able to move easily; this makes the realisation of congested edges hard. We can also see, that it is apparently easier to expand all edges around a point instead of edges on strokes.

Our next experiment tests the quality of our metro map approach; we ask our algorithm for uniform edge length, angular resolution and proximity constraints on various metro networks. Optionally, we additionally ask for octilinearity. Table 6.2 shows the results for the metro networks of Montreal, Karlsruhe, Sydney and London. For our metro maps without octilinearity we see relatively high direction errors between 8° and 13°; however, this is expected, since we ask for perfect angular resolution, which is often not possible with straight-line drawings. The requested edge lengths get realised well with errors between 2% and 5% for the simpler metro networks, and around 10% for the more complex network of London.

When asking for octilinearity the average direction error goes down significantly, to between 0.3° to 1.4°: we get a very good approximation of an octilinear drawing. We can see that the adjusted weights for the octilinearity constraints work really well, but of course this comes with the


(a) Road network of Würzburg

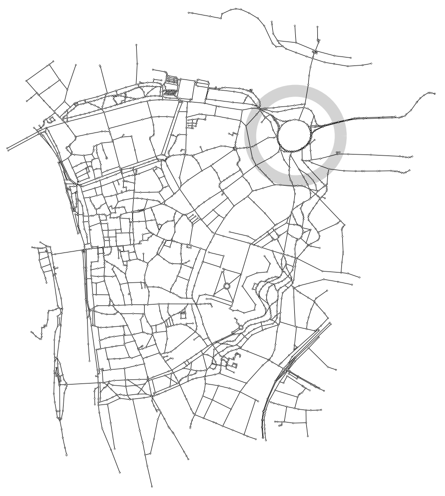

(b) Exemplary drawing for experiment "point"

Figure 6.7: The road network of the German city of Würzburg and an exemplary drawing for our experiment "point". The grey circle indicates the area of the affected edges.

cost of increased length errors. Still, the average relative length error only exceeds 6% for the metro network of London with around 15%. However, for octilinear drawings we do not care for length discrepancy that much anyway.

Table 6.3 shows the results of computing a focus-and-context metro map inspired by Wang and Chi [23]: we want every edge to have length 1, except for a single "focus" metro line with edges of length 2. We run this experiment with angular resolution, proximity and octilinearity constraints. The experiment is done for each metro line and the results are then averaged over all these runs. We see that the results are similar to our metro map

Map	$\|V\|$	$\|E\|$	Octilinearity	Direction error	Rel. length error
Montreal	65	66	No	8.32°	0.015
			Yes	0.28°	0.056
Karlsruhe	126	132	No	9.83°	0.017
			Yes	0.48°	0.047
Sydney	174	183	No	11.02°	0.045
			Yes	0.25°	0.048
London	308	361	No	13.10°	0.096
			Yes	1.34°	0.149

Table 6.2: Average direction error and average relative length error for metro maps computed by our algorithm.

Map	$\|V\|$	$\|E\|$	#Metro lines	Direction error	Rel. length error
Montreal	65	66	4	0.48°	0.059
Karlsruhe	123	132	10	0.26°	0.047
Sydney	174	183	10	0.44°	0.087
London	308	361	13	1.34°	0.159

Table 6.3: Average direction error and average relative length error over all metro lines of focus-and-context metro maps computed by our algorithm.

approach with octilinearity constraints. The main difference are the higher average relative length errors for Sydney and London, but they still remain within reasonable range.

Figure 6.8 shows an example for a focus-and-context metro map for the public transport network of Sydney of our approach and the algorithm of Wang and Chi. The bold edges represent the focused line. We think that our focus-and-context metro map is more advanced, since it resembles the geographical input map (Fig. 5.1(a)) more closely and realises the uniform inter-station distance better in the context part.

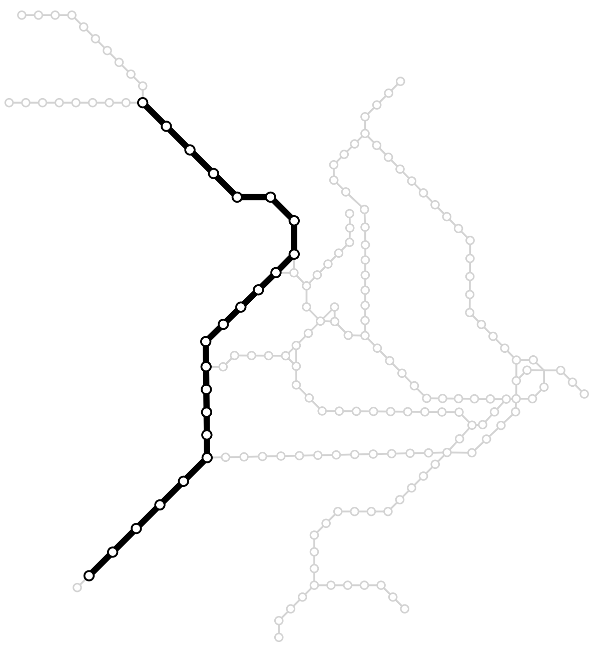
6.6 Runtime

We measure the runtime for various maps, again averaged over 100 runs with random edge lengths for each map as described in the experimental setup. Table 6.4 lists the maps, the number of vertices and edges as well as the following runtimes:

- Runtime R_E with basic edge constraints
- Runtime R_I with intersection prevention (using the event constraints)
- Runtime R_A with angular resolution
- Runtime R_O with octilinearity constraints

We test the different constraint types separately to see their influence on the runtime.

We can see that the runtimes with basic edge constraints mostly just depend on the number of edges of the map. The runtime with intersection prevention also depends on the map itself and the probability that intersections occur. For example, the runtime for the road network of Wüerzburg (Fig. 6.7(a)) increases significantly when we prevent intersections, since this network is very dense and complicated. For simpler networks, however, the runtime increase is not that drastic, even for the complex metro map of London. Angular resolution has only a small impact on the runtime: we just


(a) Focus-and-context metro map of our algorithm.

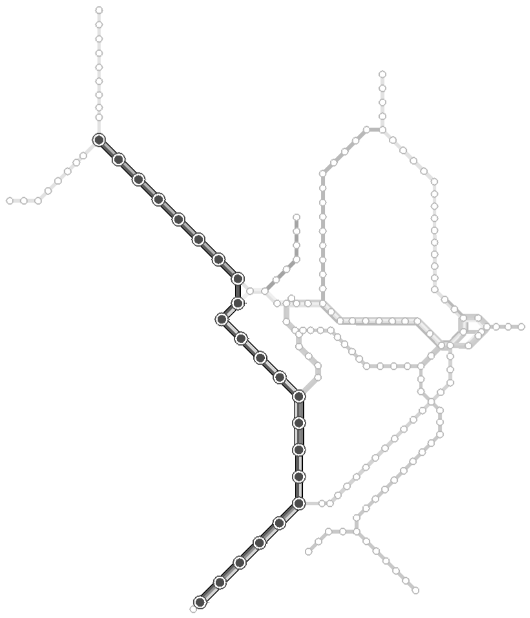

(b) Focus-and-context metro map of Wang and Chi.

Figure 6.8: Focus-and-context metro maps computed by our algorithm and by Wang and Chi. The bold edges represent the focused line.

Map	#Nodes	#Edges	R_{E} (ms)	R_{I} (ms)	R_{A} (ms)	R_{O} (ms)
Montreal	65	66	0.2	0.3	0.3	1.0
Karlsruhe	126	132	0.3	0.5	0.5	1.7
Sydney	174	183	0.4	0.5	0.6	2.3
London	308	361	0.8	2.0	1.2	4.5
Würzburg	2511	2995	8.3	120	11.7	40

Table 6.4: Average runtimes over 100 runs with random edge lengths as specified in Sect. 6.2 with basic edge constraints (R_{E}), with intersection prevention (R_{I}), with angular resolution (R_{A}) or with octilinearity constraints (R_{O}).

have to additionally calculate the optimal directions for every vertex. This can also be done as preprocessing when loading the network. Octilinearity constraints, on other hand, increase the runtime significantly. This is due to the fact that we have to run our optimisation two times, since the octilinear directions get calculated based on the network computed without octilinearity constraints. Also, we now have to solve a minimum-cost flow network for every vertex.

If we compare our runtime on the Sydney map with Wang and Chi [23] (see Fig. 6.8), we observe a large improvement from 816 ms to 15 ms on comparable hardware. It should be noted that their algorithm forces the drawing to be contained in certain boundaries, whereas we do not. However, our drawing can be resized to fit the boundaries. The runtime reported by Inoue and Shimizu [21] for computing a time-distance cartogram of a map of Japan with 81 vertices and 109 edges is about 80 ms, without intersection detection. Our algorithm would take less than 1 ms on such a map.

Chapter 7

Conclusion

We have introduced an algorithm for spatially-informative linear cartograms. Our algorithm uses a single run of least-squares optimisation, while other algorithm use iterative least-squares optimisation or some variant of multidimensional scaling, which typically also utilises iterative solvers. However, iterative solving increases runtime significantly and often has problems with convergence and bad local optima. We believe that least-squares optimisation is better suited for the computation of spatially-informative linear cartograms: edge length and direction, as shown in this thesis, can be reasonably encoded in linear constraints, whereas classical multidimensional scaling only cares about the edge length. It is also easy to incorporate modifications, like our realisation of given edge directions and proximity constraints, into least-squares optimisation.

Using an implementation of our proposed algorithm, we have shown that it is both efficient and effective. With a runtime in the order of tens of milliseconds, it is ideally suited for interactive applications, even on mobile devices. We have furthermore shown that our algorithm is able to realise the requested lengths and directions well in practice.

We think our results for drawing metro maps using unit-length linear cartograms are promising; perhaps as-is, or at least as a very efficient preprocessing step that automatically adjusts local scale to account for busy city centres and sparse suburbs. It improves angular resolution and smooths paths while keeping branches apart. Furthermore it can provide approximate octilinearity. All of these are desirable properties of metro maps.

Future work

One topic for future work is our choice of edge directions for angular resolution and octilinearity constraints, which is made vertex-wise. However, this often leads to disagreement for the edge direction for it's endpoints. It would be more desirable to choose the direction based on the whole graph.

Unfortunately we do not know if this problem can be solved efficiently. If this is not the case, it would be interesting to find a way to resolve disagreements for octilinearity in a better way than taking the mean direction. We would rather prefer for every edge to be actually assigned to a port and not possibly the middle between two ports.

Another topic regarding octilinearity is detecting and resolving malicious port assignments as seen in Fig. 4.9. This should be manageable especially if one of the involved vertexes has degree 2.

Vertexes with degree 2 are also interesting for angular resolution. In the ideal case angular resolution should evaluate paths in the network to straight lines if they share the same general direction. Our algorithm just smooths paths. Additional constraints for these cases should improve the results.

Proximity constraints are a big field for future work. We would like to adapt them to general linear cartograms. However, it is not clear how to choose the minimal margin between branches. For example, we can not take the minimal edge length, since there can be really short edge in a general linear cartogram. Also the considered graph distance can either be the actual graph distance, the number of hops or a combination of those two. We would also like to have an adapting margin for the proximity constraints, depending on how much space is available. Albeit it is not clear how this could be implemented.

As mentioned in Sect. 5.1 angular resolution does the ‘branch resolution’ on edges that are close in the network. Actually proximity constraints would not work for these edges; that is the reason why we introduced the dependence on graph distance with condition (5.1). The value 0.05, however, was chosen on experience only, because there is no clear cut-off for the impact radius of the angular resolution. Therefore optimising this value, perhaps even based on the network, would be preferable.

Appendix A

Metro maps of London

Here we show the metro maps of London computed with our two approaches introduced in Chap. 5, with and without octilinearity constraints. Figure A.1 shows the geographical input network of London's public transport system. See Fig. A.2 for the results of our first method utilising uniform edge length, angular resolution, proximity constraints and optionally octilinearity constraints. The results of our second approach are presented in Fig. A.3. Both approaches have difficulties realising uniform edge length and octilinearity; however, this is expected, since the metro network of London is rather dense and complicated. We think that our first method leads to better drawings in this instance and that they are of reasonable quality considering the complexity of the network.

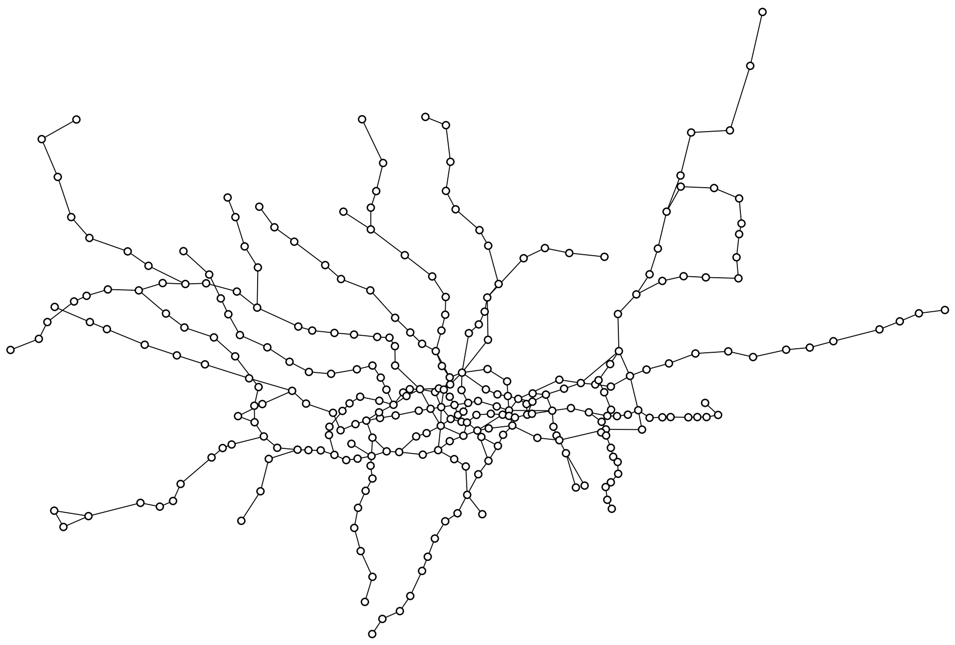
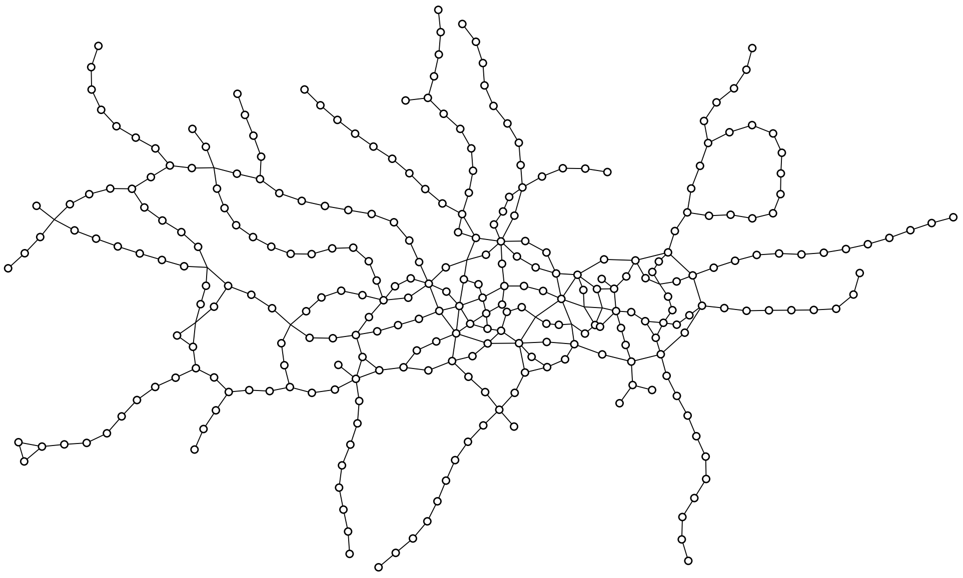
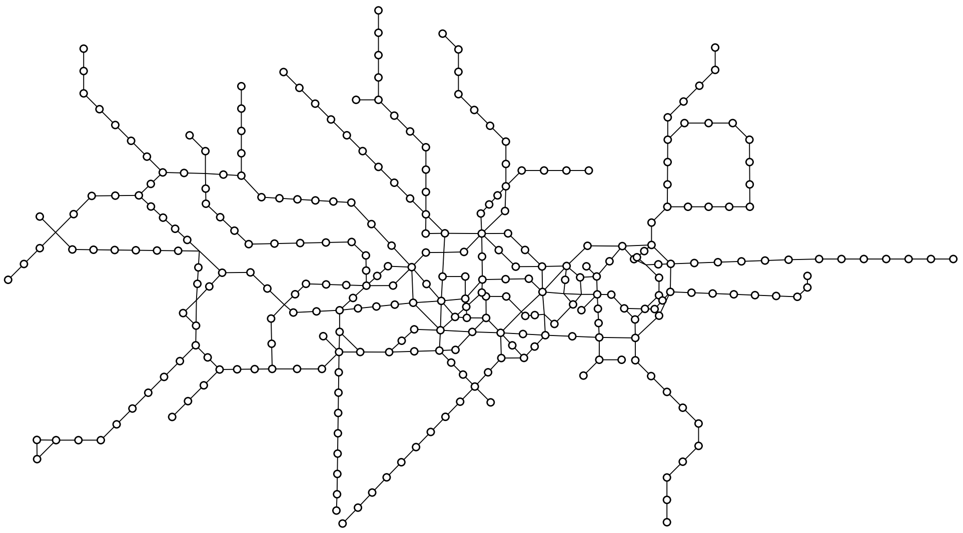


Figure A.1: Geographical input network of London

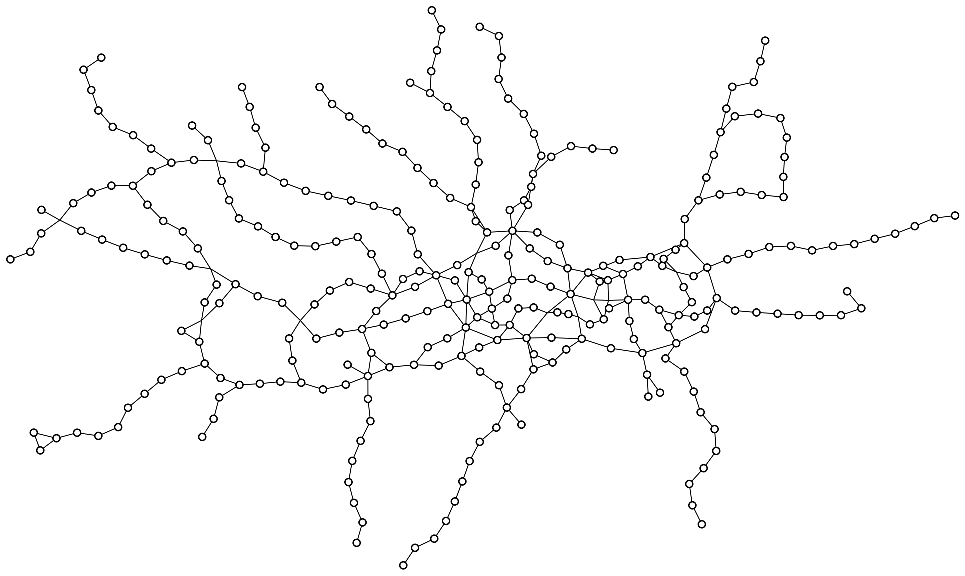


(a) Metro map of London without octilinearity constraints



(b) Metro map of London with octilinearity constraints

Figure A.2: Metro maps of London computed by our algorithm with uniform edge length, angular resolution, proximity constraints and optionally octilinearity constraints (Runtimes 46 ms and 68 ms).


(a) Metro map of London without octilinearity constraints

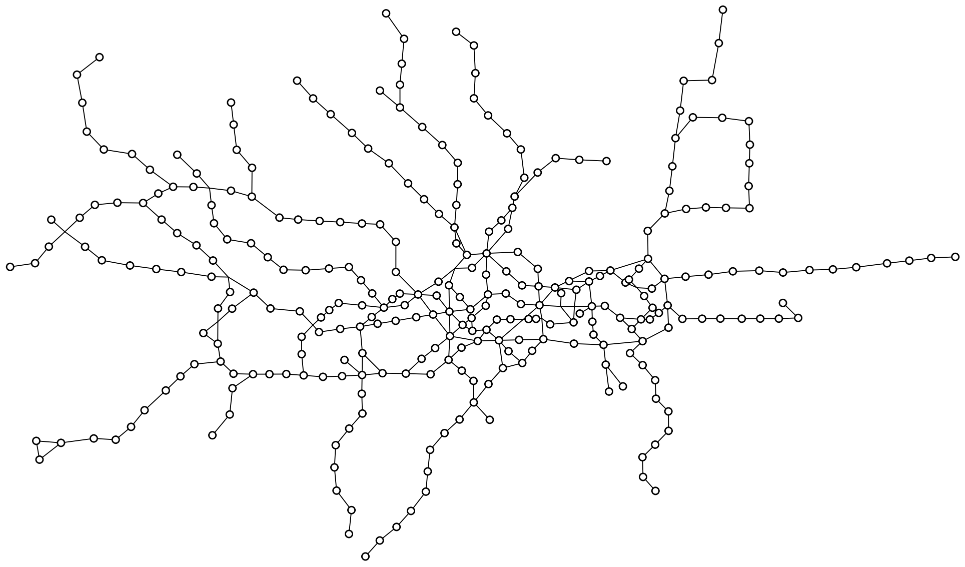

(b) Metro map of London with octilinearity constraints

Figure A.3: Metro maps of London computed by all-pairs metro map approach with and without octilinearity constraints.

Bibliography

[1] S. Bies and M. van Kreveld. Time-space maps from triangulations. In *Proceedings of the 20th International Conference on Graph Drawing*, GD'12, pages 511–516, Berlin, Heidelberg, 2013. Springer-Verlag.

[2] I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Applications*, chapter 11.4. Springer-Verlag, 1997.

[3] J. Böttger, U. Brandes, O. Deussen, and H. Ziezold. Map warping for the annotation of metro maps. *Computer Graphics and Applications, IEEE*, 28(5):56–65, Sept 2008.

[4] U. Brandes, G. Shubina, and R. Tamassia. Improving angular resolution in visualizations of geographic networks. In W. C. de Leeuw and R. van Liere, editors, *Data Visualization 2000*, Eurographics, pages 23–32. Springer Vienna, 2000.

[5] K. Buchin, A. van Goethem, M. Hoffmann, M. van Kreveld, and B. Speckmann. Travel-time maps: Linear cartograms with fixed vertex locations. In *Proceedings 8th International Conference on Geographic Information Science*, 2014.

[6] S. Cabello, E. D. Demaine, and G. Rote. Planar embeddings of graphs with specified edge lengths. *Journal of Graph Algorithms and Applications*, 11(1):259–276, 2007.

[7] J. W. Clark. Time-distance transformations of transportation networks. *Geographical Analysis*, 9(2):195–205, 1977.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[9] Y. Ding, N. Krislock, J. Qian, and H. Wolkowicz. Sensor network localization, Euclidean distance matrix completions, and graph realization. *Optimization and Engineering*, 11(1):45–66, 2010.

[10] A. Efrat, D. Forrester, A. Iyer, S. G. Kobourov, C. Erten, and O. Kilic. Force-directed approaches to sensor localization. *ACM Trans. Sen. Netw.*, 7(3):27:1–27:25, October 2010.

[11] E. R. Gansner, Y. Hu, and S. North. A maxent-stress model for graph layout. *IEEE Transactions on Visualization and Computer Graphics*, 19(6):927–940, June 2013.

[12] G. Guennebaud, B. Jacob, et al. *Eigen v3*, 2010. `http://eigen.tuxfamily.org`.

[13] C. Kaiser, F. Walsh, C. J. Q. Farmer, and A. Pozdnoukhov. User-centric time-distance representation of road networks. In S. I. Fabrikant, T. Reichenbacher, M. van Kreveld, and C. Schlieder, editors, *Geographic Information Science*, volume 6292 of *Lecture Notes in Computer Science*, pages 85–99. Springer Berlin Heidelberg, 2010.

[14] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, April 1989.

[15] K. Kraus. *Photogrammetry – Geometry from Images and Laser Scans*. de Gruyter, Berlin, Germany, 2004.

[16] R. Lowry. *Concepts and Applications of Inferential Statistics*, 2013. `http://vassarstats.net/textbook/ch12a.html`.

[17] D. Merrick and J. Gudmundsson. Increasing the readability of graph drawings with centrality-based scaling. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation - Volume 60*, APVis '06, pages 67–76, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[18] M. Nöllenburg. Automated drawings of metro maps. Technical report, Fakultät für Informatik, Universität Karlsruhe, 2005.

[19] M. Nöllenburg 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.

[20] J. B. Saxe. Embeddability of weighted graphs in k-space is strongly NP-hard. *Proceedings of the 17th Allterton Conference in Communications, Control and Computing*, pages 480–489, 1979.

[21] E. Shimizu and R. Inoue. A new algorithm for distance cartogram construction. *International Journal of Geographical Information Science*, 23(11):1453–1470, 2009.

[22] T. C. van Dijk and J.-H. Haunert. Interactive focus maps using least-squares optimization. *International Journal of Geographical Information Science*, 2014.

[23] Y.-S. Wang and M.-T. Chi. Focus+context metro maps. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2528–2535, Dec 2011.

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Masterarbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde vorgelegt habe.

....
Würzburg, 5.September 2014

Inhalt der CD-ROM

- Masterarbeit im PDF-Format
- In der Ausarbeitung verwendete Grafiken
- Mathematica-Datei für die gemachten Berechnungen
- Programmdatei (64-bit) unserer Implementierung
- Quellcode-Repository unserer Implementierung