DOI: 10.1111/cgf.13986
Eurographics Conference on Visualization (EuroVis) 2020
M. Gleicher, T. Landesberger von Antburg, and I. Viola
(Guest Editors)

Volume 39 (2020), Number 3

Metro Maps on Octilinear Grid Graphs

Hannah Bast[1], Patrick Brosi[1] and Sabine Storandt[2]

[1]University of Freiburg, Department of Computer Science, Freiburg, Germany
[2]University of Konstanz, Department of Computer and Information Science, Konstanz, Germany

Abstract

Schematic transit maps (often called "metro maps" in the literature) are important to produce comprehensible visualizations of complex public transit networks. In this work, we investigate the problem of automatically drawing such maps on an octilinear grid with an arbitrary (but optimal) number of edge bends. Our approach can naturally deal with obstacles that should be respected in the final drawing (points of interest, rivers, coastlines) and can prefer grid edges near the real-world course of a line. This allows our drawings to be combined with existing maps, for example as overlays in map services. We formulate an integer linear program which can be used to solve the problem exactly. We also provide a fast approximation algorithm which greedily calculates shortest paths between node candidates on the underlying octilinear grid graph. Previous work used local search techniques to update node positions until a local optimum was found, but without guaranteeing octilinearity. We can thus calculate nearly optimal metro maps in a fraction of a second even for complex networks, enabling the interactive use of our method in map editors.

CCS Concepts

• ***Human-centered computing*** → ***Graph drawings;*** • ***Theory of computation*** → *Integer programming;* • ***Mathematics of computing*** → *Approximation algorithms;*

1. Introduction

Maps of public transit networks usually depict the lines in a schematized way to ensure readability. In 1931, Harry Beck presented his idea to draw the London subway lines as alternating sequences of horizontal, vertical and diagonal line segments [Gar94]. This octilinear design has since become the de facto standard and its usage goes beyond the cartographic representation of public transit networks.

The high practical relevance of these maps has lead to numerous approaches to render them automatically. We give an overview of existing work in Section 1.3. However, existing methods usually do not guarantee octilinear results, require impractically long solution times and/or only allow a small fixed number of bends (or none at all) along edges in the final drawing. This leads to several restrictions in their practical applicability. In particular, previous work which guaranteed octilinear results often did not have solution times fast enough to be used interactively in a map editor. Additionally, we are not aware of any previous work which allows octilinear drawings to approximate the real geographical courses of a line between stations, which is a requirement if the final maps should be combined with either existing maps or satellite imagery. In this work, our goal is to overcome these restrictions.

Our approach is to search for a metro map drawing in a specially crafted octilinear grid graph. For each input station, a suitable grid node is determined, and connecting line segments are paths on the grid graph between these nodes. The optimal assignment of nodes and edges can be found via an integer linear program. More importantly, our approach allows for an efficient approximation algorithm which is based on iterative shortest path calculations on the grid graph and can produce solutions of high quality in a fraction of a second even for complex networks.

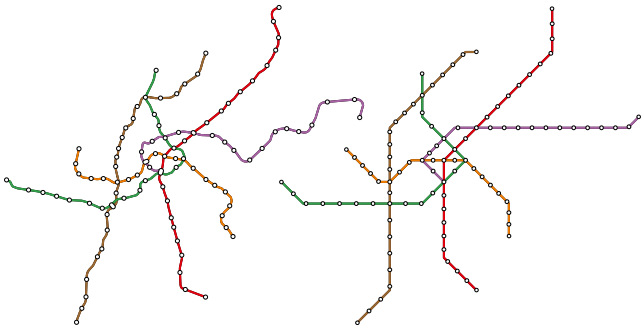

Figure 1: *Left: The Vienna subway network, drawn with real-world geographical station positions and line courses. Right: Octilinear drawing by our approx. approach. Octilinearization took 202 ms.*

1.1. Contributions

We consider the following as our main contributions:

• We provide a novel formulation of the problem of drawing a metro map on an octilinear grid graph which allows an arbitrary number of edge bends between stations.

• We formulate an integer linear program (ILP) to optimize the problem exactly.
• As ILP solution times are too long for bigger networks, we describe a fast approximation algorithm which produces solutions in under 3 seconds and has an approximation error of under 7.5% on our testing datasets (for which an optimal solution can be found) when degree 2 nodes are contracted first and later re-inserted equidistantly.
• We describe how our approximation algorithm can be sped up by a simple A^* heuristic.
• We evaluate our approach on six datasets (Freiburg, Stuttgart, Vienna, Berlin, London and Sydney). Our maps can be inspected online: http://octi.cs.uni-freiburg.de

1.2. Problem definition

Given an undirected labeled input graph $G = (V, E, L)$, where V are stations, E are connections between those stations and each edge $e \in E$ is labeled with a subset $L(e) \subseteq \mathcal{L}$ of lines traveling on this edge. We call G the *line graph*. We say $\mathcal{D}_G = (P, C)$ is a drawing of G, where $P(v) \in \mathbb{R}^2$ assigns a position to every node $v \in V$ and $C(e) = (q_0, q_1, \ldots, q_{n-1}), q_i \in \mathbb{R}^2$ assigns a piece-wise linear curve to every edge $e \in E$. The initial input drawing $\mathcal{D}^*_G$ assigns each node a geographical position (Fig. 1, left). Optionally, each edge may be assigned its real-world course. Our goal is to find a schematic drawing $\mathcal{D}'_G$ that resembles a classic metro map (Fig. 1, right). This is often formalized as a set of hard and soft constraints [NW05, NW11]. We consider these hard constraints:

1. *Octilinearity*. Each edge curve $C(e)$ may only consist of segments whose orientation is a multiple of 45°.
2. *Topology Preservation*. The topology of the line graph must be respected. No crossings between edges must be introduced, non-incident edges must not share common points and the circular edge ordering around nodes must be preserved.
3. *Map Density*. The distance from each station to all other curve anchor points must be above a given threshold $\hat{d}$.

We consider the following soft constraints (see Equation 8 for how we combine them into one objective function):

1. *Edge Monotony*. The number of edge bends should be minimized and large angles preferred.
2. *Edge Length*. Edge curve lengths should be minimized.
3. *Geographical Accuracy*. The original station positions should be changed as little as possible.

Soft constraint 3 (Geographical Accuracy) is usually weakened to only apply to nodes with a degree different than 2 (intersection nodes and terminus nodes), as this both improves the overall map appearance and simplifies the problem. Nodes of degree 2 are then contracted prior to drawing and later re-inserted equidistantly onto the final drawing. We call this the deg-2 heuristic.

Previous work defined the problem as finding an octilinear *embedding* of the input graph, i.e. each edge is represented by a straight octilinear arc. We are interested in octilinear *drawings* and therefore use a slightly different approach. We state the problem as finding the optimal positions $P(v) \in \mathbb{N}^2$ on a grid for each station node and curves $C(e) = (q_0, \ldots, q_{n-1})$, $q_i \in \mathbb{N}^2$ connecting them. Most importantly, for two succeeding points $q_i = (x_i, y_i)$ and $q_{i+1} = (x_{i+1}, y_{i+1})$, we require their Chebyshev distance $D_{\text{Ch}}(q_i, q_{i+1}) = \max\{|x_{i+1} - x_i|, |y_{i+1} - y_i|\}$ to be exactly 1, which ensures octilinearity of the final drawing.

To project this grid onto a map plane, we use a scale factor D, which is essentially the height and length of a grid cell. If D is set to $\hat{d}$, a minimum distance between any two grid points is guaranteed to be greater or equal to $\hat{d}$. The grid size $X \cdot Y$ is determined by the bounding box of the input line graph. In particular, $X \cdot Y = \lceil A/D^2 \rceil$, where A is the area of this bounding box.

1.3. Related Work

As metro map layouts and octilinear drawings are of high practical relevance, they both have been studied extensively in the past.

For metro maps, it was observed early that multiple criteria are important to produce visually pleasing and informative layouts. In [SR04], a fitness score was introduced which incorporates the number of edge crossings, edge lengths, node distribution, angular resolution, direction changes, and other layout aspects to judge its quality. A hill climbing heuristic was applied to find a layout with a high fitness score. Similar heuristic approaches were described in [HMDN04], [HMdN06] and [SRMOW10]. While such approaches might result in useful layouts, they come without any quality guarantees; and expressing all layout aspects with a single fitness score might obfuscate better trade-offs. In [NW05] and [NW11], the metro map layout aspects were subdivided in hard and soft constraints, and a mixed-integer program was used to ensure that all hard constraints (among them octilinearity of all edges) are fulfilled. This method comes with some restrictions on the input graph (planarity, maximum degree of 8, the latter being also a restriction of our approach) and only allows edges without bends. The observed running times were high already for small networks. In [BNUW06, BKPS07], the metro-line crossing minimization problem (MCLM) was introduced. Here, the goal is to draw a set of simple paths (representing metro lines) along the edges of an embedded underlying graph with a minimum number of pairwise crossings. In [BBS19], a pipeline was presented in which a problem similar to MCLM was solved efficiently to obtain transit maps with few line crossings and few line separations. However, the goal there was to come up with transit maps in which the abstraction from the real-world course of the lines was negligible. Hence map schematization and in particular octilinearity were not considered in this context.

Octilinear drawings, in which every edge of a given graph is drawn as an alternating sequence of horizontal, vertical and diagonal line segments, have been studied in different contexts before. In [BGKK14], it was proven that for planar input graphs with low degree there always exists an octilinear drawing with at most one direction change (or bend) per edge when using an integer grid where the number of grid nodes is polynomial in the number of input points. The general problem of finding planar octilinear drawings with a minimum number of bends is NP-hard. In [Nöl05], an NP-hardness proof for the problem of deciding whether a given embedded graph can be drawn using only straight octilinear edges was given. Our goal is to find an optimal octilinear drawing, i.e., edges are allowed to be represented by octilinear paths rather than just

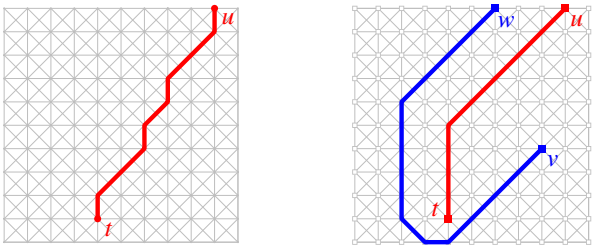

Figure 2: *Left: Shortest path between t and u on a grid graph with uniform edge cost. Path bends are not minimized. Right: Two shortest paths for (t,u), and (v,w) on our octilinear grid graph with uniform grid edge cost 2 and additional path bend penalties $c_{135} = 1$, $c_{90} = 2$ and $c_{45} = 3$. Path (t,u) acts as an obstacle for (v,w).*

by a single segment but planarity is not a necessary prerequisite. In this configuration, our problem has resemblance to the k-node disjoint path problem. There, given a graph and k node pairs, the goal is to find a set of node-disjoint paths connecting each pair. The problem was proven to be NP-hard even on grid graphs [CK15]. As our model allows to assign arbitrary costs to the edges in the octilinear grid graph, the k-node disjoint path problem on grids can be seen as a special case of our problem with diagonal costs set to infinity. Our problem therefore is NP-hard, too.

A recent vision paper on passenger navigation in metro networks [CL19] emphasizes the importance of (dynamically adaptable) octilinear drawings of metro maps for guiding passenger flows. But current methods which are fast enough to produce schematic metro maps on demand (as e.g. described in [CY14], [vDL18], [WC11] and [WP16]) either compromise octilinearity or topology preservation, and do not allow for adapting the visualization to different application scenarios. Our pipeline will be shown to be very flexible, and to produce nearly optimal octilinear drawings quickly.

The metro map layout problem is often considered in conjunction with station labeling. In [NW11], an extension of the mixed-integer linear program for drawing the metro lines allows to guarantee enough space around station markers for labels. In [WTLY12] and [WTH*13], this was extended to work with large annotation labels like photographs. In [WC11] and [WP16], labels are placed after the metro map layout was obtained by minimizing an energy function which captures labeling aspect as their directions, spaces between labels and coherence among labels of the same line. While labeling is not the main focus of this paper, we sketch how labels can be included in our model towards the end of the paper.

Finally it should be noted that metro map layout algorithms are not only relevant for visualizing the tracks of real trains, but also for visualizing 'trains of thought'. In fact, the term *metro map metaphor* was coined specifically to capture non-spatial information visualization problems with similar visualization demands as metro maps [SGSK01, Nes04, SRB*05]. While our evaluation in this paper sticks to actual metro maps, our pipeline works for any graph with given node coordinates and optional edge shapes.

2. Octilinear Grid Graph

We use an auxiliary undirected grid graph $\Gamma = (\Psi, \Omega)$ with diagonal edges (Fig. 2, left), where Ψ are the grid nodes and

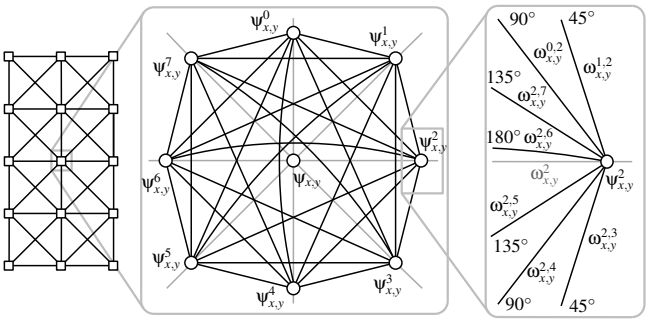

Figure 3: *A $3 \cdot 5$ octilinear grid graph. Each node $\psi_{x,y}$ has 8 ports $\psi^0_{x,y}, \ldots, \psi^7_{x,y}$ which are connected to $\psi_{x,y}$ by direct sink edges $\omega^0_{x,y}, \ldots, \omega^7_{x,y}$. Each port is additionally connected to its 180°, 135°, 90° and 45° neighbor ports by bend edges $\omega^{i,j}_{x,y}$.*

Ω the grid edges. For every position (x,y) on an $X \cdot Y$ grid, we add a grid node $\psi_{x,y}$. Each $\psi_{x,y}$ is connected to its neighbors $N^0(\psi_{x,y}), \ldots, N^7(\psi_{x,y})$, where $N^0(\psi_{x,y})$ is the north neighbor of $\psi_{x,y}$, $N^1(\psi_{x,y})$ the north-east neighbor, etc. A path $p = (\psi_0, \psi_1, \ldots, \psi_{n-1}), \psi_i \in \Psi$ is then an octilinear curve with cost $c(p) = (n-1) \cdot c_h$, where c_h is the hop cost of using a grid edge.

2.1. Penalizing Line Bends

To optimize soft constraint 1 (Edge Monotony) between stations, we additionally want the cost for a path in Γ to reflect the number of bends. A bend penalty should also reflect its angle - either 135°, 90° or 45°. We call these penalties c_{135}, c_{90} and c_{45}. A straight pass through a node should go unpunished, so $c_{180} = 0$. Since we aim for a smooth path through Γ, we favor obtuse angles and require $c_{180} \leq c_{135} \leq c_{90} \leq c_{45}$.

We extend our cost function and now want to search for the path $p = (\psi_0, \psi_1, \ldots, \psi_{n-1})$ which minimizes the cost

$$c(p) = (n-1) \cdot c_h + \sum_{i=1}^{n-2} c_b(\psi_{i-1}, \psi_{i+1}), \tag{1}$$

where $c_b(\psi_{i-1}, \psi_{i+1})$ is the angular bend cost between adjacent edges $\{\psi_{i-1}, \psi_i\}$ and $\{\psi_i, \psi_{i+1}\}$, that is, either 0, c_{135}, c_{90} or c_{45}.

We model this by adding 8 auxiliary port nodes $\psi^0_{x,y}, \ldots, \psi^7_{x,y}$ to every grid node (Fig. 3). Each port again corresponds to an outgoing angle in clockwise fashion and is connected to $\psi_{x,y}$ via sink edges $\omega^0_{x,y}, \ldots, \omega^7_{x,y}$. These sink edges allow us to leave and reach an original grid node $\psi_{x,y}$. For paths passing through $\psi_{x,y}$, we connect each port $\psi^i_{x,y}$ with its 7 - i succeeding (in clockwise fashion) sibling ports at position j with bend edges $\omega^{i,j}_{x,y}$ (Fig. 3, right).

To distinguish the different node and edge types, we define the set of original grid nodes as $\Psi^g \ni \psi_{x,y}$, the set of port nodes as $\Psi^p \ni \psi^i_{x,y}$, the set of bend edges as $\Omega^b \ni \omega^{i,j}_{x,y}$, the set of sink edges as $\Omega^s \ni \omega^i_{x,y}$ and the set of original grid edges as Ω^g. Additionally, for each $\psi \in \Psi$, we define $\psi^* \in \Psi^g$ to be the original grid node belonging to ψ (this may be ψ itself). For ease of notation, we denote by $P(\psi)$ the projected coordinates of the original grid node $\psi^* = \psi_{x,y}$ belonging to ψ.

As we want to prevent the use of sink edges in pass-through

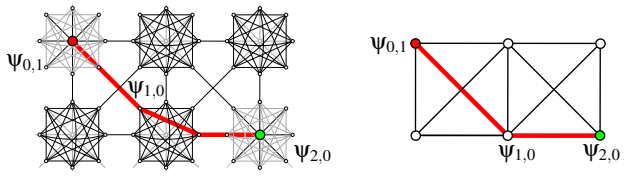

Figure 4: *The same path* $(\psi_{0,1}, \psi_{1,0}, \psi_{2,0})$ *on two grid graph variants. Left: with bend edges. Right: without bend edges.*

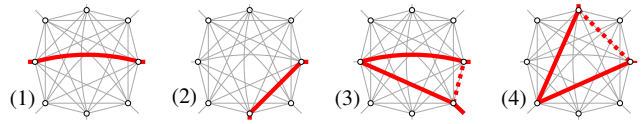

Figure 5: *1.* 180° *pass through a node* ψ*. 2. A* 90° *pass. 3.* 45° *pass simulated by a* 180° *and* 135° *pass. 4.* 90° *pass simulated by two* 135° *passes.*

nodes, we set a uniform sink cost c_s high enough so that a sink edge is always more expensive than a bend edge, for example $c_s = c_{45}$. In a shortest path from s to t, the only sink edges are then a leaving sink edge at s and an arriving sink edge at t (Fig. 4, left).

2.2. Modeling Edge Costs

As both a 45° and a 90° bend edge may be substituted by cheaper edges, special care has to be applied to the modeling of the actual edge costs. For example, a 45° bend may be replaced by a 180° bend edge and a 135° bend edge (Fig. 5.3). Similarly, a 90° bend can be substituted by two cheaper 135° bend edges (Fig. 5.4).

To prevent such shortcuts, we introduce a constant $a \geq 0$ and offset the bend costs by a. We call the updated bend edge costs $c'_{180}, \ldots, c'_{45}$ and choose a so that the following inequalities hold:

$$2c'_{135} = 2a + 2c_{135} \geq a + c_{90} = c'_{90} \tag{2}$$

$$c'_{135} + c'_{180} = 2a + c_{135} + c_{180} \geq a + c_{45} = c'_{45}. \tag{3}$$

Inequality 2 ensures that simulating a 90° bend with two 135° passes is never cheaper than c'_{90}. Inequality 3 ensures that simulating a 45° bend with a 135° pass and a 180° pass is never cheaper than c'_{45}. They are fulfilled for $a = c_{45} - c_{135}$.

A shortest path p' on our octilinear grid graph with bend penalties will now consist of *two* nodes for each original grid node: for the start and end node, the original grid node and a single port node appear in the path (Fig. 4). For pass-through nodes, a port node for arriving at and a port node for leaving the original grid node appears (w.l.o.g., we ignore the case where a bend edge is replaced by two bend edges with similar cost, as this does neither affect the final path on the original grid graph, nor the cost of that path).

This shortest path $p' = (\psi_0, \psi_1, \ldots, \psi_{n'-1})$ thus always describes a path $p = (\psi^*_0, \psi^*_2, \psi^*_4, \ldots, \psi^*_{n'-1})$ on the corresponding original grid graph, with $|p| = n = |p'|/2 = n'/2$. It has the cost

$$c'(p') = \overbrace{2c_s}^{\text{sink edges}} + \underbrace{(n-1)\cdot c_h}_{\text{grid hops}} + \overbrace{\sum_{i=1}^{n-2} a + c_b\left(\psi^*_{2i-2}, \psi^*_{2i+2}\right)}^{\text{offsetted bend costs}}. \tag{4}$$

To remove the constant offset a, we set $c_{135} = 1$, $c_{90} = 1.5$ and $c_{45} = 2$, which means $a = c_{45} - c_{135} = 1$. If we then set the cost of a single hop $c_h = 1 = a$ and the actual grid edge costs to $c'_h = c_h - a = 0$, we can rewrite Equation 4 as

$$c'(p') = 2c_s + (n-1)\cdot c'_h + \sum_{i=1}^{n-2} c_h + c_b\left(\psi^*_{2i-2}, \psi^*_{2i+2}\right) \tag{5}$$

$$= 2c_s + (n-2)\cdot c_h + \sum_{i=1}^{n-2} c_b\left(\psi^*_{2i-2}, \psi^*_{2i+2}\right) \tag{6}$$

$$= 2c_s + c(p) - c_h = c(p) + 2c_s - 1. \tag{7}$$

As the shortest path p' thus minimizes $c(p) + 2c_s - 1$, it also minimizes $c(p)$. Note that since we set the actual grid edge cost c'_h to 0, we are able to introduce arbitrary offset costs to individual grid edges by setting the cost of that grid edge to the offset cost. This will be used in Section 6.1 to prefer grid edges close to the original course of an input edge.

3. Optimal Solution via ILP

Using the octilinear grid graph, we now define the problem of finding the optimal drawing on Γ like this: find grid nodes $\psi(v)$ for each input node v and non-intersecting shortest paths in the grid graph between the grid nodes for adjacent input nodes. This means that for each edge $e = \{v, u\}$ in the input graph, we search for a shortest path in Γ between the grid nodes for v and u and for an optimal assignment of input nodes to grid nodes at the same time. This solution should minimize the sum of (1) all path costs, (2) the distances between the original and the grid position of an input station node, and (3) the line bend penalties at stations (which are not covered by the path costs on Γ). Formally, we want to minimize

$$t(\mathcal{D}_G) = \sum_{e \in E} c(p(e)) + \sum_{v \in V} d(v, \psi(v)) \frac{c_h + c_m}{D} + c^v_b, \tag{8}$$

where $p(e)$ is the path through Γ for input edge e, c is the cost of the path as defined in Equation 1, $\psi(v)$ is the grid node v was assigned to, $d(v, \psi(v))$ is the euclidean distance between the input node v and its grid node $\psi(v)$ and c^v_b is the sum of the bend costs $c_b(p_a(e), p_b(f))$ between adjacent grid edges $p_a(e)$ and $p_b(f)$ of input edges e and f (both adjacent at v and with at least one line in common, that is $L(e) \cap L(f) \neq \emptyset$). As a move penalty too low will result in positions which minimize the total path costs, we normalize the distance $d(v, \psi(v))$ by the grid cell size D and multiply this normalized distance by the hop cost c_h as well as an additional explicit move penalty c_m.

This section describes how to optimize $t(\mathcal{D}_G)$ using an integer linear program (ILP). We first model each undirected grid edge $\{\psi, \psi'\}$ as a pair of directed edges (ψ, ψ') and (ψ', ψ). To be able to later retrieve the placement of stations, we add binary decision variables $x_{v,\psi}$ for each input node v and grid node $\psi \in \Psi^g$ which should be 1 of v was assigned to ψ, or 0 otherwise. $x_{v,\psi}$ is added to the objective function with $d(v, \psi(v)) \frac{c_h + c_m}{D}$ as a coefficient.

To be able to retrieve the course of the shortest paths, we define binary variables $x_{e,\omega}$ for each input edge e and each grid edge ω which should be 1 if ω is used in the path for e, or 0 otherwise. $x_{e,\omega}$ is added to the objective function with the cost of ω as a coefficient.

3.1. Station Placement

To ensure that each input node v is assigned to exactly one grid node $\psi_{x,y}$, we add the following constraint:

$$\forall v \in V : \sum_{\psi \in \Psi^g} x_{v,\psi} = 1. \tag{9}$$

A grid node may either be assigned a single input node, used as a pass-through for a single input edge, or be not used at all. We enforce this with the following set of constraints:

$$\forall \psi \in \Psi^g : \sum_{v \in V} x_{v,\psi} + \sum_{e \in E} \sum_{\omega \in \Omega^b_\psi} x_{e,\omega} \leq 1, \tag{10}$$

where Ω^b_ψ is the set of bend edges adjacent to ψ. If an input station is assigned to ψ, the first sum is already 1, forbidding further use. Similarly, if ψ is used as a pass-through, it cannot be assigned to an input station node or be used as a pass-through by another path. Note that Equation 10 also enforces that a grid edge ω can only be used by a single path, as a second path using ω would either arrive or pass through a grid node already used by the other path.

3.2. Edge Continuity

To globally compute the optimal paths on Γ between all adjacent input nodes, we build on the standard formulation of the shortest path problem as a linear program. We first have to make sure that edges assigned to the path from $\psi(s)$ to $\psi(t)$ are connected. We add the following constraints (where $e = \{s,t\}$):

$$\forall e \in E \; \forall \psi \in \Psi^p : \quad \sum_{\omega \in \text{out}(\psi)} x_{e,\omega} - \sum_{\omega \in \text{in}(\psi)} x_{e,\omega} = 0, \tag{11}$$

$$\forall e \in E \; \forall \psi \in \Psi^g : \quad x_{t,\psi} - 2x_{s,\psi} + \sum_{\omega \in \text{out}(\psi)} 2x_{e,\omega} - \sum_{\omega \in \text{in}(\psi)} x_{e,\omega} = 0. \tag{12}$$

Equation 11 guarantees that the number of outgoing and incoming edges at each port node is the same. Equation 12 handles $\psi(s)$ and $\psi(t)$. Here we count an outgoing edge twice, which means that the grid node could only make up for an outgoing edge with two incoming edges. This, however, would mean that our shortest path split somewhere, which is prevented by Equation 11. The only way to fulfill the constraint is thus for ψ to be the source node $\psi(s)$ for the input edge. Similarly, the only way to counter an incoming edge is for ψ to be the target node $\psi(t)$ for the input edge.

If we allow grid edges of cost 0, we also have to ensure that a pair of directed grid edges (ψ,ψ') and (ψ',ψ) is not activated as a stray circular path of cost 0 by the ILP solver. This is enforced by the following set of constraints:

$$\forall (\psi,\psi') \in \Omega^g : \sum_{e \in E} x_{e,(\psi,\psi')} + x_{e,(\psi',\psi)} \leq 1. \tag{13}$$

3.3. Preservation of Topology

To preserve the input topology, it suffices to ensure that no two paths in Γ (which both correspond to a single edge in the planar input graph G) intersect and that the circular ordering of adjacent edges is the same as in G. Equation 10 already prevents paths crossing at grid nodes. As our octilinear grid graph is not planar, we also have to prevent crossings at intersecting grid edges. We define Ω^d as the set of diagonal grid edges and say that for $\omega \in \Omega^d$, $\omega^\times \in \Omega^d$ is the diagonal edge crossing ω. We then add the constraint

$$\forall \omega \in \Omega^d : \sum_{e \in E} x_{e,\omega} + x_{e,\omega^\times} \leq 1. \tag{14}$$

To respect the original circular edge ordering, we would first like to have a variable $\delta_{v,e} \in \{0,\dots,7\}$ which tells us the octilinear direction of input edge e at adjacent input node v in the final drawing. To get the desired assignments, we add the following constraints:

$$\forall e = \{s,t\} \in \Omega : \left(\sum_{\psi \in \Psi^g} \sum_{p=1}^{7} p x_{e,(\psi,\psi^p)} \right) - \delta_{s,e} = 0 \tag{15}$$

$$\forall e = \{s,t\} \in \Omega : \left(\sum_{\psi \in \Psi^g} \sum_{p=1}^{7} p x_{e,(\psi^p,\psi)} \right) - \delta_{t,e} = 0. \tag{16}$$

In Equation 15, each outgoing sink edge (ψ,ψ^p), $p \in 0,\dots,7$ adds p to the sum. As Equations 12 and 10 ensure that only a single outgoing sink edge may be used by the path for an input edge, the left side of the equation is guaranteed to equal the octilinear direction $0,\dots,7$ of $e = \{s,t\}$ at s. The only way to fulfill the constraint is then to set the value of $\delta_{s,e}$ to the octilinear direction. Equation 15 is modeled equivalently for incoming paths at t.

To finally preserve the circular edge ordering, we employ a technique originally used in [Nöl05]. If $v_0,\dots,v_p,\dots,v_{\deg(u)-1}$ is the clockwise ordering of nodes adjacent to u in the original drawing, then $\delta_{v,(v,u_p)} < \delta_{v,(v,u_{p+1})}$ has to be true for all but one $p \in \{0,\dots,\deg(u)-1\}$ in the final octilinear drawing. This can be enforced with the following constraints, which we add for all $u \in V$ with $\deg(u) > 2$ and where $\beta_{p,v}$ is a new binary variable:

$$\delta_{v,(v,u_{p+1})} - \delta_{v,(v,u_p)} + 8 \cdot \beta_{p,v} \geq 1, \tag{17}$$

$$\sum_{p=0}^{\deg(u)-1} \beta_{p,v} \geq 1. \tag{18}$$

If $\delta_{v,(v,u_p)} \not< \delta_{v,(v,u_{p+1})}$, then $\delta_{v,(v,u_{p+1})} - \delta_{v,(v,u_p)} \leq 0$ and the only way to fulfill Equation 17 is to set $\beta_{p,v}$ to 1. Equation 18 ensures that this can only happen once.

3.4. Avoiding Line Bends

So far, our shortest paths only penalize line bends along paths. We have to ensure that bends at input nodes are equivalently penalized. We would like to have binary variables telling us whether edges e and f in the input graph describe a 45°, 90°, 135° or 180° bend at their joint node in the final drawing.

We first note that for two directional variables $\delta_{u,e}$ and $\delta_{u,f}$, $\delta_{u,e} - \delta_{u,f} \mod 8$ is either 1 or 7 for 45° bends, 2 or 6 for 90° bends, 3 or 5 for 135° bends and 4 for 180° bends. As modulo cannot be used directly in an ILP, we use the following equivalent constraint for each pair e,f of edges in the input graph adjacent at node u and sharing a line:

$$0 \leq \delta_{u,e} - \delta_{u,f} + 8\gamma_{ef} \leq 7, \tag{19}$$

where γ_{ef} is an auxiliary binary variable which will be 1 if $\delta_{u,f} > \delta_{u,e}$. We then have $\delta_{u,e} - \delta_{u,f} + 8\gamma_{ef} = \delta_{u,e} - \delta_{u,f} \mod 8$, which we will denote by $\Delta_{e,f}$. We now add binary decision variables

$\Delta^0_{e,f}, \ldots, \Delta^7_{e,f}$ for each of the 8 possible values of $\Delta_{e,f}$ and the following constraint for each pair of adjacent input edges:

$$\Delta_{e,f} - \sum_{i=0}^{7} i\Delta^i_{e,f} = 0. \tag{20}$$

To ensure that only one of the bend decision variables is set to 1, we add the following constraint:

$$\sum_{i=0}^{7} \Delta^i_{e,f} = 1. \tag{21}$$

Each of the 8 bend decision variables is then added to the objective function with its corresponding penalty.

3.5. ILP Size

For an $X \cdot Y$ grid, our octilinear grid graph has $\Theta(XY)$ edges and nodes. For the station placement, we need $\Theta(|V|XY)$ variables. Additionally, Equations 9 and 10 add $\Theta(|V|XY)$ constraints. For the shortest path calculations, we need $\Theta(|E|XY)$ variables and constraints. For the preservation of the input topology, Equation 14 adds $\Theta(XY)$ constraints. Equations 15 and 16 add $\Theta(|E|)$ constraints. Equations 17 and 18 add $\Theta(|V|)$ constraints. Finally, for avoiding line bends and input nodes, we add at most $8 \cdot 7$ auxiliary variables per input node (as there can be at most $8 \cdot 7$ edge pairings per input node in our octilinear setting) and at most $8^2 \cdot 7$ bend variables per input node. Similarly, Equations 19, 20 and 21 all add at most $8 \cdot 7$ constraints per input node, so the overall number of constraints and variables for the line bend penalty at input nodes is $\Theta(|V|)$. The total number of constraints and variables in our ILP is thus $\Theta(|E|XY + |V|XY) = \mathcal{O}(|E|XY) = \mathcal{O}(|E| \cdot \lceil A/D^2 \rceil)$.

4. Approximate Solution

ILP solution times tend to get very big for complex input graphs (Table 2). This section addresses the need for a fast method that works well in practice. We describe a fast algorithm to solve the problem approximately. Our method works as follows: (1) Order the edges of the input graph. (2) For each (ordered) input edge $e = \{u, v\}$, calculate the shortest path from a set of possible start nodes S to a set of possible target nodes T on the grid graph (if the grid node for u or v has already been settled, the corresponding node set has size 1). Paths already calculated act as obstacles. (3) If no initial drawing could be found, randomize the ordering and try again. (4) Optimize the initial drawing via a local search approach where individual nodes are moved to one of their 8 neighboring positions and adjacent edges re-routed.

4.1. Input Edge Ordering

In addition to the standard degree $\deg(v)$ of an input node v, we define the line degree $\ldeg(v)$ of a node to be the number of (non-unique) lines on each adjacent edge. We then establish an initial input edge ordering $(e_0, e_1, \ldots, e_{|E|-1})$ as follows: (1) Mark all input nodes as unprocessed. (2) Take the unprocessed node v with highest line degree $\ldeg(v)$ and mark it as dangling. (3) As long as there are dangling nodes, take the dangling node v_d with highest line degree and add all adjacent edges $\{v_d, u_0\}, \ldots, \{v_d, u_k\}$ leading to an unprocessed node u_i to the edge ordering, where the u_i are sorted in descending order w.r.t. $\ldeg(u_i)$. Mark each u_i as dangling, and v_d as processed. (4) If there are no dangling nodes anymore, but unprocessed nodes remain, then the input graph was not connected. In this case, we start again at (2). Figure 6, left gives an example.

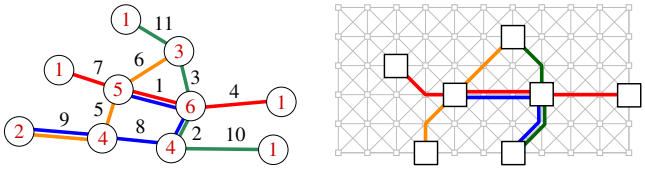

Figure 6: *Left: Input line graph with edge processing order (black) and node line degrees (red). Right: Line graph on grid after the seventh edge has been routed.*

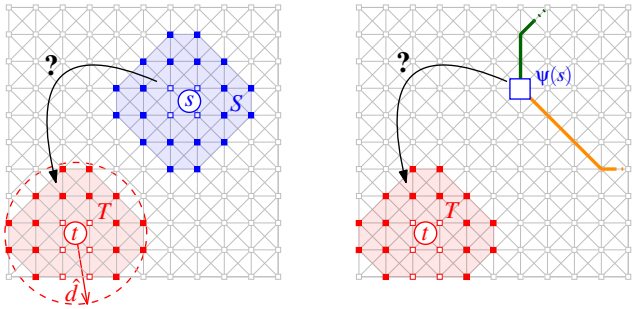

Figure 7: *Left: Routing an edge on the grid graph from grid node candidates S for input node s to grid node candidates T for input node t (both within a distance $\hat{d}$ around t and s). The grid nodes depicted as full rectangles constitute the hull of the S and T, respectively. Right: routing an edge on the grid graph from an already settled grid node ψ to a set T of target node candidates.*

4.2. Edge Routing and Station Placement

With an initial edge ordering at hand, we route each input edge $e_i = \{v, u\}$ through the octilinear grid graph iteratively (Fig. 6, right). As the grid nodes for v and u may be still unknown, we route between node sets S and T. S consists of candidate grid nodes for s, T are candidate grid nodes for t. As candidates, we use all grid nodes within a distance r around the input node's original position (Fig. 7, left). If both s are t are not settled yet, it may happen that S and T are not disjoint. To prevent this, we build a local Voronoi diagram: we define $ST = S \cup T$ and say that for each $\psi_{x,y} \in ST$, $\psi_{x,y}$ is moved into S if it is nearer to s than t, or else into T.

To prefer grid nodes near the original position of the input node, we offset the cost of each sink edge adjacent to the candidate grid node with the corresponding distance penalty. As in the ILP formulation, this penalty is the distance between $p(v)$ and $p(\psi)$ normalized by D and multiplied by $c_h + c_m$, where c_m is the move penalty.

Afterwards, we calculate the set-to-set shortest path between S and T using a standard implementation of Dijkstra's algorithm. If s or t were not settled before, they are now settled to the start (and/or end) node of the resulting shortest path.

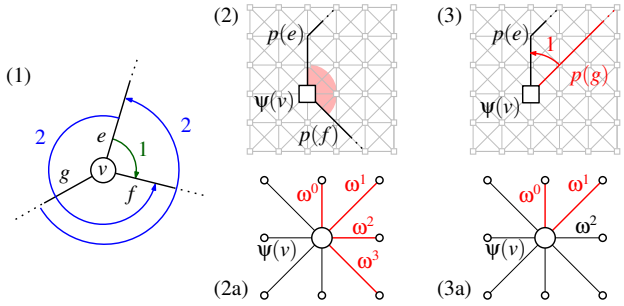

Figure 8: *(1) Part of an input graph G. Edges e and f have a clockwise (green) distance of 1 at v, meaning that there are no edges between them in an angular, clockwise ordering. Their counterclockwise distance (blue) is 2. (2) Paths p(e) and p(f) have been routed. The red area is now blocked for path p(g). (3) Paths p(e) and p(g) have been routed, but the placement of p(g) means the original edge ordering cannot be respected for p(f).*

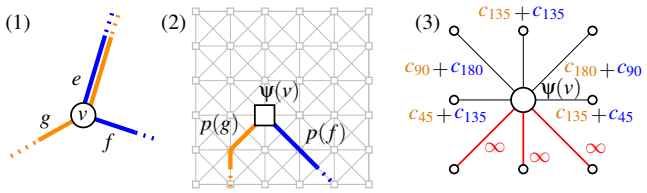

Figure 9: *(1) Part of an input line graph with a node v and three adjacent edges e, g, f. (2) g and f have already been routed on the grid graph, v has been settled at ψ(v). (3) The sink edge costs at ψ(v) prior to routing e. As v has been settled, the bend edges at ψ(v) are closed and are not depicted. Already used or blocked sink edges (red) have cost ∞. Unused sink edges have costs equal to the bend costs induced by letting p(e) leave in the corresponding direction.*

4.3. Preservation of Embedding

To prevent paths crossing each other, we update the grid graph after each Dijkstra run. Grid nodes that were used in the previous shortest path are both *bend-closed* by setting the cost of each adjacent bend edge to ∞ and *sink-closed* by setting the cost of each adjacent sink edge to ∞. This prevents further paths to use any of these nodes. If a node $\psi_{x,y}$ was settled as the source or target node for the previously routed edge, it may later be used again as a source or target node. We then have to re-open its sink. To prevent crossing paths at diagonal grid edges, we close for each diagonal grid edge used in the previously found path all crossing diagonal grid edges by setting their cost to ∞.

To preserve the circular edge ordering at nodes, we update the costs of adjacent sink edges for the grid node used for s, and the grid node used for t. We consider Figure 8.2. The routing order at v is (e, f, g), and paths $p(e)$ and $p(f)$ have already been routed. To respect the input edge ordering at v, we have to make sure that $p(g)$ does not enter or leave $\psi(v)$ in the area marked red. We achieve this by closing each sink edge ω^p that lies between ω^0 and ω^3 prior to routing $p(g)$ by setting its cost to ∞ (including the sink edges used by $p(e)$ and $p(f)$) (Fig. 8.2a).

We now consider Figure 8.3 and assume the routing order at v to be (e, g, f). Paths $p(e)$ and $p(g)$ have already been routed. However, there is no sink edge left at $\psi(v)$ to route $p(f)$ in such a way that the input edge ordering is preserved. To prevent dead-ends like this, we make sure that there are always enough sink edges left in both directions of f. For example, the clockwise distance between e and g at v in the input graph is 2, their counterclockwise distance is 1 (Fig. 8.1). If $p(e)$ is already routed (Fig. 8.3), we now close the sink edge ω^0 used by $p(e)$, but also the succeeding (in clockwise direction) sink edge ω^1 (Fig. 8.3a). This ensures that any path $p(g)$ found for g will now leave one sink edge open between $p(e)$ and $p(g)$, allowing $p(f)$ to later respect the original edge order at v.

4.4. Avoiding Line Bends

Just as in the ILP formulation, our shortest paths so far only optimize line bends along paths. We have to also ensure that line bends at nodes are equivalently penalized. We handle this similar to the edge ordering constraints in the previous section. Assume a grid node ψ that was already settled for an input node v with adjacent edges (in routing order) $e_0, \ldots, e_i, \ldots, e_{\deg(v)-1}$. Prior to routing an edge e_i, we calculate the line bend penalty between every routed edge e_j, $j < i$ and e_i for each of the possible placements of e_i on adjacent sink edges. The sum of the line bend penalties on each adjacent sink edge is then used as the cost for this sink edge (Fig. 9).

4.5. Complexity

To establish the initial input edge ordering, we have to process $|V|$ nodes and find the node with lowest line degree $|V|$ times in the worst case. This can be done in $\mathcal{O}(|V| \log |V|)$. To assign each v a set of node candidates, we have to make $\mathcal{O}(|V| \cdot |\Psi|)$ distance calculations. For the edge routing, we need $|E|$ Dijkstra runs on the grid graph Γ, which takes $\mathcal{O}(|E| \cdot (|\Omega| + |\Psi| \log |\Psi|))$. After each run, we have to close the used nodes, which takes time $\mathcal{O}(|E| \cdot |\Psi|)$, and update the sink edge costs at up to two newly settled nodes. As the degree of input nodes is at most 8 and if we assume that we can check whether two input edges share a common line in constant time, the latter can be done in constant time. Note that $|\Psi| \in \mathcal{O}(|\Omega|) = \mathcal{O}(XY)$ as we use an octilinear grid graph, $|V| \in \mathcal{O}(|E|)$ as we can ignore input nodes of degree 1 and $|V| \leq |\Psi|$ as the input graph is trivially undrawable if there are more input nodes than grid nodes. The total running time of our approximation approach is therefore $\mathcal{O}(|E| \cdot (XY + XY \log XY)) = \mathcal{O}(|E|XY \cdot (1 + \log XY)) = \mathcal{O}(|E| \cdot A/D^2 \cdot \log A/D^2)$, where A is again the bounding box area of the input line graph.

4.6. Optimization via Local Search

To further polish the look of our final maps, we employ a local search approach. Given an octilinear drawing $\mathcal{D}_G^0$ on a grid graph Γ, we define the local neighborhood of $\mathcal{D}_G^0$ as the set of drawings where exactly one grid node position of an input node v_m is moved to one of the 8 neighboring grid position (if free). For each neighbor of the current drawing, we remove the paths for each edge adjacent to v_m, move v_m to the designated new position, re-route all adjacent edges in clockwise ordering and calculate the new overall score of the drawing. At the end, the best neighbor is taken, and the optimization proceeds from there (Figure 10).

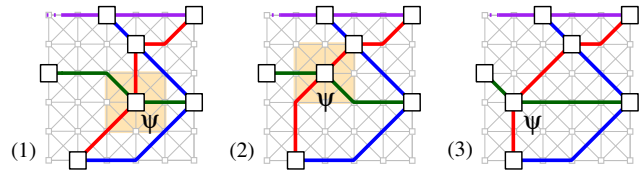

Figure 10: *Optimizing a drawing (1) by exploring the local search neighborhood, consisting of the 8 neighboring positions (depicted here for node ψ) for each settled grid node. All edges adjacent to ψ are re-routed after it is moved to a neighboring position, and the position which yields the biggest improvement is taken (2, 3).*

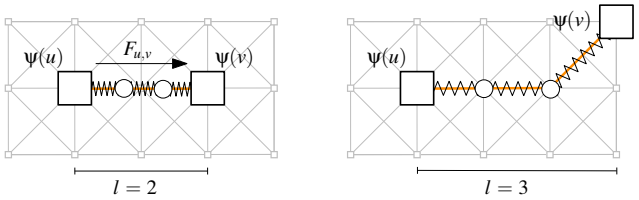

Figure 11: *Left: $k = 2$ stations are re-inserted equidistantly onto an octilinear curve with L^∞ length $l = 2$. The hypothetical spring stores a potential energy of $\frac{1}{4}c$. Right: Node $\psi(v)$ is moved to relax the spring, the potential energy is now 0.*

4.7. Optimizing Distances Between Contracted Nodes

The deg-2 heuristic mentioned in Section 1.2 contracts all nodes of degree 2 and later re-inserts them equidistantly. It is then no longer guaranteed, however, that there is a minimal distance $\hat{d}$ between adjacent stations, as the edge they are re-inserted onto may not be long enough. We consider the octilinear drawing before the re-insertion of the contracted nodes. For an input edge e with an octilinear curve $C(e)$ built of l grid edges and containing k contracted stations, we define a spring force $F_{u,v} = \frac{c}{k} \cdot (k+1-l)$, where c is a penalty factor. This spring is relaxed if the L^∞ (grid edge) length l of e is such that we can insert k stations equidistantly with distance 1 (Fig. 11). If $F_{u,v} > 0$ (that is, if the hypothetical spring is compressed), we add the potential energy $E_{u,v} = \frac{c}{2k} \cdot (k+1-l)^2$ to our objective function during the local search optimization phase.

4.8. Speed-up Heuristic

The octilinear grid graph allows for a simple cost heuristic for the path-finding step. Given a (set) of target nodes, a cost heuristic $h(\psi)$ gives an estimate of the shortest path cost from ψ to any target node. It is called *admissible* if it never overestimates the real cost.

We reconsider Figure 7. In the right example, the cost for reaching a grid node (or any of its port nodes) in T from a grid node ψ outside of T is at least the cost of reaching one of the grid nodes that constitute the hull of the node candidates. To estimate the shortest path cost from ψ to the hull, we use the fact that to reach a grid node $\psi_{x,y}$ (or any of its port nodes) from a grid node $\psi_{x',y'}$ (or any of its port nodes) we have to pass through at least $D_{\text{Ch}}((x',y'),(x,y)) - 1$ grid nodes. We can thus use the following heuristic:

$$h(\psi) = \min\left\{D_{\text{Ch}}\left(P(\psi),P(\psi')\right) - 1 | \psi' \in H_T\right\}, \qquad (22)$$

as each shortest path has to take some bend edge at the pass-through nodes and the cost of this bend edge is at least $c'_{180} = 1$.

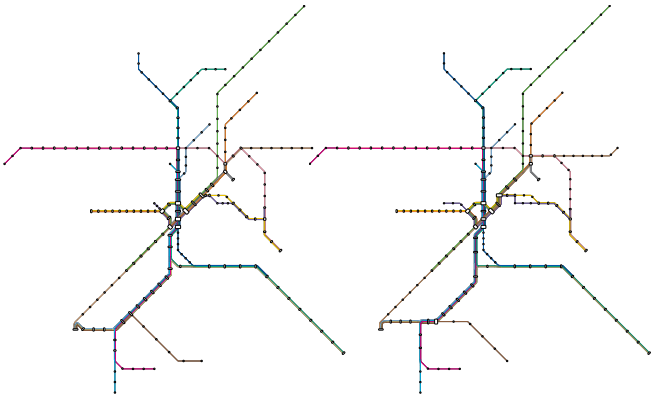

Figure 12: *Right: Map of the Stuttgart light rail network, octilinearized with our approach A-2+D with grid size $D = 0.75 \cdot \bar{d}$ in 843 ms. Left: Same map, before the local search phase.*

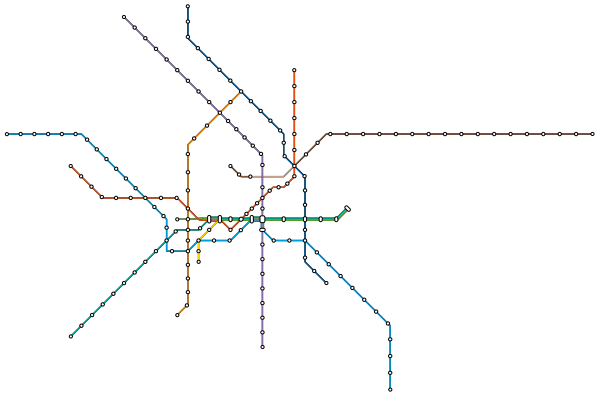

Figure 13: *Map of the Berlin subway network, octilinearized with our approach A-2+D with grid size $D = 0.75 \cdot \bar{d}$ in 764 ms.*

5. Labeling

Neither our ILP nor our approximation algorithm so far consider the optimal placement of station labels. In particular, our methods do not explicitly optimize for enough space for labels around stations. In Figure 15.4, we give an example of the Sydney map labeled a posteriori with a very simple approach which greedily places labels around stations at one of 8 octilinear directions, where no collision is induced. Nodes with higher line degrees are labeled first. A slight advantage is given to labels on certain directions to break ties, which leads to maps where labels are grouped on one side of a line. Although the result is satisfactory, three stations could not be labeled because the local map density was too high. Further work is therefore necessary.

6. Evaluation

We implemented both the ILP generation and the approximation algorithm in a tool called *octi*, which expects a line graph with geographical node positions and optional geographical line courses and outputs an octilinearized line graph. Using this tool, we evaluated six networks: the light rail networks of Stuttgart (ST) and Sydney (SY), the underground networks of Vienna (V), Berlin (B) and London (L) as well as Freiburg's tram network (F).

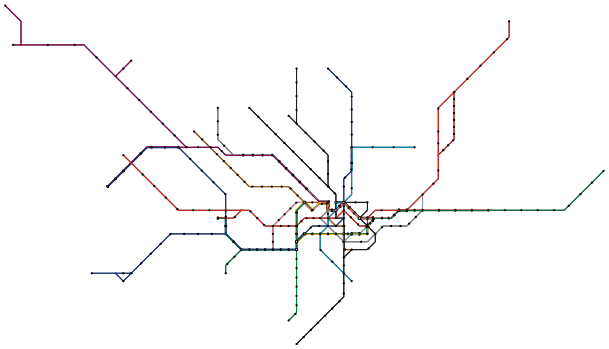

Figure 14: *Map of the London underground network, octilinearized with our approach A-2+D with grid size $D = 0.75 \cdot \bar{d}$ in 2.7 s.*

For each dataset, we evaluated our ILP approach with or without the deg-2 heuristic (LP, LP-2) and our approximation approach with or without the deg-2 heuristic (A, A-2). We also evaluated the approximation approach with the deg-2 heuristic and additional distance optimization of contracted nodes (A-2+D). Running times of all approaches with a grid size $D = 0.75 \cdot \bar{d}$ (where $\bar{d}$ is the average distance between adjacent nodes in the input graph) are given in Table 2. For LP and LP-2, we additionally evaluated the optimization times when the solver was provided with the approximate solution (LP*, LP-2*). Interestingly, optimization took longer for the Vienna network for LP-2* than for LP-2, but in general it sped up the optimization by up to a factor of 2. For LP-2 and A-2, we evaluated both the final objective function value and the approximation error produced by A-2 for $D = 0.75 \cdot \bar{d}, D = 1.0 \cdot \bar{d}$ and $D = 1.25 \cdot \bar{d}$. The results are given in Table 1. As the penalty function for the distance optimization of contracted nodes is quadratic, we cannot use it in our ILP formulation. We could therefore not evaluate the approximation error of A-2+D. However, we report the solution times for A-2+D in Table 2 and give visual results for this approach in Figures 1, 12, 13, 14, 15 and 16. For all evaluation runs, we used the following bend penalties: $c_{135} = 1$, $c_{90} = 1.5$ and $c_{45} = 2$ and restricted the possible grid node candidates for an input node v to lie within a radius of $3 \cdot D$ around v. The node move cost c_m was always set to 0.5. An additional offset cost of 0.5 was added to diagonal edges, as we found that slightly preferring vertical and horizontal edges produced results that were more esthetically pleasing. If the distance between contracted nodes was penalized (A-2+D), we always set the spring constant to $c = 10$. Tests were run on an Intel Xeon E5649 machine with 12 cores à 2.53 GHz and 96 GB RAM. ILPs were solved with Gurobi 8.1.1 using default parameters, the number of threads was set to 8.

Apart from optimizing the distances between contracted nodes, we found the local search phase to be indeed only a polishing step for all datasets. For example, Figure 12, left shows the Stuttgart map before the local search phase, the result is already satisfactory.

In general, the deg-2 heuristic had a significant impact on solution times. None of the ILPs could be optimized in under 24 hours without it (we could therefore not evaluate the approximation error for method A). With the deg-2 heuristic enabled, an optimal solution was found in under 24 hours for all datasets except London. Similarly, the deg-2 heuristic led to solution times of our approx. approach which were up to an order of magnitude faster (Table 2).

Table 1: *Final objective values of our ILP (LP-2) and our approx. approach (A-2), both with deg-2 heuristic for grid cell sizes $D = 0.75 \cdot \bar{d}, D = 1 \cdot \bar{d}$ and $D = 1.25 \cdot \bar{d}$. The approximation error is given as δ. Scores of — mean no solution could be found. If no optimal LP solution was found in under 24 hours, we give the best bound.*

	$D = 0.75 \cdot \bar{d}$			$D = 1.0 \cdot \bar{d}$			$D = 1.25 \cdot \bar{d}$		
	LP-2	A-2	δ	LP-2	A-2	δ	LP-2	A-2	δ
F	144.6	146.5	1.3%	119.1	121.6	2.1%	95.1	101.6	6.8%
V	170.5	175.1	2.7%	132.5	132.6	0.1%	109.7	110.5	0.7%
ST	383.2	399.2	4.1%	308.4	319.7	3.7%	264.3	275.9	4.4%
B	315.4	326.0	3.4%	252.3	269.4	6.8%	215.2	230.9	7.3%
SD	360.6	361.4	0.2%	291.3	311.6	7%	247.9	252.9	2%
L	≥669.2	758.3	≤14%	≥559.6	—	—	≥333.3	—	—

Table 2: *Solution times for grid cell size $D = 0.75 \cdot \bar{d}$ for LP, LP-2, A, A-2 and our approx. approach with deg-2 heuristic and optimized distance between contracted nodes (A-2+D). For LP* and LP-2*, the solver was provided with the approximate solution. Times of — mean we aborted after 24 hours (for LP-2 and LP) or that no solution was found (for A and A-2). For A, A-2 and A-2+D, 'its.' is the number of local search iterations until convergence.*

	LP	LP*	A	its.	LP-2	LP-2*	A-2	its.	**A-2+D**	its.
F	—	—	290ms	15	11m	10m	73ms	2	**110ms**	5
V	—	—	2s	33	13h	19h	171ms	4	**202ms**	6
ST	—	—	3s	37	12h	6h	510ms	9	**843ms**	12
B	—	—	3s	49	20h	12h	513ms	11	**764ms**	15
SD	—	—	3s	36	7h	6h	250ms	2	**515ms**	7
L	—	—	—	—	—	—	2.1s	12	**2.7s**	22

We also conducted a visual comparison of our methods against 3 state-of-the-art approaches from [NW11], [WC11] and [WP16]. The results can be seen in Figure 15.

All results for all datasets and all methods can be inspected online at `http://octi.cs.uni-freiburg.de`.

6.1. Grid Cost Experiments

As mentioned in Section 2.2, we may introduce arbitrary offsets to individual grid edges. Figure 16.2 gives an example of the Freiburg map rendered with a cost offset of 6 for vertical and horizontal edges. Figure 16.3 uses a cost offset of 3 for diagonal edges. Additionally, we added obstacle polygons for lakes, park areas and mountains. Grid edges contained in or intersecting such an obstacle received infinite offset costs. A particularly interesting use of offset costs is shown in Figure 16.4. The offset cost for each grid edge was set to the quadratic distance of the grid edge to the original geographical course of the input line. The result is an octilinear drawing in which edges close to the original geographical path are preferred. One practical application of this is that the resulting octilinear maps can be used as overlays over existing maps or satellite imagery. Since our method allows an arbitrary number of bends per edge, this technique still works even for large distances between stations (as is for example the case in national railway maps).

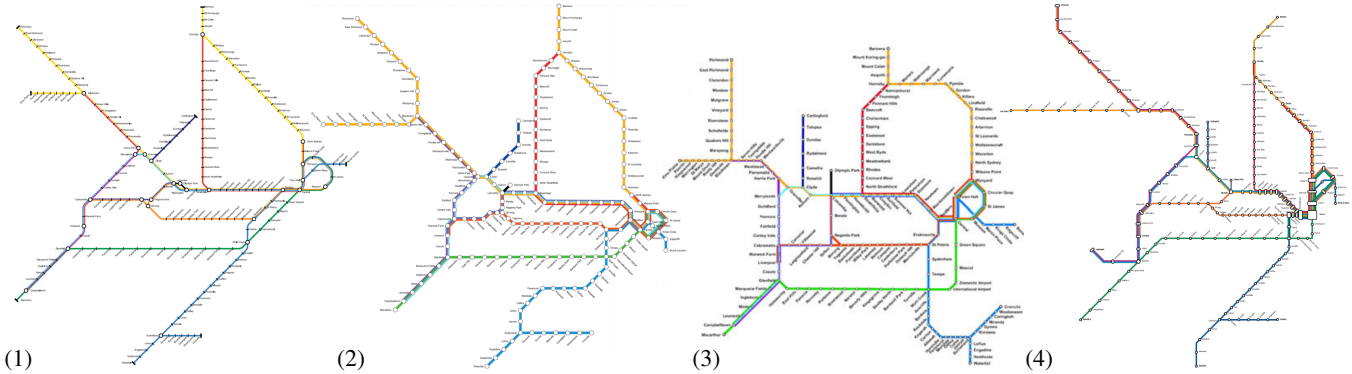

Figure 15: *Visual comparison of the Sydney CityRail network map rendered by our approach and several state-of-the-art approaches. (1) Result from [NW11], reported octilinearization time (incl. labeling optimization) was 23 minutes. (2) Result from [WC11], reported octilinearization time was 816 ms. (3) Results from [WP16], no octilinearization time was reported, but a network of similar size (Berlin) was optimized in under 150 ms. (4) Our map with simple labeling, octilinearized with A-2+D and grid size $D = 1.25 \cdot \bar{d}$ in 370 ms.*

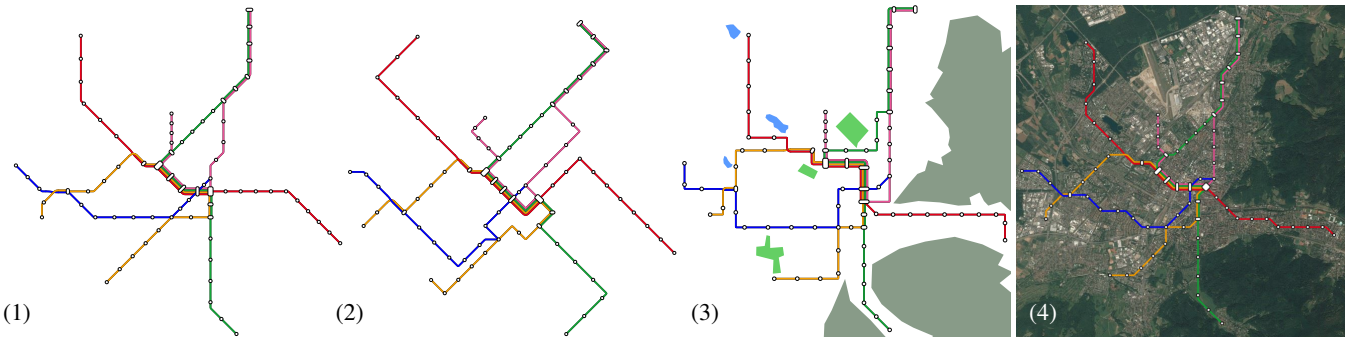

Figure 16: *Grid cost experiments. (1) Freiburg tram network map octilinearized with our approach A-2+D with grid size $D = 0.75 \cdot \bar{d}$. (2) Map with additional cost offset 6 for vertical and horizontal edges. (3) Map with additional cost offset 3 for diagonal edges and obstacles for mountains, park areas and lakes. (4) Octilinear map of the Freiburg tram network used as an overlay over a satellite image, grid edge costs reflected the distance to the original geographical line course, resulting in a map closer to reality.*

7. Conclusions and Future Work

We considered the general problem of drawing octilinear transit maps with a potentially arbitrary number of edge bends between stations. We provided an integer linear program (ILP) to solve the problem optimally. Since solving the ILP takes several hours even for simple networks, we developed a fast approximation algorithm which calculates near-optimal solutions in under three seconds for all datasets when input nodes of degree 2 are first contracted. The approximation error is always under 7.5% for datasets which could be optimized exactly. Even without contraction, solution times are under five seconds for all datasets for which a solution could be found, while none of the corresponding ILPs could be optimized in under 24 hours. We also described how a minimum distance between contracted stations may be enforced in our approach.

Our maps have objective advantages over existing maps: due to the potentially arbitrary number of bends, we can approximate a given geographical course and we can exclude forbidden areas. Throughout the paper and on `http://octi.cs.uni-freiburg.de` we provide numerous examples of the high esthetic quality of our maps. It would be interesting to conduct a more rigorous evaluation of the esthetic value, both relative to existing maps as well as among variants of our own method. This would require a comprehensive user study, which was out of scope for this paper, but which we consider a very worthwhile pursuit.

Our model so far constrains the bend penalties and the cost of a grid hop by $c_{135} \leq c_{90} \leq c_{45}$ and $c_h \geq c_{45} - c_{135}$. This is caused by the way we model the bend edges between node ports. A more flexible model could use a directed grid graph where each original grid node is outfitted with two port nodes per octilinear direction: one for arriving and one for leaving the grid node. If an arriving port is only connected via bend edges to leaving ports and vice versa, it would be impossible for a path to use two bend edges in sequence.

As mentioned in Section 5, further work on labeling is necessary. For example, it would be possible to consider the label placement during the local search phase of our approximate approach. To achieve this, the score of an optimized labeling for the drawing of the current local search iteration could be added to the objective score. Lastly, we feel that requiring adjacent nodes to have a uniform minimum distance is not always desirable. Consider Figures 12 and 13. Both drawings have long outlier edges for peripheral lines. It may be interesting to locally enlarge areas of the input line graph (for example, the city center) prior to octilinearization.

Acknowledgements

This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Project-ID 50974019 - TRR 161.

Open access funding enabled and organized by Projekt DEAL. [Correction added on 26 February 2021, after first online publication: Projekt Deal funding statement has been added.]

References

[BBS19] BAST H., BROSI P., STORANDT S.: Efficient generation of geographically accurate transit maps. *ACM Transactions on Spatial Algorithms and Systems (TSAS) 5*, 4 (2019), 25. 2

[BGKK14] BEKOS M. A., GRONEMANN M., KAUFMANN M., KRUG R.: Planar octilinear drawings with one bend per edge. In *International Symposium on Graph Drawing* (2014), Springer, pp. 331–342. 2

[BKPS07] BEKOS M. A., KAUFMANN M., POTIKA K., SYMVONIS A.: Line crossing minimization on metro maps. In *International Symposium on Graph Drawing* (2007), Springer, pp. 231–242. 2

[BNUW06] BENKERT M., NÖLLENBURG M., UNO T., WOLFF A.: Minimizing intra-edge crossings in wiring diagrams and public transportation maps. In *International Symposium on Graph Drawing* (2006), Springer, pp. 270–281. 2

[CK15] CHUZHOY J., KIM D. H.: On approximating node-disjoint paths in grids. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)* (2015), Schloss Dagstuhl - Leibniz-Zentrum für Informatik. 3

[CL19] CRAIG P., LIU Y.: A vision for pervasive information visualisation to support passenger navigation in public metro networks. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops* (2019), IEEE, pp. 202–207. 3

[CY14] CLAUDIO P., YOON S.-E.: Octilinear layouts for metro map visualization. In *2014 International Conference on Big Data and Smart Computing (BIGCOMP)* (2014), IEEE, pp. 19–21. 3

[Gar94] GARLAND K.: *Mr. Beck's Underground Map: A History*. Capital Transport Pub, 1994. 1

[HMDN04] HONG S.-H., MERRICK D., DO NASCIMENTO H. A.: The metro map layout problem. In *International Symposium on Graph Drawing* (2004), Springer, pp. 482–491. 2

[HMdN06] HONG S.-H., MERRICK D., DO NASCIMENTO H. A.: Automatic visualisation of metro maps. *Journal of Visual Languages & Computing 17*, 3 (2006), 203–224. 2

[Nes04] NESBITT K. V.: Getting to more abstract places using the metro map metaphor. In *Proceedings. Eighth International Conference on Information Visualisation, 2004. IV 2004.* (2004), IEEE, pp. 488–493. 3

[Nöl05] NÖLLENBURG M.: *Automated drawing of metro maps*. Universität Karlsruhe, Fakultät für Informatik, 2005. 2, 5

[NW05] NÖLLENBURG M., WOLFF A.: A mixed-integer program for drawing high-quality metro maps. In *International Symposium on Graph Drawing* (2005), Springer, pp. 321–333. 2

[NW11] NÖLLENBURG M., WOLFF A.: Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Trans. Vis. Comput. Graph. 17*, 5 (2011), 626–641. 2, 3, 9, 10

[SGSK01] SANDVAD E. S., GRØNBÆK K., SLOTH L., KNUDSEN J. L.: 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* (2001), ACM, pp. 326–333. 3

[SR04] STOTT J. M., RODGERS P.: Metro map layout using multicriteria optimization. In *Proceedings. Eighth International Conference on Information Visualisation, 2004. IV 2004.* (2004), IEEE, pp. 355–362. 2

[SRB*05] STOTT J. M., RODGERS P., BURKHARD R. A., MEIER M., SMIS M. T. J.: Automatic layout of project plans using a metro map metaphor. In *Ninth International Conference on Information Visualisation (IV'05)* (2005), IEEE, pp. 203–206. 3

[SRMOW10] STOTT J., RODGERS P., MARTINEZ-OVANDO J. C., WALKER S. G.: Automatic metro map layout using multicriteria optimization. *IEEE Transactions on Visualization and Computer Graphics 17*, 1 (2010), 101–114. 2

[vDL18] VAN DIJK T. C., LUTZ D.: Realtime linear cartograms and metro maps. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2018), ACM, pp. 488–491. 3

[WC11] WANG Y., CHI M.: Focus+context metro maps. *IEEE Trans. Vis. Comput. Graph. 17*, 12 (2011), 2528–2535. 3, 9, 10

[WP16] WANG Y., PENG W.: Interactive metro map editing. *IEEE Trans. Vis. Comput. Graph. 22*, 2 (2016), 1115–1126. 3, 9, 10

[WTH*13] WU H., TAKAHASHI S., HIRONO D., ARIKAWA M., LIN C., YEN H.: Spatially efficient design of annotated metro maps. *Comput. Graph. Forum 32*, 3 (2013), 261–270. 3

[WTLY12] WU H., TAKAHASHI S., LIN C., YEN H.: Travel-route-centered metro map layout and annotation. *Comput. Graph. Forum 31*, 3 (2012), 925–934. 3