23

Graph Drawing and Cartography

Alexander Wolff
University of Würzburg

23.1 Introduction

Graph drawing and cartography come together when networks whose elements have geographic locations, that is, geometric networks, have to be visualized. Examples of such networks are street, subway, river, or cable networks. Often it helps to visualize the underlying network for analyzing certain network parameters. For example, traffic on a road network can be visualized by drawing each road as a rectangle whose width is proportional to the amount of traffic going through that road.

One of the main problems in map production is a process called *generalization*. Given cartographic data that has been collected at large scale, this data must be simplified in order to produce maps at small scale. In order to obtain readable maps, detail must be reduced and spacing must be enlarged. Traditionally this has been done manually by cartographers, but increasingly semi-automated and even automated methods are in use, particularly in conjunction with geographic information systems (GIS) [Ass96]. Cartographers have identified a number of generalization *operators* such as displacement, size exaggeration, size reduction, and deletion in order to cope with the many constraints that govern the generalization process. The main difficulty in automating generalization is the interdependency of these operators.

Saalfeld [Saa95], both geodetic and computer scientist, pointed out (in one of the first editions of the graph drawing conference) that map generalization can be seen as a graph drawing problem—if one accepts that a cartographic map is but a straight-line drawing of a graph in the plane. Then the process of redrawing a map at smaller scale can be interpreted as a sequence of modifications of both the graph and its drawing. Graph elements must be

contracted or removed, and the drawing must be modified to reflect the graph reductions. Moreover, the drawing must be modified in that "old" graph elements must be moved, for example, due to distance constraints. Saalfeld is an early advocate of continuous generalization: his ultimate goal is a map with a slider bar for scale. (For a rather restricted continuous generalization problem, see Section 23.2.2.) Saalfeld points to the key issue: address the "big picture"—take feature interaction into account. He challenges the graph drawing community to "design and implement an efficient and effective automated map generalization system for the line network of a digital map."

Note that general graph-drawing algorithms cannot be used ad hoc for drawing geometric networks since they do not respect the geometry that comes with the vertices and edges. A good drawing of a geometric network must reflect geometry since a user typically has an intuitive notion of the underlying geometry, in other words, a *mental map* [ELMS91]. For example, the user of a metro system expects stations in the north to appear on the top of maps that depict the metro system. Thus the "art" of drawing geometric networks is to find a good compromise between distorting geometry and maximizing aesthetics. This will be the leitmotif of this chapter, which also explains why we will not touch *point-set embeddability* problems. Recall that, in a point-set embeddability problem, one is given not just a graph but also a set of points in the plane (or on a line) and the aim is to find a mapping between vertices and points such that the edges can be drawn under some drawing convention. For example, Gritzmann et al. [GMPP91] showed that any n-vertex outerplanar graph can be embedded on any set of n points in the plane (in general position) such that edges are represented by straight-line segments connecting the respective points and no two edge representations cross. In the type of problem we are interested here, in contrast, the mapping between vertices and points is part of the input and, in many cases, we may move the points to some extent.

We focus on node-link representations of geometric networks, that is, we insist on representing vertices by points or small icons such as disks or squares and edges by some linear features (Jordan curves, in general). This excludes contact or intersection representations (such as rectangular cartograms) where edges are represented implicitly; by the contact or intersection behavior of the "large" geometric objects that represent the vertices. For such representations, see Chapter 10 on rectangular drawings and Chapter 11 on simultaneous drawings.

Note, however, that additional requirements come into play in a geographic context. For example, the relative position or the relative sizes of the geometric objects representing the vertices are often prescribed by the user. As an example for this additional difficulty, take Koebe's beautiful theorem [Koe36] that says that every planar graph can be represented as a coin graph, that is, as a set of interior-disjoint disks, two of which touch if and only if the corresponding vertices are adjacent in the given graph. If one now introduces geometric constraints by prescribing a set of "anchor" points and a bijection between points and vertices (and, hence, disks), and by insisting that each disk contains its point, then realizability of a given planar graph as a *cover contact graph* becomes NP-hard [AdCC$^+$12].

In this chapter, we give an overview of the main types of geometric networks that are being visualized in an automated fashion, using node-link diagrams. For each network type, we consider the application-dependent aesthetic constraints. We group the network types according to the graph class to which they belong: paths (simplified, schematized and generalized in Section 23.2), matchings (used in boundary labeling in Section 23.3), trees (as in flow maps; see Section 23.4), (near-) plane graphs (such as street or metro maps; see Section 23.5), and other graphs (such as timetable graphs, the Internet multicast backbone, or social networks; see Section 23.6). Note that we use the term *plane* to stress that the

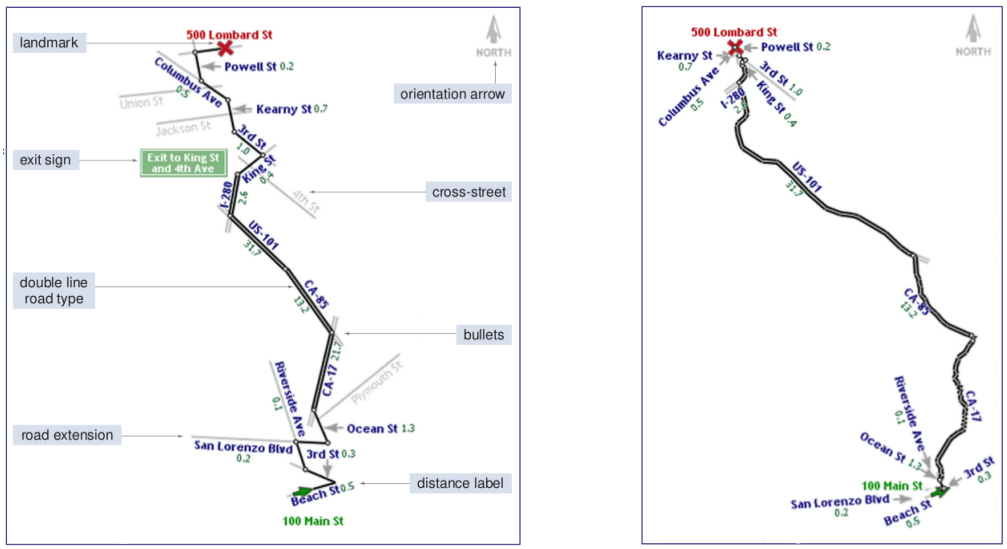
graphs are given with a planar embedding. For example, a self-intersecting polygonal line can be considered a path and hence a planar graph, but it is not a plane graph.

23.2 Paths

When drawing paths nicely, the main problem is data reduction: which points can be dropped while maintaining the important features of a polygonal line? Due to its many applications, polygonal line simplification has been identified as an important problem both in cartography and in computational geometry. Since Douglas and Peucker [DP73] presented a simple and frequently used algorithm, cartographers have devised solutions of higher cartographic quality [VW93, Saa99, LL99], while geometers have given a more efficient implementation of the Douglas-Peucker algorithm [HS94] and have designed new algorithms for specialized error criteria [AV00, BCC+06] or for a restricted number of orientations [Ney99, MG07, DHM08]. Still, finding a near-linear time solution for polygonal line simplification is listed as problem 24 in the *Open Problems Project* [MO01].

23.2.1 Simplifying and Schematizing Polygonal Paths

The path-drawing problem that Agrawala and Stolte [AS01] considered has more of a graph-drawing flavour. Their *route maps* help car drivers to get from A to B. While most route planners draw routes using a fixed-scale map as background, Agrawala and Stolte suggested to draw edges of the path (that is, roads between turns) as straight-line segments which are usually *not* to scale. Instead, their system LineDrive exaggerates the length of short road segments in order to label them properly with street name and real length, see Figure 23.1.



(a) map generated with LineDrives (b) constant-scale map (for comparison)

Figure 23.1 LineDrives generates driving directions. Sketches taken from [AS01].

In the resulting drawings, angles at turns are mostly kept, except at very sharp turns. Roads that are close to being vertical or horizontal are usually made vertical or horizonatal, respectively. The LineDrive system, which is based on simulated annealing, was publicly available for some period of time and received very positive response from most users.

Later on, the path-drawing problem of Agrawala and Stolte [AS01] inspired research in the graph-drawing community. Brandes and Pampel [BP13] showed, by reduction from MONOTONE3SAT, that the rectilinear (orthogonal) case is NP-hard; more precisely, it is NP-hard to decide whether a given polygonal path has a simplification that consists exclusively of horizontal and vertical segments and preserves the *orthogonal order* (that is, horizontal and vertical order) of the vertices along the path. The ordering constraint is meant to help the user maintain his mental map.

On the positive side, Delling et al. [DGNP10] showed that, given an polygonal path and a set $\mathcal{C}$ of directions, they can efficiently compute a simplification such that (a) all edge directions are in $\mathcal{C}$ and (b) the orthogonal order of the vertices is preserved—if the input path is x-monotone. Their algorithm finds a simplification of minimum cost, which they define to be the sum over the costs of all edges. The cost of an edge, in turn, is defined to be the angle between the edge in the output and the direction in $\mathcal{C}$ that is closest to the direction of the edge in the input. The algorithm is based on a clever characterization of optimum solutions and on dynamic programming. When the number of directions, $|\mathcal{C}|$, is considered a constant, their algorithm runs in $O(n^2)$ time and uses $O(n)$ space, where n is the number of vertices of the input path. Using a linear-programming formulation (of linear size), Delling et al. can even find, among all simplifications with a fixed direction for each edge, one of minimum total length. In addition, they present a heuristic for dealing with the non-monotone case.

A natural generalization of the rectilinear case considered by Brandes and Pampel [BP13] is the *d-regular* case, where the set of directions consists of multiples of $90°/d$. Delling et al. [DGNP10] established their positive result for x-monotone paths for *any* set of directions (actually, any set containing the multiples of 90°); in particular, their result holds for the d-regular case for any $d \geq 1$. Gemsa et al. [GNPR11] generalized the negative result of Brandes and Pampel from $d = 1$ to any $d \geq 1$, using a different reduction (from MONOTONEPLANAR3SAT). On the other hand, they presented a mixed-integer linear programming (MIP) formulation for d-regular path simplification (for any $d \geq 1$) and evaluated it on real-world instances (quickest routes between random destinations in the German road network). They concluded that the MIP runs fast enough if the road geometry is preprocessed with a conventional path simplification method (such as Douglas-Peucker [DP73]). They suggested that $d = 3$ is a good compromise between accuracy and abstraction.

23.2.2 Continuous Generalization for Polygonal Lines

A path simplification problem of a rather different flavor was investigated by Merrick et al. [MNWB08]. They assumed that both a detailed and a less detailed drawing of a path are given; they are interested in how to get from one to the other in a continuous fashion. In computer graphics, such a transition is called a *morph*. From a cartographic point of view, their problem is a *continuous generalization* problem: given two linear objects (such as streets or rivers) on maps of different scale, deform one representation continuously into the other such that intermediate representations are valid generalizations for their scale.

The problem naturally decomposes into two subproblems: first, find a correspondence between parts of one path and parts of the other path; second, define a movement that moves the parts of one path onto the corresponding parts of the other path. Merrick et al. focused on the first subproblem and solve the second subproblem by simply moving the

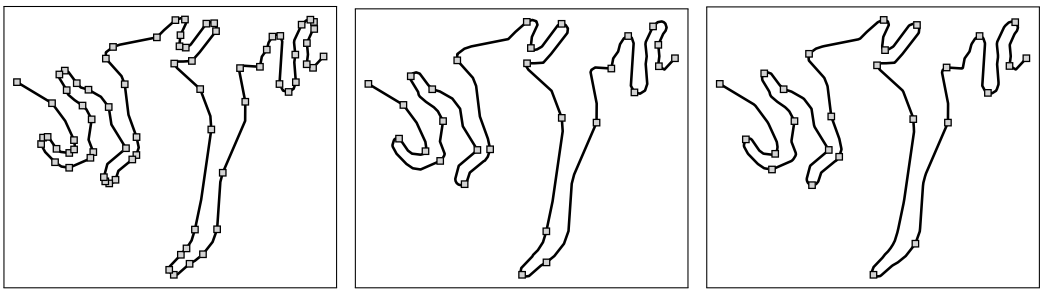
vertices of one path on *linear* trajectories to their counterparts. The first subproblem can again be subdivided into two tasks: first, find *characteristic* points on both paths; second, find a good correspondence between the subpaths defined by consecutive corresponding points. The idea behind the characteristic points is not only data reduction, but detecting such points and treating them with special care makes it more probable that the viewers of the resulting morph keep their mental map during the animation.

For the first task, Merrick et al. incrementally fitted cubic Bézier curves to a growing part of the given polygonal path. When the distance between the current subpath and the curve surpasses a pre-specified error bound $\varepsilon > 0$, Merrick et al. viewed the point added last as a characteristic point, and repeat the fitting process with the subpath starting at that point. The distance between subpath and curve is approximated by sampling both with a relatively large number of points, measuring the distances only between corresponding points and taking the maximum over these point-to-point distances.

Figure 23.2 shows a mountain road in the French Alps and the characteristic points that were detected using the Bézier-fitting method of Merrick et al. for two different values of the error bound ε; 1 and 25. Subfigure (c) shows the same road with manually selected characteristic points. The automatically detected set of characteristic points for $\varepsilon = 25$ and the manually detected set are quite similar.

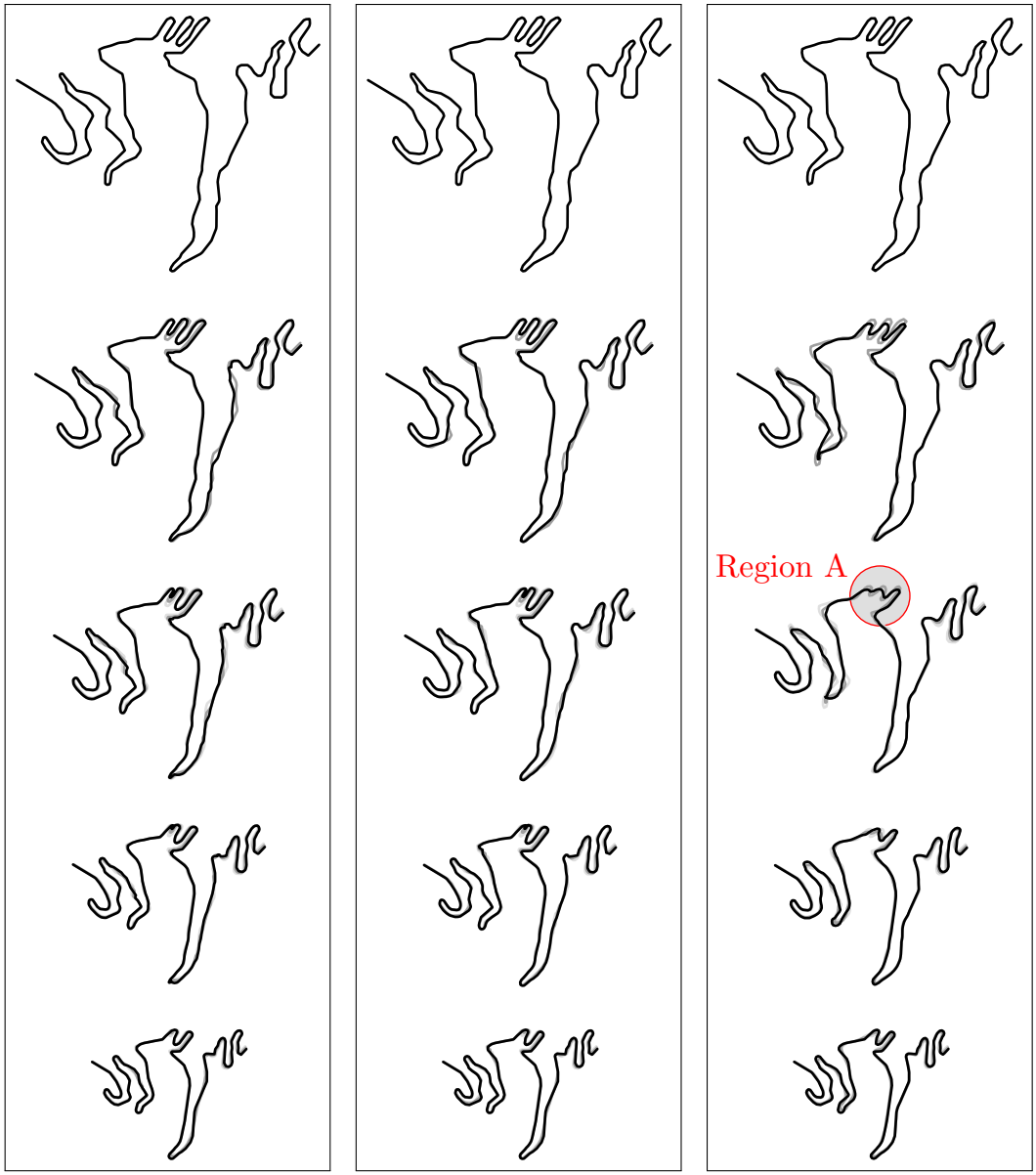
For the second task, Merrick et al. presented a dynamic program that computes a correspondence between the two paths, in $O(nm)$ time, where n and m are the numbers of subpaths of the first and second path, respectively. The correspondence is optimal with respect to the distance function defined by the user; the authors make a number of suggestions for such functions.

Figure 23.3 shows snapshots morphs between two representations of the road in Figure 23.2. The more detailed representation is from a BD(R) Carto map at scale 1:50,000; the less detailed, generalized representation of the same road at scale 1:100,000 is from an IGN Carto2001 TOP100 map. The example road was chosen because it is represented by three serpentines on the detailed scale but only by two serpentines in the less detailed scale. Each morph is based on a different choice of characteristic points; *linear interpolation* (variant (c)) is a simple ad-hoc method that matches each point on one polyline to the point at the same relative distance from the start on the other polyline. The middle snapshot produced by this method shows its weakness, especially in the part of the polyline labeled "Region A." While the two other morphs (in subfigures (a) and (b)) keep the "amplitude" of the serpentines while merging the first two, linear interpolation first reduces the amplitude and then increases it again.



(a) $\varepsilon = 1$ (72 points) (b) $\varepsilon = 25$ (28 points) (c) manual (26 points)

Figure 23.2 Selection of characteristic points according to Merrick et al. [MNWB08]. The polyline is a mountain road from the French Alps; it consists of 155 vertices.



(a) $\varepsilon = 25$ (b) manual (c) linear interpolation

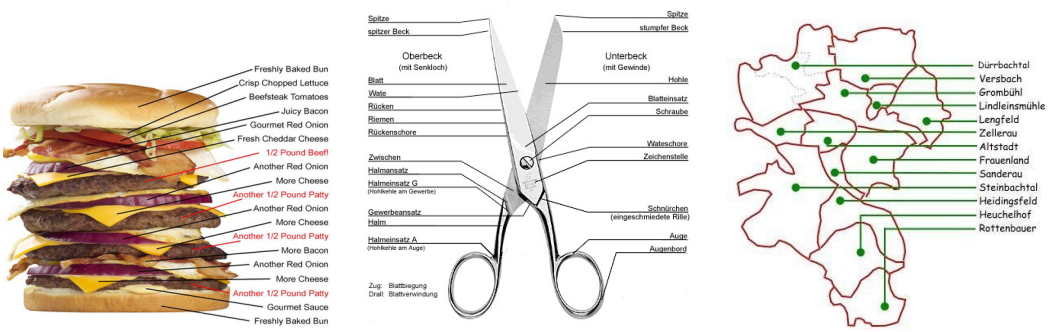
Figure 23.3 Morphs generated by Merrick et al. [MNWB08] depending on the method for selecting characteristic points. In each snapshot, previous frames are shown in increasingly light shades of gray to assist perception of the animation.

For this road, which has 190 and 155 vertices on the 1:50K and the 1:100K maps, respectively, it took less than 0.01 seconds to compute the characteristic points, 1.39 seconds to compute the optimal correspondence for $\varepsilon = 1$, and 0.59 seconds for $\varepsilon = 25$. The road is part of a map sheet with 382 roads consisting of 13345 and 10869 vertices on the two maps, which were reduced (for $\varepsilon = 25$) to 2742 and 2387 characteristic points, respectively. For the whole 1:50K map sheet, this reduction took 0.69 seconds; computing the correspondence then took 13.17 seconds. The experiments were performed on an AMD Athlon XP 2600+ PC with 1.5 GB main memory running under SuSE Linux 10.1. These running times are acceptable since tasks can be considered pre-processing. Only the resulting simple linear morph needs to be executed in real time. In order to solve the continuous generalization problem for complete street or river networks across large scale intervals, the line-simplification algorithm sketched here must be combined with a topology-simplification algorithm, which yet has to be devised.

23.3 Matchings

Matchings do not appear to be an exciting graph class for graph drawing, but they have an interesting application that brings cartography and graph drawing together: so-called *boundary labeling*. In boundary labeling, one is given a set of point sites on a rectangular map and, for each site, a rectangular label that contains, for example, textual information about the site. Other than in normal point labeling, labels are not placed next to the site they label, either because the point set is too dense with respect to the label sizes or because the map background must not be covered by the labels. Instead, labels are placed outside the map such that they touch the map boundary with one side. In order to visualize the mapping between sites and labels, each site is connected to its label with a polygonal line, the so-called *leader*. For three real-world examples with different leader types, see Figure 23.4.

The boundary labeling problem was introduced by Bekos et al. [BKSW07]. For a given rectangle R (for example, a cartographic map), a set P of point sites in R and, for each site s in S, a rectangular label L_s, Bekos et al. define a *feasible leader-label placement* to be a placement of the labels and a drawing of the leaders that fulfills the following requirements:



(a) parts of a hamburger (b) parts of a pair of scissors (c) districts of Würzburg

Figure 23.4 Examples of boundary labeling.

(B1) Labels are disjoint.

(B2) Labels lie outside (the interior of) R such that, for each label, one of its edges is contained in one of the edges of R.

(B3) Each point is connected to its unique label by a leader.

(B4) Leaders may not intersect other leaders, points or labels.

(B5) The point where a leader touches a label is called *port*; ports may be *fixed* (for example, to the centers of the label edges) or *sliding* (that is, arbitrary).

(B6) Labels either have fixed positions or can slide along an edge of R.

(B7) Labels can be attached to one, two or all four edges of R. The resulting problems are called one-side, two-side and four-side leader-label placements.

In addition to feasible leader-label placements, mainly the following objective functions have been considered:

(O1) small ink consumption (minimize total leader length),

(O2) straightness (minimize number of bends).

These are typical graph drawing objectives; they help to keep the visual complexity of the resulting drawing low.

Several types of leaders have been considered; until now all of them are polygonal with up to two bends. Generally, a leader type is denoted by a word from the set $\{s, \{p, o, d\}^*\}$; the letters refer to the direction of the line segments that form the leader, starting at the point to be labeled and ending at the port that lies on some edge e of R. The leader type s refers to straight-line leaders; their direction is arbitrary. Leader segments labeled p are *parallel* with e, segments labeled o are *orthogonal* to e, and segments labeled d are *diagonal*, that is, they form an angle of $45°$ or $-45°$ degrees with e.

Two-sided boundary labeling with labels of *non-uniform* height is NP-hard; the reduction from Partition is obvious. Therefore, most references focus on uniform labels, that is, all labels are unit-height rectangles. In Table 23.1, we summarize the running times of the best known algorithms (in big-Oh-Notation) for various versions of the boundary labeling problem.

The following variants and extensions of the boundary labeling problem have been considered:

- boundary labeling with *octilinear* leaders, that is, leaders whose segments are horizontal, vertical, or diagonal at $\pm 45°$ [BKNS10],
- multi-criteria boundary labeling [BHKN09],
- boundary labeling for area features [BKPS10],
- boundary labeling under rotations [NPS10],
- text annotation [LWY09],
- multi-stack boundary labeling [BKPS06],
- many-to-one boundary labeling [Lin10, LKY08],
- one-and-a-half-side boundary labeling [LPT$^+$11],
- boundary labeling combined with traditional map labeling [BKPS11], and
- boundary labeling for panorama images [GHN11].

In order to give the reader at least a flavor of this variety of results, we review some of the early algorithms for type-s and type-po leaders. In the case of one- and two-side problems, we attach labels to the right edge and both vertical edges of R, respectively.

leader type	# map edges with labels	feasible solution	bend-minimal solution	length-minimal solution		
				fixed ports	sliding ports	reference
s	1	$n \log n$	N/A	$n^{2+\varepsilon}$	n^3	[BKSW07]
s	4	$n \log n$	N/A	$n^{2+\varepsilon}$	n^3	[BKSW07]
po	1		n^3	$n \log n$	$n \log n$	[BHKN09]
po	2			n^2	n^2	[BKSW07]
po	2		n^8			[BHKN09]
opo	1	$[n \log n]$	$[n^2]$	$n \log n$	$[n^2]$	[BKSW07]
opo	2		open	n^2 $[nH^2]^\star$	n^2	[BKSW07]
opo	4	$n \log n$	open	$n^2 \log^3 n$	n^3	[BKSW07]
do	1		n^5	n^2	n^2	[BHKN09]
do	2		n^{14}			[BHKN09]
$\{do, pd\}$	1		open	n^3 $[—]^\star$	n^3	[BKNS10]
$\{od, pd\}$	1	$n \log n$	open	n^3	n^3	[BKNS10]
$\{do, pd\}$	2		open	n^3	n^3	[BKNS10]
$\{od, pd\}$	2	$n \log n$	open	n^3	n^3	[BKNS10]
$\{od, pd\}$	4	$n \log n$	open	n^3	n^3	[BKNS10]

Table 23.1 Running times of the best known algorithms (in big-Oh-Notation) for various versions of boundary labeling, where ε is an arbitrarily small positive constant and n is the number of sites. The time bounds in square parentheses refer to the case of non-uniform labels. The problems marked by * are NP-hard. The pseudo-polynomial algorithm for 2-sided *opo*-type leader-label placement assumes that label heights and the height H of the bounding rectangle are integers. N/A stands for non-applicable. Entries in column "Feasible solution" are filled only if there is a feasible solution that is asymptotically faster than a bend- or length-optimal solution.

23.3.1 Boundary Labeling with Type-s Leaders

In the case of fixed ports and fixed labels, a type-s label-leader placement or total length L corresponds to a Euclidean perfect bipartite matching of cost L. For the case of sliding ports (and fixed labels), the problem can also be reduced to a matching problem, albeit at a somewhat higher computational cost.

Theorem 23.1 *[[BKSW07]] Given a set S of n point sites, a one-side type-s leader-label placement of minimum total leader length for fixed labels can be computed in $O(n^{2+\varepsilon})$ time for any $\varepsilon > 0$ in the case of fixed ports and in $O(n^3)$ time in the case of sliding ports.*

Proof: In the case of fixed ports, we have a set P of n ports. Then a Euclidean minimum-cost perfect bipartite matching in the set $S \cup P$ yields a feasible leader-label placement of minimum total leader length. Feasibility follows from two properties of the Euclidean plane; the triangle inequality and the fact that the distances from the endpoints of a line segment to a point on the segment add up to the length of the segment. Indeed, suppose that two leaders would intersect then swapping the matching locally would decrease its cost; see Figure 23.5. (For the same reason, any solution to the Euclidean traveling salesperson

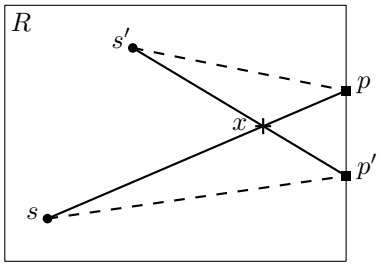


Figure 23.5 A minimum-length Euclidean matching is plane.

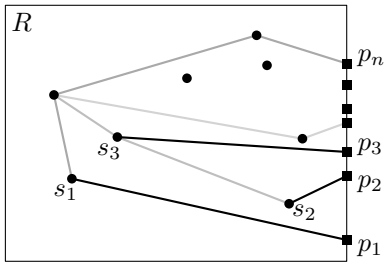


Figure 23.6 Feasible type-s leader layout via dynamic convex-hull.

problem forms a simple polygon unless all points lie on a line.) A Euclidean minimum-cost perfect bipartite matching can be computed $O(n^{2+\varepsilon})$ time for any $\varepsilon > 0$ [AES99].

For the case of sliding ports, the time complexity increases since we now need a *general* minimum-cost perfect bipartite matching in the complete bipartite graph on the set S of n points and the set L of the n label positions; the weight of an edge $(s, \ell) \in S \times L$ is the Euclidean distance of s to its closest point on ℓ. Since we assume that labels are attached to the right edge e of R, the point on ℓ closest to s is either the top or bottom point of ℓ or the orthogonal projection of s on e. A general minimum-cost perfect bipartite matching can be computed in $O(n^3)$ time [Law76]. □

If we content ourselves with a *feasible* leader-label placement, we can, in the case of fixed labels with fixed ports, speed up the computation.

Theorem 23.2 *[[BKSW07]] Given a set S of n point sites, a feasible one-side type-s leader-label placement for fixed labels with fixed ports can be computed in $O(n \log n)$ time.*

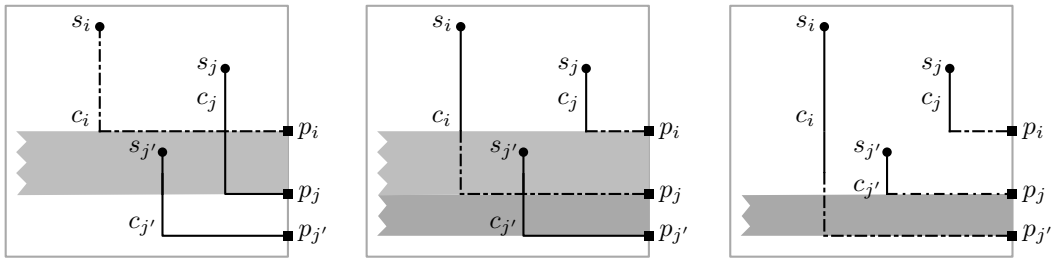
Proof: We assume that the set of ports, $P = \{p_1, \ldots, p_n\}$, is sorted according to increasing y-coordinate. Let H be the convex hull of the set $S \cup P$. Consider the edge of H that connects the bottommost point p_1 in P to a site. Call this site s_1 and make the line segment s_1p_1 a leader; see Figure 23.6. Remove s_1 from S and p_1 from P. Repeat until each site is matched to a port. Since no two ports have the same y-coordinate, in each step, the convex hull of the diminished set $S \cup P$ is disjoint from the line segment connecting the site and the port that were removed last. Hence, the resulting leader-label placement is feasible.

To make our algorithm run in $O(n \log n)$ time, we just need a semi-dynamic convex-hull data structure that preprocesses a set of n points in $O(n \log n)$ time to allow for neighbor queries and point deletions in $O(\log n)$ time. Hershberger and Suri [HS92] provided such a data structure. □

23.3.2 Boundary Labeling with Type-*po* Leaders

We start with the simplest possible variant of the problem; the algorithm for this variant is illustrated in Figure 23.7. The idea behind the algorithm will turn out to be useful for two generalizations.

Theorem 23.3 *[[BKSW07]] Given a set S of n point sites, a feasible one-side type-po leader-label placement for fixed labels with fixed ports can be computed in $O(n^2)$ time.*



(a) after inserting c_i (b) after rerouting c_i and c_j (c) after rerouting c_i and $c_{j'}$

Figure 23.7 Rerouting type-*po* leaders.

Proof: We first sort sites and ports such that $s_1, \ldots, s_n$ and $p_1, \ldots, p_n$ are indexed in order of non-decreasing y-coordinates. For $i = 1, \ldots, n$, we connect s_i to p_i by a *po*-leader c_i that consists of a (possibly zero-length) vertical line segment incident to s_i and a horizontal line segment incident to p_i. We assume that the previously placed leaders $c_1, \ldots, c_{i-1}$ are pairwise disjoint, and we show that we can add c_i such that this assumption continues to hold.

In the following, we treat the case that s_i lies above p_i; the other case can be analyzed analogously. If c_i does not intersect any of the (pairwise disjoint) leaders $c_1, \ldots, c_{i-1}$, we are done. Otherwise, let s_j be the rightmost site with $j < i$ whose leader intersects c_i; see Figure 23.7(a). We reroute the leaders such that s_j is connected to p_i and s_i to p_j; see Figure 23.7(b).

After the rerouting, the new leader c_j does not intersect any other leader since (i) its vertical segment is shorter than before and (ii) its horizontal segment used to belong to c_i, which—due to the choice of s_j—did not intersect any other leader to the right of s_j. Hence, in this process, we remove the intersections of other leaders with the horizontal segment of c_i one by one, even if new intersections occur, as in the step from Figure 23.7(a) to Figure 23.7(b).

It remains to observe that the growing vertical segment of c_i never intersects other leaders. This is true since, initially, c_i goes to the top-most port p_i and, after each rerouting operation, c_i is prolonged by a vertical sub-segment that used to "belong" to a leader to the right of c_i; the sub-segments move within the gray horizontal strips in Figure 23.7. Thus, if a leader was to intersect the new vertical sub-segment of c_i, it would have earlier intersected one of the other leaders, contradicting our above assumption. □

As it turns out, the feasible leader layout that the algorithm in the proof of Theorem 23.6 computes is already length-minimal.

Theorem 23.4 *[[BKSW07]] Given a set S of n point sites, a minimum-length one-side type-po leader-label placement for fixed labels with fixed ports can be computed in $O(n^2)$ time.*

Proof: Consider the site–port correspondence that we used in proof of Theorem 23.6: going through sites and ports from bottom to top, we connected the i-th site s_i to the i-th port p_i. We claim that the type-*po* leader layout induced by this correspondence has minimum total length among all type-*po* leader layouts (including the layouts with crossings). Combining this claim with the simple observation that rerouting does not change the total length of the leaders (see Figure 23.7), yields the theorem.

In order to prove the claim, we observe that, in all type-*po* leader-label placements, the total length of the horizontal leader segments is the same. We convert our type-*po* instance to a type-*s* instance by moving the sites to the right so that they all lie on a vertical line infinitesimally close to the right side of the boundary rectangle R. Then the vertical segments of a given type-*po* leader layout become (nearly) type-*s* leaders. Note that the above site–port correspondence is the only one that induces a plane type-*s* leader layout. Every other correspondence induces a layout with at least one pair of crossing leaders. If we untangle such a pair, the total leader length does not increase. (The only case where it remains the "same" is in the degenerate case that one of the two leaders is horizontal.) We used basically the same observation in the proof of Theorem 23.1. □

The same result holds if labels are attached to two (opposite) sides of the bounding rectangle R.

Theorem 23.5 *[[BKSW07]] Given a set S of n point sites, a minimum-length two-side type-po leader-label placement for fixed labels with fixed ports can be computed in $O(n^2)$ time.*

Proof: As in the proofs of Theorems 23.3 and 23.4, we first compute a minimum-length layout without caring about crossings. For the one-side case, this was trivial; for the two-side case, we employ a simple dynamic program. Specifically, we use a two-dimensional table; table entry (l, r) contains the minimum total leader length for the $l + r$ lowest sites under the condition that l are connected to labels at the left side of R and the remaining r to labels at the right side. Since each entry in the table can be filled in constant time, the dynamic program runs in $O(n^2)$ total time.

Again, as in the proofs of the two preceding theorems, we then apply our rerouting scheme in order to remove all crossings. Recall that this does not change the total leader length. It remains to observe that leaders going to different sides of R never cross in this process; if they did cross, rerouting them would decrease the total leader length. This, in turn, would contradict the minimality of the total leader length of the original layout. □

For the one-sided case, Benkert et al. [BHKN09] have observed that a length-minimal leader layout has a structure that can be exploited in order to speed-up its computation. The rectangular map R can be partitioned in horizontal strips such that all sites within a strip have horizontal leaders, have upward-going leaders (as in Figure 23.7(c)), or have downward-going leaders. Strips of upward- or downward-going leaders are always separated by strips with horizontal leaders, which can be detected easily. (In the case of fixed ports, "horizontal" means here that, as in the case of sliding ports, the site lies in the vertical range of the label.) Benkert et al. determine these strips in a first pass through the instance. Then, in a second pass, they determine the leader layout for the sites within a strip using a sweep-line algorithm. In total, their algorithm takes $O(n \log n)$ time. They show that this running time is worst-case optimal; sorting reduces to length-minimal leader layout. All in all, Benkert et al. have the following result.

Theorem 23.6 *[[BHKN09]] Given a set S of n point sites, a minimum-length one-side type-po leader-label placement for fixed labels with fixed or sliding ports can be computed in $\Theta(n \log n)$ time.*

23.4 Trees

In economy and social sciences, a common problem is to visualize the flow of goods or people from or into a specific destination. It makes sense to require that the flow between two nodes of the underlying network is depicted by curves whose width is proportional to the amount of flow. Usually these curves are drawn on the background of a regular map. For visualization purposes, the network is drawn as a tree—although, in general, the actual flow network is a rooted directed acyclic graph. The drawing of such a network is called a *flow map*.

Henry Drury Harness [Har38] is being cited [Rob55, FD01] for having created the first flow maps; in an atlas accompanying a report of the Railway Commissioners concerning population and movement of goods in Ireland in 1837. A few years later, Charles Joseph Minard, a French civil engineer, made flow maps mostly on economic topics, depicting, for example, the amount of wine export from France, but also, in 1869, the location and size of Napoleon's army during its 1812/13 Russian campaign; see Figure 23.8. Tufte [Tuf01, p. 40] says that this map "may well be the best statistical graphic ever drawn."

Drawing flow maps automatically was first studied by Tobler [Tob87] who used straight-line arrows of appropriate width. The restriction to straight-line edges causes a lot of visual clutter; see Figure 23.9(a).

Nearly twenty years later, Phan et al. [PXY+05] set out to improve on Tobler's result by taking advantage of clustering and curved edges. Given the positions of the network nodes, they first compute an agglomerative hierarchical clustering—independent of the position of the root. The binary tree that corresponds to the clustering captures the spacial distribution of the input. Then, they transform this unrooted binary tree into a tree rooted at the given root node. In this process, the root can get several children. The layout of the flow tree follows this tree recursively. A tree edge connecting a node to its child is routed from the position of the node to the closest corner of the bounding box of the cluster that corresponds to the child. The routing detours boxes containing sibling clusters. To make the final layout of the flow map more aesthetically pleasing, the polygonal paths that represent the edges are drawn as *Catmull–Rom splines*, that is, as special cubic curves that go through the given points. For the resulting layout, see Figure 23.9(b). Under the (rather strong)

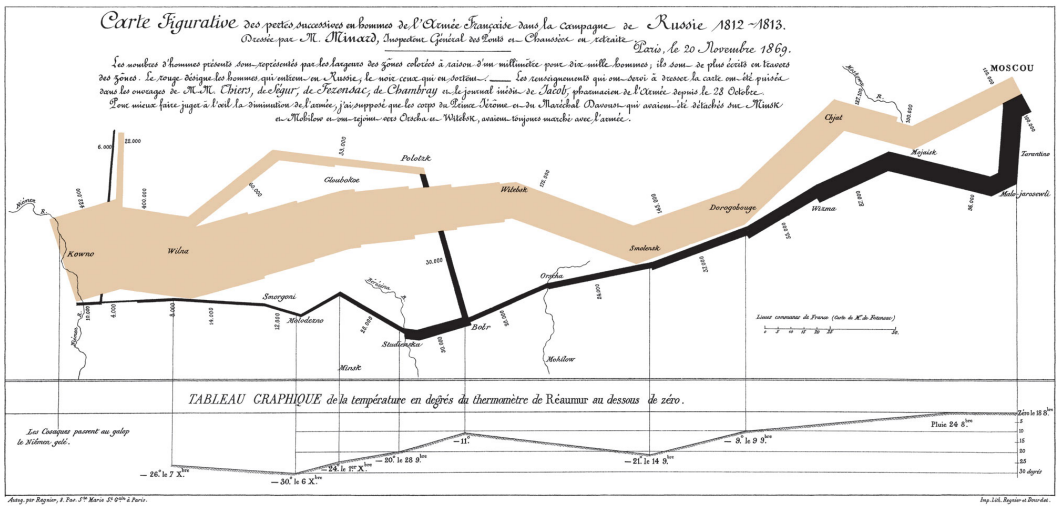
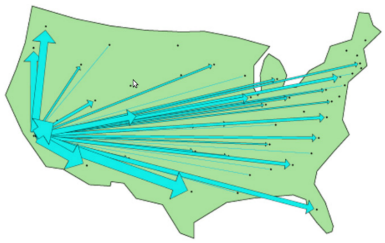
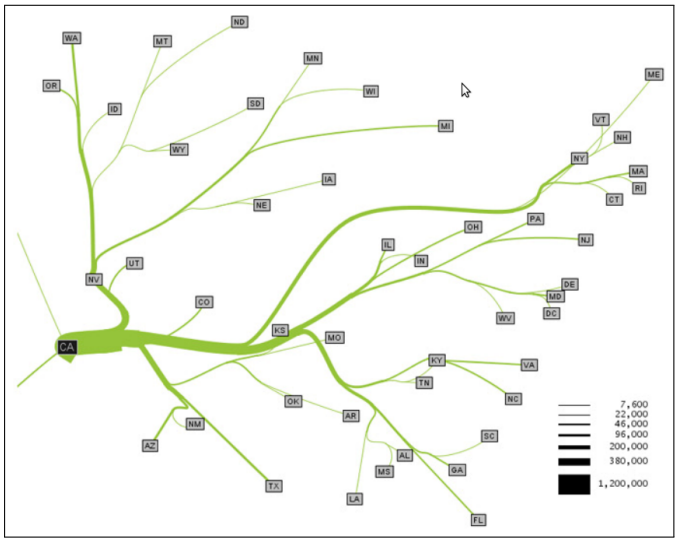


Figure 23.8 Minard's map of Napoleon's Russian campaign of 1812/13 [Min69].

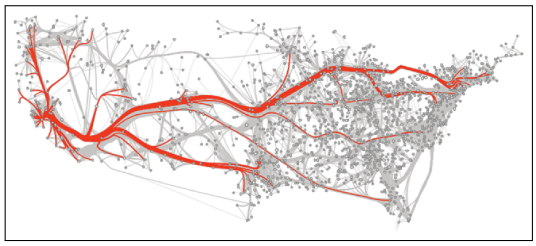


(a) Tobler [Tob87]

(c) Verbeek et al. [VBS11]



(b) Phan et al. [PXY+05]



(d) Cui et al. [CZQ+08]

(e) Holten and van Wijk [HVW09]

Figure 23.9 Flow maps showing migration leaving California in the years 1995–2000.

assumption that the boxes of child clusters are pairwise disjoint, the (polyline) tree layout is crossing-free. The complete (non-optimized) algorithm runs in quadratic worst-case time; the authors report that the examples they computed took their Java implementation a few seconds on a 1.4-GHz laptop.

Recently, Verbeek et al. [VBS11] presented a method for drawing flow maps that is based on so-called (approximate) *spiral trees*. Given a set of points (one being labeled as root) and an angle, a spiral tree is a directed angle-restricted Steiner tree of minimum length. A directed angle-restricted Steiner tree for an angle α is a tree where each edge is drawn as a curve with the property that, in every point p on the curve, the angle between the vector from p to the root and the tangent in p (pointing backward) is bounded by α; see Figure 23.10(a). The same set of authors [BSV11] showed that it is NP-hard to compute spiral trees but that 2-approximations (in terms of length) can be computed, even in the presence of obstacles, in $O(n \log n)$ time. Edges of (approximate) spiral trees are logarithmic spirals.

Starting from such an approximate spiral tree for the given point set (with all leaves being obstacles; see Figure 23.10(b)) and a user-chosen value of α (roughly in the range between 15° and 35°), Verbeek et al. compute a tree layout with edges of prescribed thickness by subdividing the original edges (see Figure 23.10(c)) and then improving a set of aesthetic

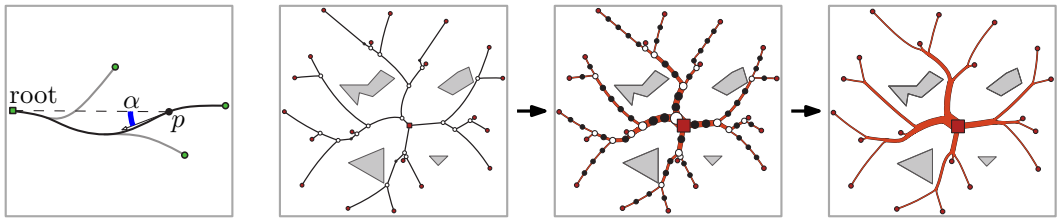


Figure 23.10 From spiral tree to tree map [VBS11]: (a) defining a directed angle-restricted Steiner tree. Workflow: (b) (approximate) spiral tree, (c) thickening and subdividing edges, (d) optimizing aesthetic criteria using the method of deepest descent.

parameters in order to smooth and straighten the tree, to avoid obstacles, to balance and to maintain its original angles (see Figure 23.10(d)). The authors model these parameters by defining cost functions; they apply the method of deepest descent in order to minimize the global cost function, which is the weighted sum of the individual cost functions. In order to ensure that no crossings are introduced in the optimization process, the algorithm checks for intersections before each move. In case an intersection would occur, the movement vector is repeatedly divided by 2 until the movement is safe. The edges are drawn as a new type of cubic Hermite splines that approximates logarithmic spirals well.

For an example output of the method of Verbeek et al., see Figure 23.9(c). For comparison, the results of two other, more general methods (by Cui et al. [CZQ+08] and Holten and van Wijk [HVW09]) are also depicted; see Figures 23.9(d) and (e). The input to these methods is a graph (with vertex positions) rather than a tree; in the output, the curved edges are *bundled* in order to better reflect the structure of the graph. Concerning running time, Verbeek et al. report that their algorithm drew most flow maps in less than a minute on a (dual-core) Pentium-D 3-GHz processor with 1 GB of RAM, whereas world maps required a few minutes.

23.5 Plane and Near-Plane Graphs

There are a number of applications where plane or near-plane graphs have to be drawn. We differentiate between four different types of applications. In all four types, the original embedding can be made planar by introducing few extra nodes where roads or tracks cross, for example, at bridges. The topology of the original embedding must be preserved and edges are drawn as polygonal lines. In most cases it is desirable to keep vertices roughly in the same place as in the original embedding or to at least preserve the relative position of vertices (for example, left/right, above/below). This helps the user to keep his mental map.

The application types that we consider in this section are as follows.

Schematic road maps are used for road or transportation networks. They try to keep vertices (that is, cities or junctions) at or close to their original location. Edges (that is, roads or tracks) can have diagonal segments.

Metro maps also use diagonals, but other than schematic maps they use very different scales for downtown versus suburban areas. Relative position is important. Another special feature of metro maps is that they usually have many degree-2 nodes.

Street maps with focus regions do not restrict edge directions but allow the user to *select* a region that is displayed at larger scale. This is different from the

usual zoom operation where the user sees only a fraction of the original map and, hence, loses overview. The difficulty lies in squeezing the remaining part of the map such that distortion is acceptable.

Cable plans are used for documenting the layout of communication networks. They are drawn orthogonally and try to preserve the angles, but not the distances of the original embedding.

Compared to the orthogonal drawing of (embedded) graphs [Tam87], the introduction of diagonals yields drawings that are more similar to the original embedding. In addition, the maximum node degree increases from 4 to 8. In a sense, however, the problem becomes more difficult as Bodlaender and Tel [BT04] point out. They define a planar graph to be *d-linear* if it can be embedded such that all angles are multiples of $2\pi/d$. The *angular resolution* of a plane straight-line drawing is the minimum angle between edges incident to a common vertex, over all vertices. Bodlaender and Tel show that, for $d = 4$, an angular resolution of $2\pi/d$ implies d-linearity and that this is not true for any $d > 4$.

In what follows, we refer to the set of directions that are given by the two coordinate axes and their two bisectors as the *octilinear directions*.

23.5.1 Schematic Road Maps

Schematic maps usually try to preserve the position of vertices as much as possible while simplifying the polygonal lines that represent edges without changing the topology of the original drawing. Edges are drawn as x- and y-monotone paths that consist of usually no more than three horizontal (H), diagonal (D), and vertical (V) line segments. Cabello et al. [CdBvD$^+$01] have given an algorithm that decides in $O(n \log n)$ time whether a node-embedded graph can be drawn such that each edge follows one of a given set of allowed segment sequences (such as $\{\text{HVH}, \text{VDV}\}$, for example). If an edge embedding of the required type exists, the algorithm finds it.

While Avelar and Müller [AM00] also try to make edges octilinear, they use a very different method that moves vertices based on local decisions. They guarantee that the topology of the original network is kept, but they do not guarantee that every edge in the final layout is actually octilinear. They first use a polygonal line simplification method to simplify all edges (that is, polygonal lines) of the original embedding. In order to preserve topology, a more involved method like Saalfeld's [Saa99] must be used instead of the classical method of Douglas and Peucker [DP73] mentioned above.

After the simplification, each street junction and each bend of a street is considered a vertex. Hence, edges are straight-line segments. Avelar and Müller iteratively go through all vertices and compute new destinations based on the current (imperfect) directions of the incident edges. They do this as follows. For each vertex v and each vertex w incident to v, they compute an offset for v that would make the edge vw confirm to one of the allowed directions. The arithmetic mean of these offsets yields a tentative new position for v. Before actually moving a vertex, Avelar and Müller check the topology of the resulting embedding. If topology would change, they restrict the vertex movement accordingly. They continue to change vertex positions until all edges follow one of the desired four directions or until the number of iterations has reached a fixed threshold.

23.5.2 Metro Maps

The problem of drawing maps of subways and other means of public transportation is an interesting compromise between schematic maps where vertex positions are (mostly) fixed

and "conventional" graph drawing where vertices can go anywhere. The first approach maximizes (user) orientation, the second aesthetics.

We now define the problem in graph-drawing terms. Let $G = (V, E)$ be the input graph. We assume that G is *plane*, that is, G comes with a planar embedding. We actually assume that we know the geographic location $\Pi(v)$ of each vertex $v \in V$ in the plane and that the straight-line embedding induced by the vertex locations is plane. In case some edges cross others, we simply introduce dummy vertices that represent the crossings. Let $\mathcal{L}$ be a *line cover* of G, that is, a set of paths of G such that each edge of G belongs to at least one element of $\mathcal{L}$. An element $L \in \mathcal{L}$ is called a *line* and corresponds to a metro line of the underlying transport network. We refer to the pair $(G, \mathcal{L})$ as the *metro graph*. The task is now to find a drawing Γ of $(G, \mathcal{L})$ according to a set of rules (which we will discuss later).

In the last few years, a number of methods for automating the drawing of metro maps have been suggested. The author [Wol07] surveyed the area earlier, with an emphasis on experimental comparison. Our treatment here is more compact, but adds some recent development. Before we go into the methods, let us quickly turn to the origins of the problem.

History. While metro networks were small in size, it made perfect sense to draw them geographically. This was easy for the graphic designers and gave map users a sense of distance, for example, between stations that are close to each other in the above-ground street network but far in the underground metro network: sometimes it is indeed faster to walk a little more than to reach the metro stop closest to one's destination. Electrical draftsman Henry Beck was the first to draw a metro network in a schematic way. His rationale was that connection information and the number of stops on a line are more important information for the network user than geographic distances. His design was so revolutionary that the London Transport Authority, in 1931, rejected his first proposal and only in 1933 dared to print and sell Beck's map. Therefore, Berlin got the honor of having the first printed schematic metro map (in 1931). While the Nazis in Berlin soon moved back to a geographic layout [Pol06], Beck's tube map was an instant success and became the basis of all subsequent official maps of the London Underground. In 2006, his original map was elected, right after the supersonic airplane Concorde, the second-most popular British design icon of the twentieth century [Wik12]; it has an interesting history in its own right [Gar94]. In the meantime, graphic designers have invented different layout styles all over the world (see the book of Ovenden [Ove03]), but the use of the octilinear set of directions for drawing is still prevailing.

Complexity. Using eight edge directions seems to be a good compromise between an unrestricted drawing and the restriction to the four orthogonal (or rectilinear) edge directions predominant in circuit diagrams, VLSI layout, and—traditionally—in graph drawing. As it turns out, the additional freedom that an octilinear layout gives the designer compared to a rectilinear layout comes at a price. Nöllenburg [Nöl05b] proved, by means of a visually very appealing reduction from Planar3Sat, that it is NP-hard to decide whether a plane graph has an octilinear drawing. This is in sharp contrast to the rectilinear case, for which Tamassia [Tam87] showed that the same question can in fact be answered efficiently. In his seminal paper, the theoretical foundation of orthogonal graph drawing, Tamassia reduced the problem to a network flow problem, which yields an orthogonal drawing with the minimum number of bends and small area.

Curve evolution. The first attempt to automate the drawing of metro maps was made by Barkowsky et al. [BLR00]. They use an algorithm for polygonal line simplification, which they call *discrete curve evolution* [LL99], to treat the lines of the Hamburg subway system. Their algorithm, however, neither restricts edge directions nor does it increase

station distances in the crowded downtown area. Stations are labeled but no effort is made to avoid label overlap.

Force-directed layout. Hong et al. [HMdN06] give five methods for the metro-map layout problem. The most refined of these methods modifies PrEd [Ber99], a topology-preserving spring embedder, such that edge weights are taken into account and such that additional magnetic forces draw edges toward the closest octilinear direction. Edges are drawn as straight-line segments connecting the corresponding vertices. Relative position is only taken into account implicitly by using the original embedding as initial layout.

In a preprocessing step, Hong et al. simplify the metro graph by contracting each edge that is incident to a degree-2 vertices. After performing all contractions, the weight of each remaining edge is set to the number of original edges it replaces. After the final layout has been computed, all degree-2 vertices are re-inserted into the corresponding edges in an equidistant manner. Due to this preprocessing the numbers of vertices and edges decrease by a factor of 3 to 8, and all networks (with 22 to 92 vertices and 32 to 317 edges after contraction) were solved within 0.2 to 22 seconds. Station labels are placed in one out of eight directions using the interactive LabelHints system [dNE03]. While label–label overlaps are avoided, diagonally placed labels sometimes intersect network edges.

The results of Hong et al. [HMdN06] are clearly superior to those of Barkowsky et al. [BLR00]. However, they are still not very similar to commercial maps drawn by graphic designers. The main deficiency is that most edges in the final layouts are close to, but not quite octilinear. This seems to be due to the fact that the magnetic forces that determine the layout are the sum of many conflicting terms.

Local optimization. Stott et al. [SRMW11] draw metro maps using multicriteria optimization based on hill climbing. For a given layout they define metrics for evaluating the octilinearity and the length of edges, the angular resolution at vertices and the straightness of metro lines. The quality of a layout is the sum over these four metrics. Their optimization process is iterative. They start with a layout on the integer grid that is obtained from the original embedding. In each iteration they go through all vertices. For each vertex they consider alternative grid positions within a certain radius that shrinks with each iteration. For each of these grid positions they compute the quality of the modified layout. If any of the positions improves the quality of the layout, they move the current vertex to the position with the largest improvement among those positions where the topology of the layout does not change. After implementing their algorithm they observed a typical problem of local optimization: overlong edges are often not shortened since this would need moving several vertices at the same time. For a *bridge*, that is, an edge whose removal disconnects the graph, this can easily be fixed by moving all nodes of the smaller component closer to the larger component. They run this fix after each iteration for all bridges.

Stott et al. have experimented with enforcing relative position, but report that the results were disappointing as there were many situations where a better layout could only be found by violating the relative position of some vertices. They can label stations, but do not check for overlaps other than with the edges incident to the current station. They use the same contraction method as Hong et al. [HMdN06] to preprocess the input graph. Even with this preprocessing their algorithm is much slower. For example, an earlier version of their algorithm [SR05] drew the simplified Sydney CityRail network in about 4 minutes and the unsimplified network in 28 minutes; the new algorithm (in Java 1.6 on a 1.4-GHz Celeron M machine with 1.5 GB RAM under Windows XP) needs about two hours for the labeled network. This compares with the 7.6 seconds that Hong et al. need for the simplified, but labeled network. The drastic increase in running time, however, is worth it—in the resulting maps nearly all edges are octilinear, which makes the maps more legible.

Global optimization. Nöllenburg and Wolff [NW11] draw metro maps using the toolbox of mathematical programming. They approach the problem by setting up the following list of design rules which are based on the design of real-world metro maps.

(R1) Restrict the drawing of edges to the octilinear directions.

(R2) Do not change the geographical network topology. This is crucial to support the mental map of the passengers.

(R3) Avoid bends along individual metro lines, especially in interchange stations, to keep them easy to follow for map readers. If bends cannot be avoided, obtuse angles are preferred over acute angles.

(R4) Preserve the relative position between stations to avoid confusion with the mental map. For example, a station being north of some other station in reality should not be placed south of it in the metro map.

(R5) Keep edge lengths between adjacent stations as uniform as possible with a strict minimum length. This usually implies enlarging the city center at the expense of the periphery.

(R6) Stations must be labeled and station names should not obscure other labels or parts of the network. Horizontal labels are preferred and labels along the track between two interchanges should use the same side of the corresponding path if possible.

(R7) Use distinctive colors to denote the different metro lines. This means that edges used by multiple lines are drawn thicker and use colored copies for each line.

A subset of these rules has also been listed by Hong et al. [HMdN06].

Nöllenburg and Wolff divide their rules into strict requirements, also called *hard constraints*, and into aesthetic optimization criteria, also called *soft constraints*. Their hard constraints are:

(H1) *Octilinearity:* For each edge e, the line segment $\Gamma(e)$ in the output drawing must be octilinear.

(H2) *Topology preservation:* For each vertex v, the circular order of its neighbors must agree in Γ and the input embedding.

(H3) *Minimum length:* For each edge e, the line segment $\Gamma(e)$ must have length at least ℓ_e.

(H4) *Minimum distance:* Each edge e must have distance at least $d_{\min} > 0$ from each non-incident edge in Γ.

Constraint (H1) models the octilinearity requirement (R1). It is this constraint that makes the problem NP-hard [Nöl05a], see the discussion in the paragraph on complexity above. Constraint (H2) models the topology requirement (R2), (H3) models the minimum edge length in (R5), and (H4) avoids introducing additional edge crossings and thus also models a part of (R2). This is because two intersecting edges would have distance $0 < d_{\min}$.

The soft constraints should hold as tightly as possible. They determine the quality of Γ and are as follows:

(S1) *Straightness:* The lines in $\mathcal{L}$ should have few bends in Γ, and the bend angles ($< 180°$) should be as large as possible.

(S2) *Geographic accuracy:* For each pair of adjacent vertices (u, v), their relative position should be preserved, that is, the angle $\angle(\Gamma(u), \Gamma(v))$ should be similar to the angle $\angle(\Pi(u), \Pi(v))$, where $\angle(a, b)$ is the angle between the x-axis and the line through a and b.

(S3) *Size:* The total edge length of Γ should be small.

Clearly, constraint (S1) models minimizing the number and "strength" of the bends (R3) and (S2) models preserving the relative position (R4). The uniform edge length rule (R5) is realized by the combination of a strict lower bound of unit length (H3) and a soft upper bound (S3) for the edge lengths. Rule (R4) for the relative position can be interpreted as both a soft and a hard constraint, for example, by restricting the angular deviation to at most 90° as a hard constraint and charging costs for smaller deviations as a soft constraint.

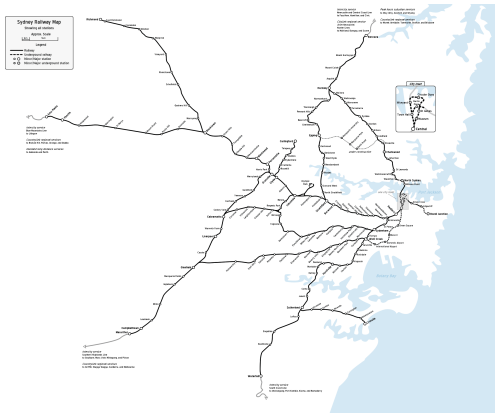
Nöllenburg and Wolff then show that the existence of a drawing that fulfills the hard constraints (H2)–(H4) and optimizes a weighted sum of the soft constraints can be formulated as a mixed-integer linear program (MIP). The basic idea behind their formulation is as follows. Each edge has a number of binary variables that correspond to its feasible octilinear directions. Exactly one of these variables must be 1. All other constraints regarding an edge, such as its minimum length and minimum distance from other edges, are expressed for each feasible direction. The constraints are designed such that they are trivially fulfilled if the edge has a different direction. Angles are "measured" in multiples of 45°, for example, in soft constraint (S 1), an angle is punished proportionally to its degree of acuteness: the bend of the edges uv and vw incident to a vertex v can be of size 180°, 135°, 90°, or 45°. The bend cost of this bend is 0, 1, 2, or 3, respectively. Expressing this with linear constraints is somewhat tricky, but it can be done using the directions of the edges uv and vw and two new binary variables per bend.

In general it is NP-hard to solve a MIP, but highly optimized commercial solvers such as Cplex or Gurobi can solve relatively large MIPs relatively quickly. Consider a medium-sized metro system such as the CityRail network of Sydney with 10 lines and 174 stations. For this network, the MIP of Nöllenburg and Wolff as sketched above consists of roughly 38,000 variables and 150,000 constraints—assuming that one applies the obvious data reduction trick of replacing each path of k degree-2 vertices by a single edge of length at least $k \cdot d_{\min}$. Actually, Nöllenburg and Wolff proposed to keep up to two vertices between each pair of neighboring interchange stations so as to have some flexibility for making bends; this helps to be more accurate in terms of relative position (geographic accuracy). Solving such a MIP to optimality can take days.

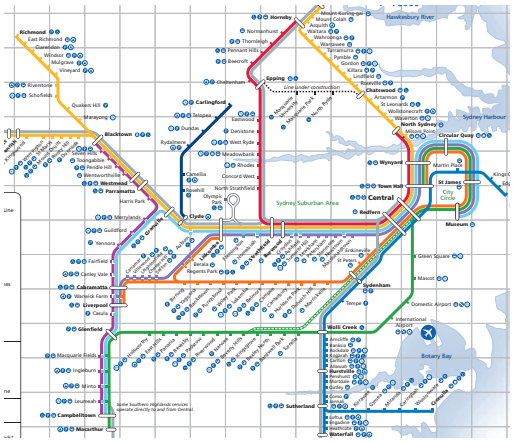
Therefore, Nöllenburg and Wolff described a number of ways in order to further reduce the size of the MIP. Their fastest approach is based on the so-called *callback function* of the Cplex solver. It allows them to set up the MIP without any planarity constraints according to hard constraint (H4), check any intermediate feasible solution for crossings and then add constraints needed to forbid the specific crossings at hand. For the reduced Sydney example, this yields a MIP with roughly 4800 variables and 3500 constraints; constraints for just three edge pairs were added during optimization. Still, computing the layout in Figure 23.11(c) from the geographic input depicted in Figure 23.11(a) took about 23 minutes. For a comparison with the work of a professional graphic designer, see Figure 23.11(b).

Things get worse when drawing maps with station labels that can change sides with respect to metro lines. Even when aggregating all labels between two interchanges into one big label (that is then modeled as a dummy metro line) and taking advantage of the callback functionality, the MIP ends up having nearly 93,000 variables and 22,000 constraints. Computing the layout in Figure 23.11(d) took 10.5 hours; in both cases an optimality gap of about 16% remained, that is, the solver knows that the unknown objective value of an optimal solution is at most 16% less than that of the layouts in Figures 23.11(c) and 23.11(d).

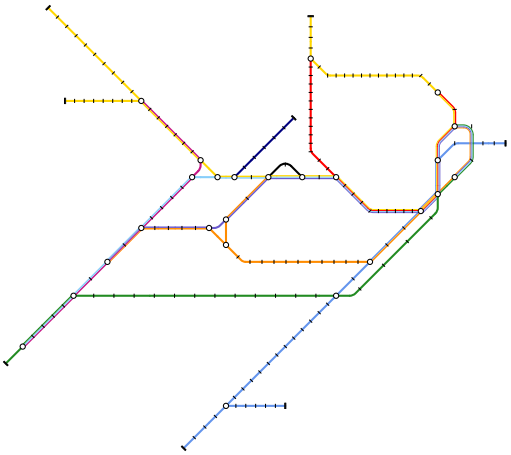
Least squares. Wang and Chi [WC11] presented a system for octilinear on-demand focus-and-context metro maps that highlight routes returned by a route planning



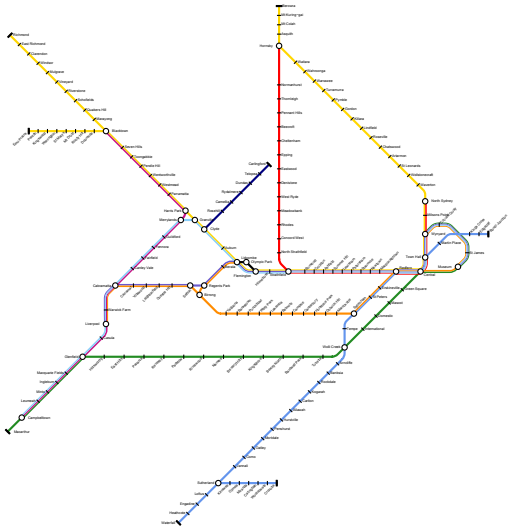
(a) geographic layout (by John Shadbolt)



(b) corresponding clipping of the official map [Syd08]



(c) unlabeled layout of Nöllenburg and Wolff



(d) labeled layout of Nöllenburg and Wolff

Figure 23.11 The Sydney CityRail network.

system while showing the rest of the network as less important context information. It can also be used to draw non-focused metro maps. They deform the given geographic map by the conjugate gradient method [HS52] in a least-squares sense, minimizing a set of energy terms that model the aesthetic constraints. Labeling is performed independently. Their method is both fast and creates good layouts, e.g., for mobile devices.

Metaphor. Sandvad et al. [SGSK01] and Nesbitt [Nes04] use the *metro-map metaphor* as a way to visualize abstract information. A particularly nice example is the map that shows the O'Reilly open source product lines [O'R03], see Figure 23.12.

Research of the metro-map layout problem triggered the investigation of a new subproblem, metro-map *line crossing minimization*. In that problem, one assumes that the layout of the underlying metro graph is known; the aim is to order the metro lines on each edge such that the number of line crossings is minimized [BNUW07]. We do not treat the problem here since its nature is purely combinatorial, not geometric.

Beyond Henry Beck Recently, a completely different style for drawing metro maps has attracted considerable attention: the *curvilinear* style. Roberts et al. [RNL+11] did user studies to compare (hand-drawn) schematized maps to (hand-drawn) maps where the Metro lines are represented by Beziér curves. Surprisingly, users were up to 50 % faster in completing certain planning tasks with the new and unfamiliar Beziér maps rather than with schematized maps. Still, being used to schematized maps, they liked them better.

These findings prompted Fink et al. [FHN+13] to investigate ways to automate the process of drawing metro maps with Bézier curves; see Figure 23.13. They use a force-directed approach. Starting with a straight-line or octilinear input drawing (see Figure 23.13(a)),

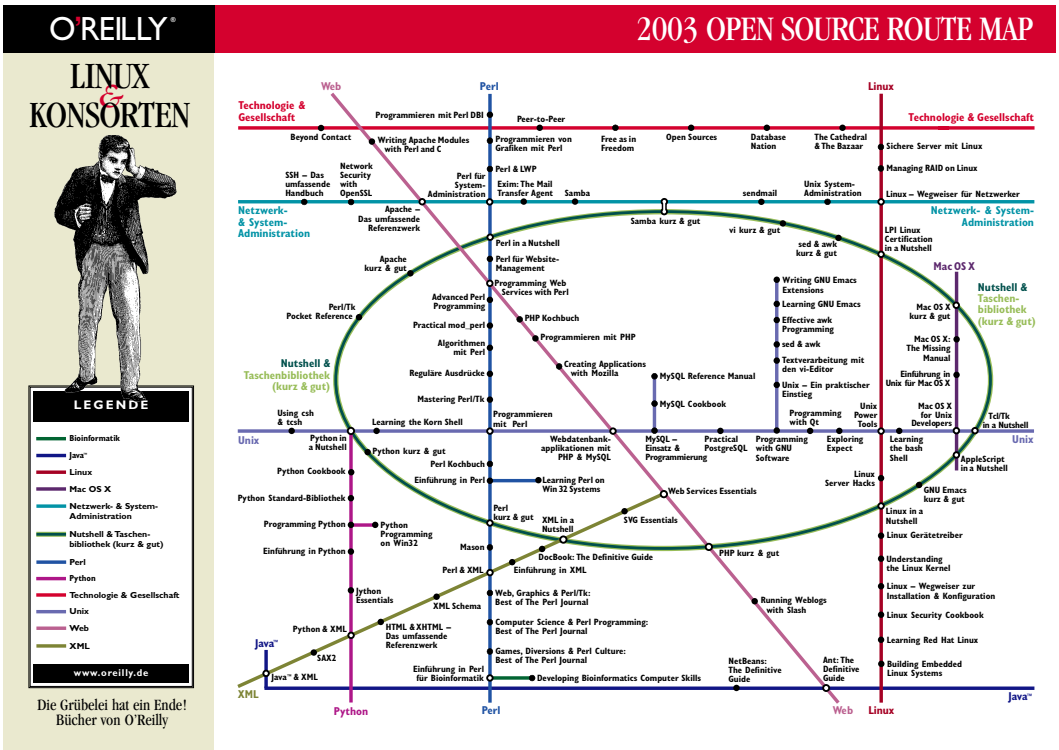
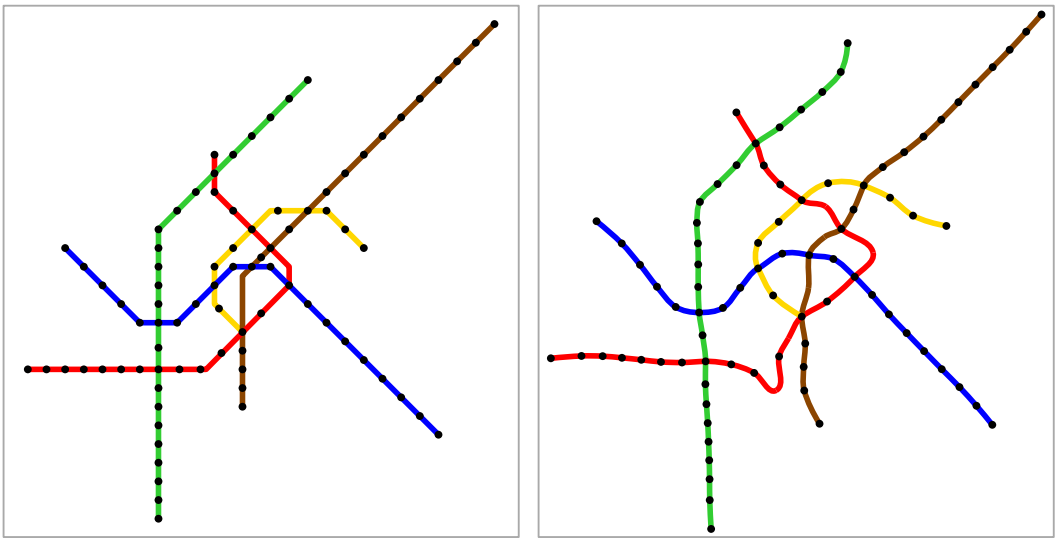
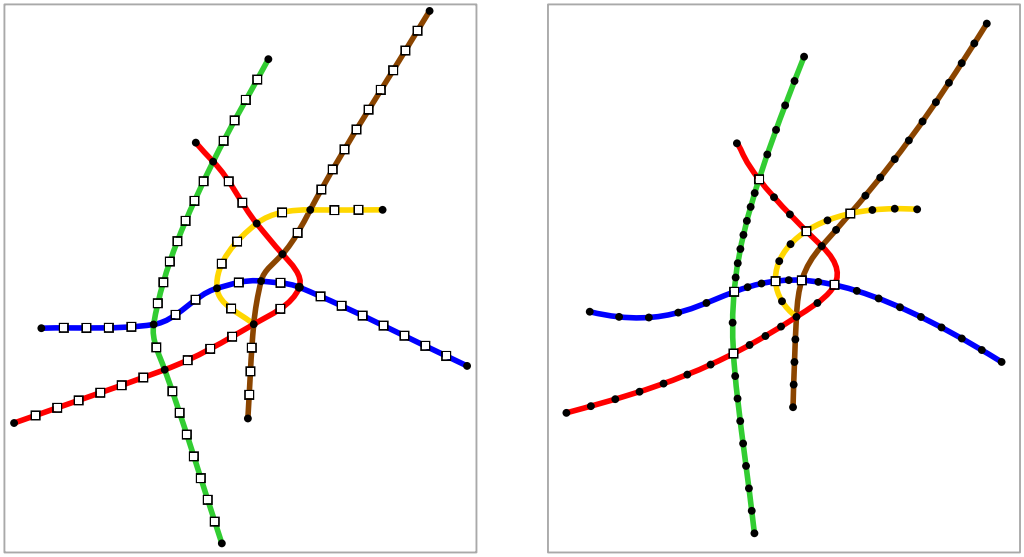


Figure 23.12 O'Reilly's open source product lines 2003.



(a) octilinear input drawing, (b) drawing without virtual vertices



(c) drawing with virtual degree-2 vertices (d) additionally, with virtual degree-4 vertices

Figure 23.13 Metro network of Vienna drawn using Bézier curves [FHN+13].

the authors go through each metro line and replace each line segment by a nearly-straight cubic Bézier curve that shares tangents with its predecessor and successor. Then they apply attracting and repulsive forces to vertices, but also to tangents. The aim is to merge as many consecutive Bézier curves on each metro line as possible in order to reduce the visual complexity; compare Figures 23.13(b), (c), and (d). Vertices that are incident to merged edges only are called *virtual*; forces can no longer be applied to them. In all but the last iteration, merges happen only at degree-2 vertices. In the final iteration, degree-4 vertices are handled, too.

Whereas the results are quite nice for small networks (such as Montreal or Vienna), long

metro lines in complex networks (such as London) remain too wiggly. A number of Bézier curves could not be merged due to contradicting constraints. The Java implementation of Fink et al. drew the London Underground (20 metro lines with 200 stations, 150 of which are degree-2 vertices) in 224 seconds on a 3-GHz dual-core computer with 4 GB RAM.

23.5.3 Street Maps with Focus Regions

Metro maps quite heavily distort distances in order to show more details in crowded downtown areas, independently of the style used for drawing the edges. The same idea is used in city maps, for example, by the German map maker Falk-Verlag who, in 1945, published its first map of Hamburg with a very mild kind of fisheye view with scale varying from 1:16.000 in the downtown area to 1:18.500 in the suburbs. Interestingly enough, the idea to use a non-uniform scale was due to the fact that the post-war military government allotted only paper of size 60 cm × 40 cm to the newly founded four-man company [Hol95]. That size would not have been enough to cover the intended part of the city *and* display the downtown in enough detail.

A major difference between metro maps and Falk-style city maps (that is, fisheye-based map representations) is the fact that in a schematic metro map not just scale, but also the *change* in scale is (highly) non-uniform. Jenny [Jen06] has analyzed and visualized distortion in metro maps, arguing that less distorted maps are to be preferred.

An idea more similar to the metro-map approach has been used by Haunert and Sering [HS11] in order to draw street networks with focus regions. Their aim is to redraw a street map within the same view frame as the original map, but such that a region specified by the user is enlarged by a given factor. Haunert and Sering model their problem as a *quadratic program* (QP), that is, a mathematical program consisting of real-valued variables, a set of quadratic constraints, and a quadratic objective function. Their QP has the additional property that both the objective function and the feasible region, that is, the set of variable vectors that fulfill all constraints, are convex. Such a convex QP can be solved efficiently.

Since the core of their QP formulation is quite simple, we present it here. We assume that we are given a plane graph $G = (V, E)$ with an input drawing that is completely determined by the positions of the vertices, that is, for each vertex $v \in V$, we know its coordinates X_v and Y_v. Moreover, we are given a subset $V' \subseteq V$ representing the focus region that is to be scaled up by a *zoom factor* $Z > 1$. Now, for each node $u \in V$, we introduce three variables: the unknown coordinates $x_u, y_u \in \mathbb{R}$ of u in the output drawing and an unknown scale factor $s_u \in \mathbb{R}^+$. We now impose constraints on these variables.

First, we define a constraint to ensure that the output drawing remains within the bounding box of the input drawing.

$$\begin{aligned} \min_{v \in V}\{X_v\} \leq x_u \leq \max_{v \in V}\{X_v\} \\ \min_{v \in V}\{Y_v\} \leq y_u \leq \max_{v \in V}\{Y_v\} \end{aligned} \qquad \text{for each } u \in V \tag{23.1}$$

Second, we fix the scale factor for each node in the focus region:

$$s_u = Z \qquad \text{for each } u \in F \tag{23.2}$$

For a node $u \notin F$, we don't know its scale factor s_u; we will determine it through the optimization, together with the coordinates of u in the output map. It remains to ensure that the scale factor s_u is valid for the neighborhood of u.

Suppose that we would express the idea of a locally valid scale factor with the constraint

$$\begin{array}{r} s_u(X_v - X_u) = (x_v - x_u) \\ s_u(Y_v - Y_u) = (y_v - y_u) \end{array} \qquad \text{for each } u \in V,\ v \in \mathrm{Adj}(u), \tag{23.3}$$

where $\mathrm{Adj}(u)$ is the set of neighbors of u in G. With constraint (23.3), the star-shaped subgraph of G that contains u and its neighbors is scaled by s_u. For two adjacent nodes i and j, however, we can only satisfy this constraint if we set $s_i = s_j$. Therefore, if G is connected, we would have to select the same scale factor for all nodes in V. With constraint (23.3), it is thus impossible to design a *variable*-scale map.

In order to allow for different scale factors in different parts of the map, we introduce a relaxed version of constraint (23.3). We do not require that the neighborhood of node u is *exactly* mapped to scale. Instead, we allow for small distortions, which we measure based on residuals δx_{uv} and δy_{uv}. For this purpose, we introduce, for each edge $uv \in E$, auxiliary variables δx_{uv} and δy_{uv} into our model. Relaxing constraint (23.3) simply yields

$$\begin{array}{r} \delta x_{uv} = s_u(X_v - X_u) - (x_v - x_u) \\ \delta y_{uv} = s_u(Y_v - Y_u) - (y_v - y_u) \end{array} \quad \text{for each } u \in V\,, v \in \mathrm{Adj}(u)\,. \tag{23.4}$$

If both u and v lie in the focus region F, we require

$$\delta x_{uv} = \delta y_{uv} = 0 \quad \text{for each } u\,, v \in F\,, v \in \mathrm{Adj}(u)\,. \tag{23.5}$$

This makes sure that edges in the focus region indeed become enlarged by the zoom factor Z.

Our objective is to minimize the *weighted* square sum of the residuals:

$$\text{Minimize} \sum_{u \in V} \sum_{v \in \mathrm{Adj}(u)} \Big(\big(w(u,v) \cdot \delta x_{uv}\big)^2 + \big(w(u,v) \cdot \delta y_{uv}\big)^2 \Big) \tag{23.6}$$

with $w(u,v) = 1/\sqrt{(X_v - X_u)^2 + (Y_v - Y_u)^2}$. With this weight setting, we express that the validity of the scale factor s_u decreases with increasing distance from node u. This finishes the description of the core of the QP. All its constraints are linear; its objective function is convex since it doesn't contain mixed terms and all the weights are positive. Therefore, the core QP can be solved efficiently.

Unfortunately, the core QP does not prevent edge crossings. Crossings are unlikely to occur in triangulations but they do occur in less strongly connected networks. Hence, an obvious idea is to triangulate the given plane graph G. Experiments, however, show that this ad-hoc solution produces drawings with rather high distortion all over the network. The reason is that the additional edges make the network inflexible. Sparse regions that otherwise can help to balance the expansion of the focus regions are artificially made dense.

A more promising approach is to define, for each pair of edges, a line that separates the two edges and to add new variables (the parameters of the line) and new constraints to the QP. As it turns out, the necessary constraints are such that the set of feasible solutions is not convex any more. In order to stay in the realm of convex quadratic programming, the authors came up with a clever trick. They simply removed one degree of freedom from the separating line; by fixing its slope. Clearly, adding the corresponding constraints to the QP yields a new QP that is more constrained than actually necessary. By choosing a "good" slope, however, the negative impact of the additional restriction can be kept small. Haunert

and Sering suggested to use the slope of the maximum-width strip that separates the two edges in the *input* drawing. The second trick they applied is to *not* add all new planarity constraints before solving the QP, but only in case the solution of the QP actually contains crossings. For each such crossing, exactly the constraints that forbid it are added to the QP, and the modified instance is given back to the QP solver. A similar trick was used by Nöllenburg and Wolff [NW11] in order to deal with planarity constraints in their MIP for drawing metro maps, see Section 23.5.2.

Concerning an example, consider the input instance depicted in Figure 23.14. This street network consists of 5864 vertices and 6675 edges. Applying the QP-based method to that input with the focus region represented by the black circle and a zoom factor of 2 took 51.8 seconds on a Windows PC with 3 GB main memory and a 3 GHz Intel dual-core CPU. The output is shown in Figure 23.15(a). For comparison, Figure 23.15(b) depicts the result of applying a fisheye transformation [YOT09] to the same input. Applying such a transformation takes only fractions of a second. Figure 23.15 also shows, in the small inlets on the right-hand side, the residuals of the street network, which can be seen as a measure for the deformation of the network. (The lower inlet has a legend that explains the color-coding.) Clearly, the method of Haunert and Sering yields very good solutions for drawing maps with focus regions. More work is needed, however, to come up with a method that is similarly good but much faster. This would be very interesting for all kinds of mobile applications.

Böttger et al. [BBDZ08] provide an interesting link between the schematized world of metro maps and the non-schematized world of city (street) maps. They show how to gradually morph a map showing both types of networks between a representation that is geographic and a representation where the map is distorted such that the metro network is schematized.

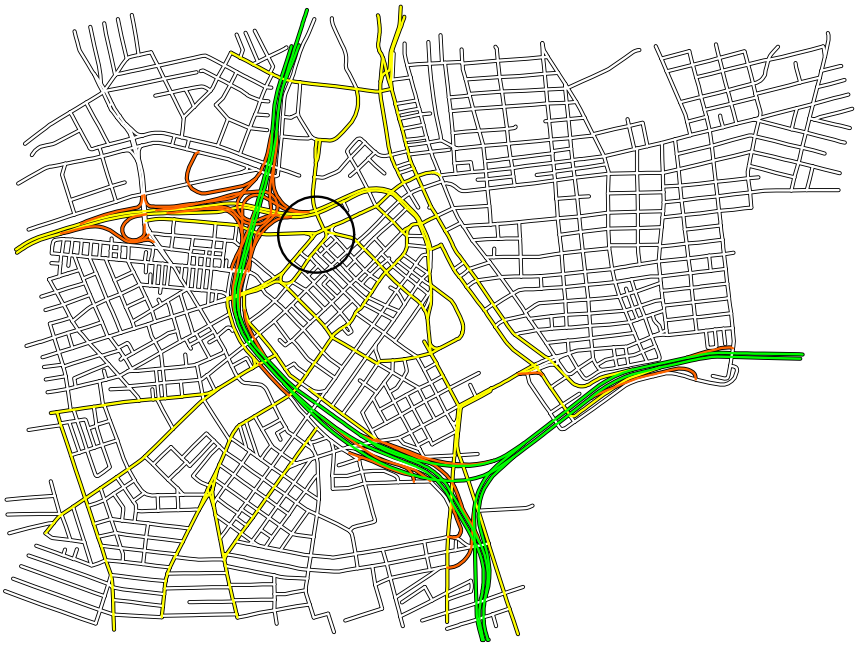
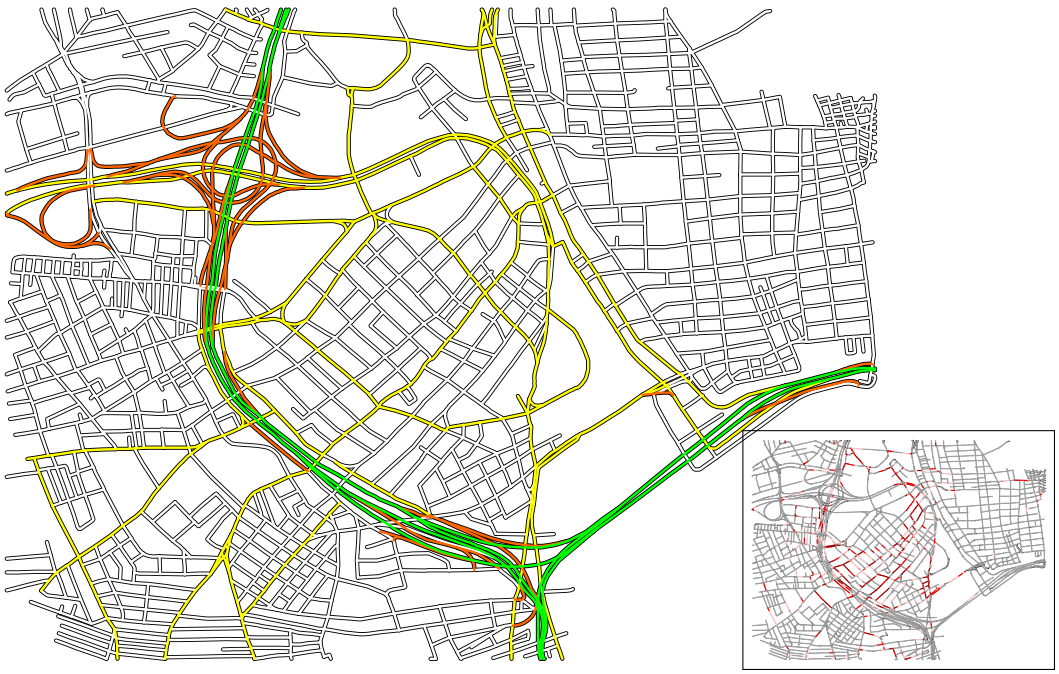
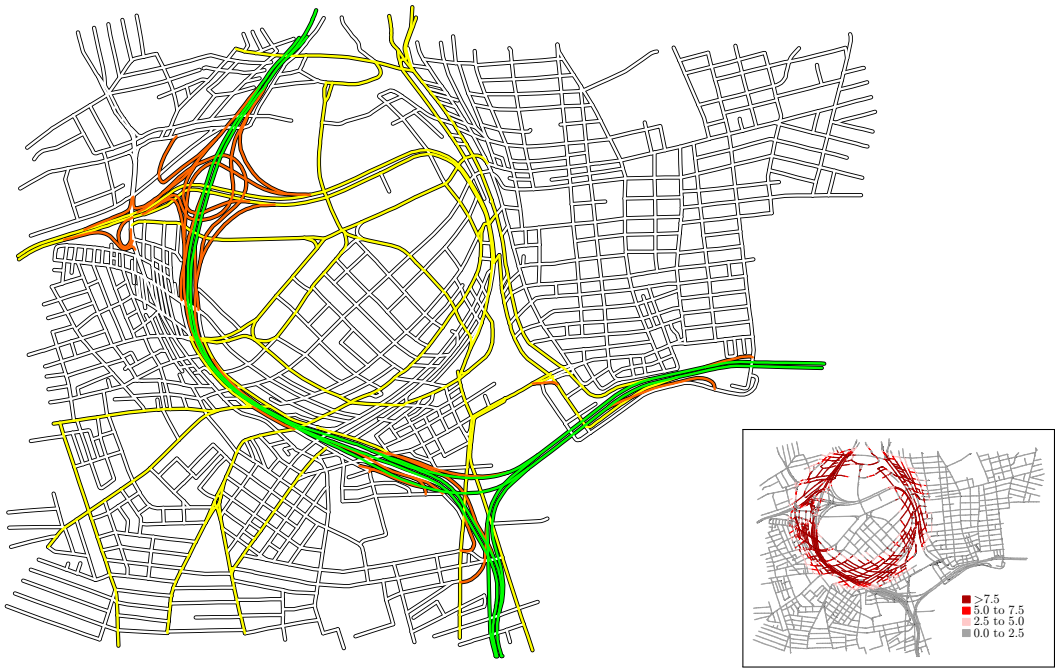


Figure 23.14 A street map (showing a detail of Providence, Rhode Island, U.S.A.) with a circular focus region that contains the conference site of InfoVis 2011.



(a) the method of Haunert and Sering [HS11]

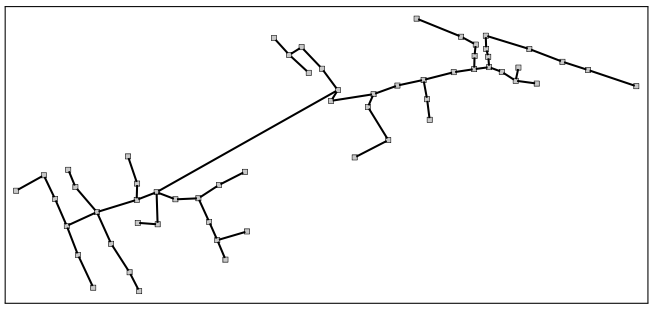


(b) the fish-eye transformation of Yamamoto et al. [YOT09]

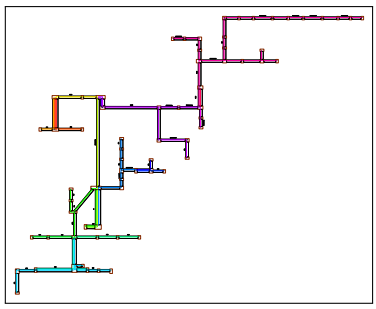
Figure 23.15 The results of applying two deformation methods to the map in Figure 23.14. The inlets show edges with residuals in red.

23.5.4 Cable Plans

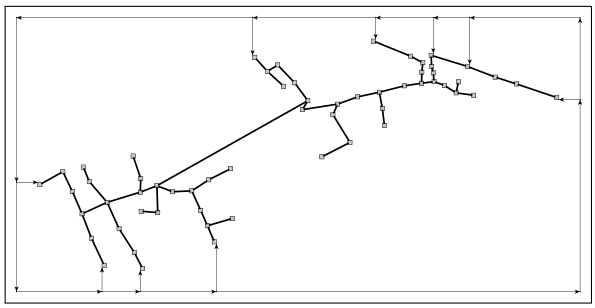
Lauther and Stübinger [LS02] briefly describe SCHEMAP, an iterative method to layout cable plans. Their method is based on a spring embedder and does not guarantee that all edges are drawn rectilinearly. Figures 23.16 (a) and (b) show the input to and the output of their method (in (b), the individual cables are drawn in various colors). Their preliminary work inspired Brandes at al. [BEKW02] who present an algorithm that produces an orthogonal drawing of a sketch of a graph. A sketch can be handmade or the physical embedding of a geometric network like the real position of telephone cables. Brandes et al. use a path-based min-cost flow formulation based on that of Tamassia [Tam87]. In order to stabilize tree-like subgraphs that stick into the outer face, they use dummy edges to connect all vertices on the convex hull of the original embedding to a rectangular frame that contains the whole embedding; see Figure 23.16 (c). The frame and the dummy edges are removed before the final layout (see Figure 23.16 (d)) is returned. Their algorithm runs in $O(n^2 \log n)$ time, where n is the number of vertices. The algorithm can, in principle, also be used to layout metro maps. It does not, however, allow for diagonals, and it does not explicitly take into account the special features and constraints of such maps.



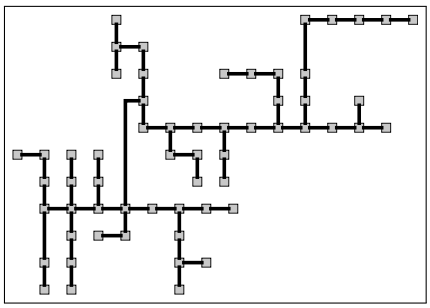
(a) input network (taken from [BEKW02])



(b) output of SCHEMAP [LS02]



(c) frame used by Brandes et al. [BEKW02]



(d) output of Brandes et al. [BEKW02]

Figure 23.16 Schematizing cable plans.

23.6 Other Graphs

In this section we consider graphs that do not fit into the classes we have treated in the previous sections. We focus on two scenarios, one scenario that has a geographic background—graphs that describe train connections (see Section 23.6.1)—and one scenario that uses the

cartographic-map metaphor to convey cluster information in (non-geographic) social networks such as collaboration graphs (see Section 23.6.3).

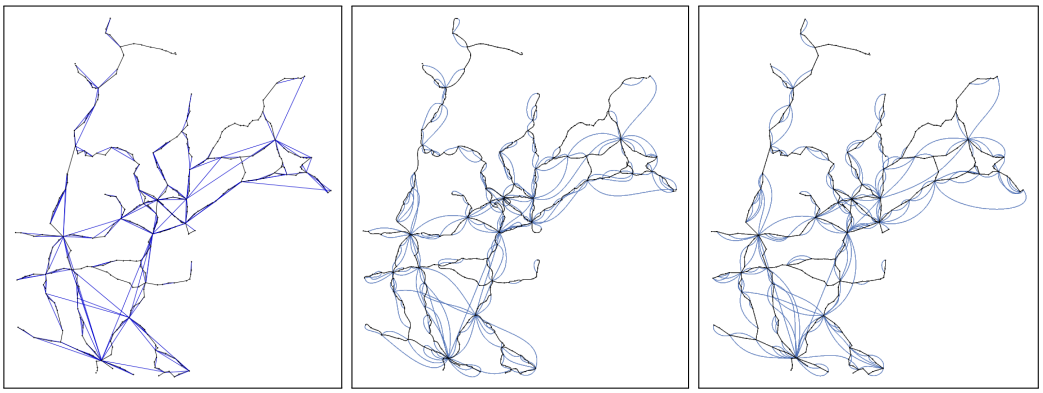
23.6.1 Timetable Graphs

A *timetable graph* has a vertex for each train station and an edge for each pair of stations connected by a train that does not stop in between. The graph is of interest to railway companies to check completeness and consistency of their schedules and to analyze changes between consecutive schedules. An obvious way to layout such graphs is to embed vertices at their geographic locations and edges as straight-line segments between them. However, this causes many edge crossings and small angles between edges along the same train line. Instead, Brandes and Wagner [BW00] introduce the concept of *minimal* and *transitive* edges. An edge $\{u, v\}$ is minimal if it corresponds to a piece of track that does not contain a station served by some other train. The remaining transitive edges correspond to through trains.

Whereas Brandes and Wagner use straight-line edges for minimal edges and long transitive edges, they suggest to use cubic Bézier curves [Béz72] to draw all other edges. Vertices are kept at their geographic location to allow for easy orientation. Then the layout problem consists of placing two control points for each Bézier curve. The authors define attractive and repulsive forces between control points and train stations within a local neighborhood. Using the random field layout framework [Bra99] and a customized version of the force-directed Fruchterman-Reingold method [FR91], they managed to draw even large timetable graphs nicely within minutes.

In subsequent work, Brandes et al. [BST00] explored ways to speed up their method and, at the same time, achieve perfect (or any prescribed) angular resolution in drawings of timetable graphs (and the Internet multicast backbone). They show that the flexibility of cubic Bézier curves allows them to optimize a number of criteria (with respect to the given straight-line drawing) in linear time by considering each vertex separately. This reduces the running time of their method on the same graphs as above from minutes to fractions of seconds. They refer to their new method as the *rotation* method. For a sample output, see Figure 23.17(b) and compare to the straight-line layout in Figure 23.17(a).

Unfortunately, due to the locality of the rotation method, the number of edge crossings and S-shaped edges increases. With a slightly different set of authors, Brandes et al.



(a) straight-line layout (b) rotation method (c) optimized by Gauss–Seidel

Figure 23.17 Timetable graphs of the sourroundings of Venice.

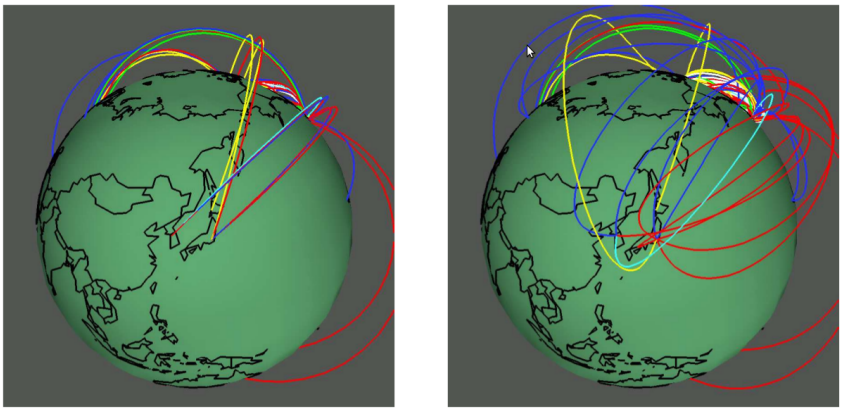


Figure 23.18 A comparison of elevated great circles (left) and Bézier curves output by the rotation method of Brandes et al. [BST00] (right).

[BSTW01] get a grip on these issues by using two new ingredients. First, the preprocess timetable graphs in order to make them more susceptible to the rotation method. Second, they introduce a new objective function that combines three criteria concerning the position and shape of edges, namely angular resolution, straightness, and roundness. They show that this objective function is a generalization of the layout function of Tutte's barycentric method [Tut63]. Therefore, the function has a unique minimum, which is the solution of a system of linear equations. Due to the size of their system, they resort in using the iterative Gauss–Seidel method, which in their case converges very fast. Their new algorithm is just about four times slower than the rotation method, and hence 50–100 times faster than the force-directed approach. In terms of aesthetics, the new method comes much closer to, but doesn't quite reach the force-directed approach; see Figure 23.17(c).

23.6.2 Internet Traffic

Clearly, computer scientists are interested in analyzing the structure of the largest man-made network, the Internet. Visualization plays an important role in this endeavor. Cox et al. [CEH96] created SeeNet3D, a tool that can be used to view and analyze traffic between routers of the Internet multicast backbone (MBone). The main view of the system represents routers at their (approximate) geographic locations on spherical or (slanted) plane maps, and it connects routers that communicate. The connections are drawn as circular arcs above the geodesics between the endpoints. To avoid clutter, the height of the arcs increases with the distance of its endpoints. SeeNet3D offers several synchronized views (spoke, helix, pincushion display) in order to facilitate data analysis.

Munzner et al. [MHCF96] extend the work of Cox et al. by using the Virtual Reality Modeling Language (VLMR 1.0) for the three-dimensional, spherical view. This allows them to display labels, modify the width of the arcs and let the user choose a rotation center different from the center of the sphere. For clutter removal, they also experiment with drawing edges only partially; namely near their endpoints.

Brandes et al. [BST00] propose a different, more traditional method for clutter reduction by applying their rotation method for timetable graphs (see Section 23.6.1) to the spherical setting, replacing the somewhat inflexible arcs by three-dimensional cubic Bézier curves. For a comparison, see Figure 23.18.

23.6.3 Social Networks

In order to get a grip on the problem of visualizing large graphs with vertex clusters, Gansner et al. [HGK10] came up with the idea of using the metaphor of a political map. In such maps, each country is colored such that no two neighboring countries use the same color. Gansner et al. take advantage of this well-known map style. Their tool *GMap* combines existing general-purpose graph drawing methods for visualizing the given graph (as a traditional node-link diagram) with new methods to create artificial maps whose countries correspond to the clusters in the graph. GMap also colors the countries, striving to make the color difference between adjacent countries large. Note that the GMap approach, while exploiting (the map-users exposure to) cartography, is about visualizing an *abstract* binary relation. Still, we found the idea of combining the drawing of graphs and maps so striking that we decided to discuss it in this chapter.

In their paper, Gansner et al. give specific solutions to two steps of the above approach, namely the steps of map making and of map coloring. The map-making step assumes that the given graph has been drawn and clustered; the authors suggest to use pairs of algorithms that have similar notions of distance, for example, multi-dimensional scaling [KW78] for drawing the graph and the k-means algorithm [Llo82] for clustering. The GMap implementation uses the GraphViz [GN00] spring embedder and modularity clustering [New06].

Making the map. Assuming a drawing of the given graph $G = (V, E)$, Gansner et al. first place vertex labels (with font size as some function of vertex weight). They use standard overlap removal techniques [GH10]. In order to subdivide the given rectangular map area such that each vertex v of G receives a cell that is large enough for its label ℓ_v, Gansner et al. use a Voronoi-based approach. Rather than directly computing the Voronoi diagram of the labels (which would give rise to rather artificial-looking regions), they select a set P_v of equidistant points on the boundary of ℓ_v and perturb them slightly; see the black dots in Figure 23.19. In order to avoid large regions with awkward shapes at the boundary of the given graph drawing, they insert random points in the "sea," that is, in the map region that is sufficiently far from the drawn graph; see the small circles in Figure 23.19. Then they compute the Voronoi diagram of the point set that they have constructed; see the gray tessellation in Figure 23.19. The region R_v that corresponds to a vertex v of G is the union of the Voronoi cells of the points in P_v. Let $V = \bigcup \mathcal{C}$ be the given clustering, that is, a partition of V. Then, for each cluster $C \in \mathcal{C}$, Gansner et al. simply define the

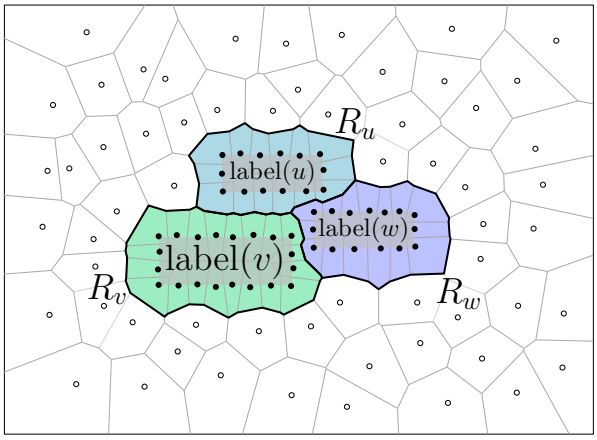


Figure 23.19 The Voronoi-based map-making step of GMap. Note that vertex v is more important than vertices u and w. Hence, v receives a label typeset in larger font size.

corresponding "country" to be $\bigcup_{v \in C} R_v$. This finishes the description of the map-making step.

For examples of maps that were generated with the method of Gansner et al., see Figures 23.20 and 23.21. The two graphs represent co-authorship of articles published at the International Symposium on Graph Drawing during the years 1994–2004 and 1994–2007. It is interesting to compare the traditional node-link diagram in Figure 23.20(a) with the corresponding map in Figure 23.20(b), which, technically, contains the same information—but in a much more accessible way. It is also interesting to observe the changes that occurred during the three additional years that were taken into account in Figure 23.21 as compared to Figure 23.20(b). Note that Figure 23.21 is a clipping of a slightly larger map that, apart from the "main land" has seven small islands (each with at most eight vertices)

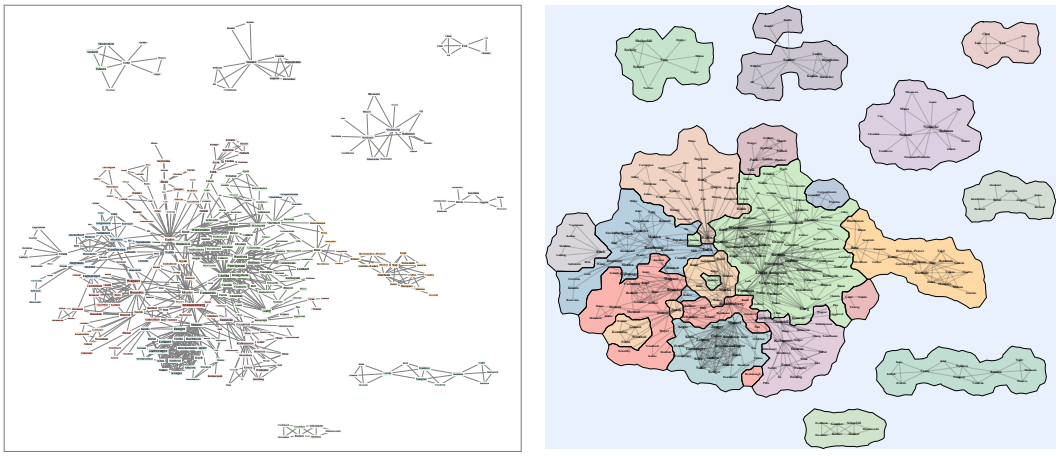
Coloring the map. In the last step of their approach, Gansner et al. color the countries of the map that they have computed. While the famous Four-Color Theorem ensures that four colors always suffice for maps whose country adjacency graph $G_c = (V_c, E_c)$ is planar, this does not hold if countries have exclaves (such as the Kaliningrad district, which is not connected to Russia proper, or Steve North, who is part of the AT&T cluster in Figure 23.20 but lies in a region disconnected from the main body of the cluster).

Gansner et al. model the coloring problem as follows. In order to handle exclaves properly, they insist that *each* of the $k := |\mathcal{C}| = |V_c|$ countries actually receives a different color. They assume that a set of colors in a linear color space has been predetermined so that the difference between the colors is roughly equidistant. Hence, they simplify the problem by asking for a (bijective) assignment $c\colon V_c \to \{1, \ldots, k\}$ of the k vertices of G_c to the numbers $1, \ldots, k$ such that

$$\sum_{uv \in E_c} (c(u) - c(v))^2 \tag{23.7}$$

is maximized over all such assignments. The problem is NP-hard [HKV11]. Therefore, they solve the continuous version of the problem, where $c'\colon V_c \to \mathbb{R}$ must fulfill the additional requirement that $\sum_{v \in V} (c'(u))^2 = 1$. This problem is solved when c' is the eigenvector corresponding to the largest eigenvalue of the Laplacian of G_c. As a heuristic for the discrete version of the problem, they let $c(u)$ be the rank of $c'(u)$ in the sorted sequence of the c'-values. They suggest to apply, in a post-processing step, a 2-opt type greedy algorithm that swaps the c-values of pairs of vertices whenever this increases the term (23.7). The combination of the two methods seems to yield good results in practice.

Concluding, even for large social networks such as co-authorship graphs or Amazon book co-purchase networks, the GMap yields very nice map-like visualizations. Recently, GMap has been extended to dynamic scenarios [MKH11, HKV12].



(a) traditional node-link diagram, label colors indicate cluster membership

(b) same graph drawing plus map background with countries corresponding to clusters

Figure 23.20 A portion of the co-authorship graph of articles published in the proceedings of the International Symposium on Graph Drawing in the years 1994–2004.

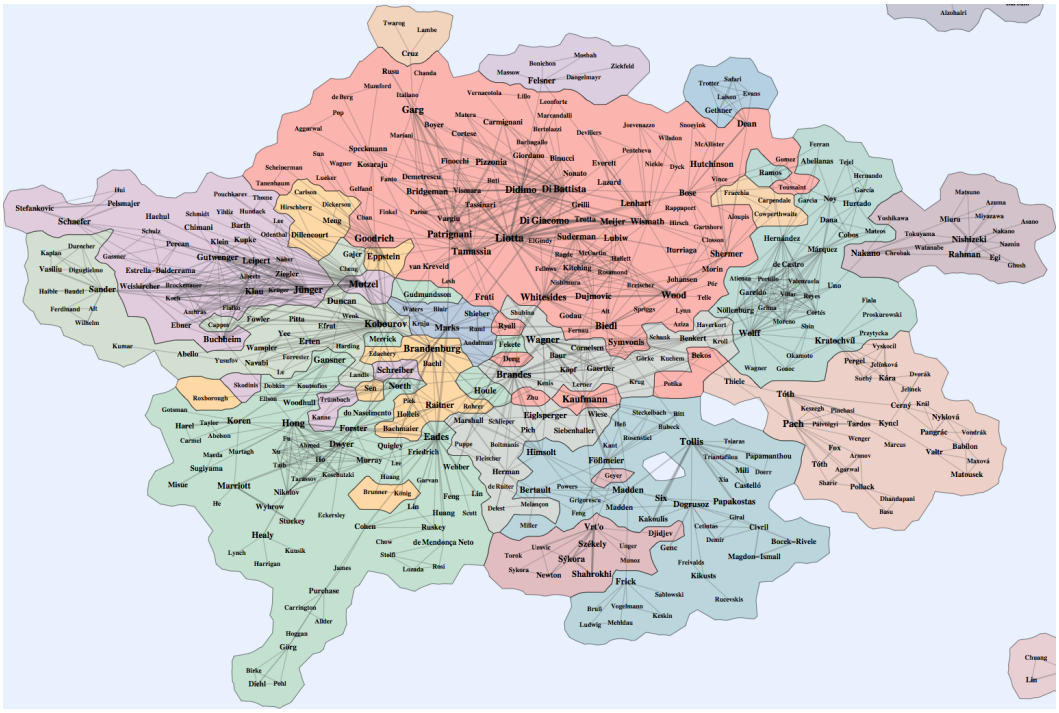


Figure 23.21 A portion of the co-authorship graph of articles published in the proceedings of the International Symposium on Graph Drawing in the years 1994–2007. The original map was clipped to increase the font size.

References

[AdCC+12] Nieves Atienza, Natalia de Castro, Carmen Cortés, M. Ángeles Garrido, Clara I. Grima, Gregorio Hernández, Alberto Márquez, Auxiliadora Moreno-González, Martin Nöllenburg, José Ramon Portillo, Pedro Reyes, Jesús Valenzuela, Maria Trinidad Villar, and Alexander Wolff. Cover contact graphs. *J. Comput. Geom.*, 3(1), 2012.

[AES99] Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29(3):912–953, 1999.

[AM00] Silvania Avelar and Matthias Müller. Generating topologically correct schematic maps. In *Proc. 9th Int. Symp. Spatial Data Handling (SDH'00)*, pages 4a.28–4a.35, 2000.

[AS01] Maneesh Agrawala and Chris Stolte. Rendering effective route maps: Improving usability through generalization. In Eugene Fiume, editor, *Proc. 28th Annu. Conf. Comput. Graphics Interactive Techniques (SIGGRAPH'01)*, pages 241–249. ACM Press, 2001.

[Ass96] Association for Geographic Information, London. GIS dictionary. `http://www.agi.org.uk/resources/dicitionary/content.htm`, 1996.

[AV00] Pankaj K. Agarwal and Kasturi R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete Comput. Geom.*, 23:273–291, 2000.

[BBDZ08] Joachim Böttger, Ulrik Brandes, Oliver Deussen, and Hendrik Ziezold. Map warping for the annotation of metro maps. *IEEE Comput. Graphics Appl.*, 28(5):56–65, 2008.

[BCC+06] Prosenjit Bose, Sergio Cabello, Otfried Cheong, Joachim Gudmundsson, Marc van Kreveld, and Bettina Speckmann. Area-preserving approximations of polygonal paths. *J. Discrete Algorithms*, 4(4):554–566, 2006.

[BEKW02] Ulrik Brandes, Markus Eiglsperger, Michael Kaufmann, and Dorothea Wagner. Sketch-driven orthogonal graph drawing. In Stephen G. Kobourov and Michael T. Goodrich, editors, *Proc. 10th Int. Symp. Graph Drawing (GD'02)*, volume 2528 of *Lecture Notes Comput. Sci.*, pages 1–11. Springer-Verlag, 2002.

[Ber99] François Bertault. A force-directed algorithm that preserves edge crossing properties. In Jan Kratochvíl, editor, *Proc. 7th Int. Symp. Graph Drawing (GD'99)*, volume 1731 of *Lecture Notes Comput. Sci.*, pages 351–358. Springer-Verlag, 1999.

[Béz72] Pierre Bézier. *Numerical Control*. Wiley, 1972.

[BHKN09] Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for multi-criteria boundary labeling. *J. Graph Algorithms Appl.*, 13(3):289–317, 2009.

[BKNS10] Michael A. Bekos, Michael Kaufmann, Martin Nöllenburg, and Antonios Symvonis. Boundary labeling with octilinear leaders. *Algorithmica*, 57(3):436–461, 2010.

[BKPS06] Michael A. Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. Multi-stack boundary labeling problems. In S. Arun-Kumar and Naveen Garg, editors, *Proc. Int. Conf. Foundat. Software Tech.*

Theor. Comput. Sci. (FSTTCS'06), volume 4337 of *Lecture Notes Comput. Sci.*, pages 81–92. Springer-Verlag, 2006.

[BKPS10] Michael A. Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. Area-feature boundary labeling. *Comput. J.*, 53(6):827–841, 2010.

[BKPS11] Michael A. Bekos, Michael Kaufmann, Dimitrios Papadopoulos, and Antonios Symvonis. Combining traditional map labeling with boundary labeling. In Ivana Cerná, Tibor Gyimóthy, Juraj Hromkovic, Keith G. Jeffery, Rastislav Královic, Marko Vukolic, and Stefan Wolf, editors, *Proc. 37th Conf. Current Trends Theory Practice Comput. Sci. (SOFSEM'11)*, volume 6543 of *Lecture Notes Comput. Sci.*, pages 111–122. Springer-Verlag, 2011.

[BKSW07] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Comput. Geom. Theory Appl.*, 36(3):215–236, 2007.

[BLR00] Thomas Barkowsky, Longin Jan Latecki, and Kai-Florian Richter. Schematizing maps: Simplification of geographic shape by discrete curve evolution. In C. Freksa, W. Brauer, C. Habel, and K. F. Wender, editors, *Proc. Spatial Cognition II—Integrating abstract theories, empirical studies, formal models, and practical applications*, volume 1849 of *Lecture Notes in Artificial Intelligence*, pages 41–53, 2000.

[BNUW07] Marc Benkert, Martin Nöllenburg, Takeaki Uno, and Alexander Wolff. Minimizing intra-edge crossings in wiring diagrams and public transport maps. In Michael Kaufmann and Dorothea Wagner, editors, *Proc. 14th Int. Symp. Graph Drawing (GD'06)*, volume 4372 of *Lecture Notes Comput. Sci.*, pages 270–281. Springer-Verlag, 2007.

[BP13] Ulrik Brandes and Barbara Pampel. Orthogonal-ordering constraints are tough. *J. Graph Algorithms Appl.*, 17(1):1–10, 2013.

[Bra99] Ulrik Brandes. *Layout of Graph Visualizations*. PhD thesis, University of Konstanz, 1999. See http://nbn-resolving.de/urn:nbn:de:bsz:352-opus-2552.

[BST00] Ulrik Brandes, Galina Shubina, and Roberto Tamassia. Improving angular resolution in visualizations of geographic networks. In *Proc. Joint Eurographics–IEEE TCVG Symp. Visual. (VisSym'00)*, pages 23–32, 2000.

[BSTW01] Ulrik Brandes, Galina Shubina, Roberto Tamassia, and Dorothea Wagner. Fast layout methods for timetable graphs. In Joe Marks, editor, *Proc. 8th Int. Symp. Graph Drawing (GD'00)*, volume 1984 of *Lecture Notes Comput. Sci.*, pages 127–138. Springer-Verlag, 2001.

[BSV11] Kevin Buchin, Bettina Speckmann, and Kevin Verbeek. Angle-restricted Steiner arborescences for flow map layout. In Takao Asano, Shin-Ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, *Proc. 22nd Int. Symp. Algorithms Comput. (ISAAC'11)*, volume 7074 of *Lecture Notes Comput. Sci.*, pages 250–259. Springer-Verlag, 2011.

[BT04] Hans L. Bodlaender and Gerard Tel. A note on rectilinearity and angular resolution. *J. Graph Algorithms Appl.*, 8(1):89–94, 2004.

[BW00] Ulrik Brandes and Dorothea Wagner. Using graph layout to visualize train connection data. *J. Graph Algorithms Appl.*, 4(3):135–155, 2000.

[CdBvD+01] Sergio Cabello, Mark de Berg, Steven van Dijk, Marc van Kreveld, and Tycho Strijk. Schematization of road networks. In *Proc. 17th Annu. ACM Symp. Comput. Geom. (SoCG'01)*, pages 33–39, 2001.

[CEH96] Kenneth C. Cox, Stephen G. Eick, and Taosong He. 3d geographic network displays. *ACM SIGMOD Record*, 25(4):50–54, 1996.

[CZQ+08] W. Cui, H. Zhou, H. Qu, P. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Trans. Visual. Comput. Graph. (InfoVis'08)*, 14(6):1277–1284, 2008.

[DGNP10] Daniel Delling, Andreas Gemsa, Martin Nöllenburg, and Thomas Pajor. Path schematization for route sketches. In Haim Kaplan, editor, *Proc. 12th Scand. Workshop Algorithm Theory (SWAT'10)*, volume 6139 of *Lecture Notes Comput. Sci.*, pages 285–296. Springer-Verlag, 2010.

[DHM08] Tim Dwyer, Nathan Hurst, and Damian Merrick. A fast and simple heuristic for metro map path simplification. In George Bebis et al., editor, *Proc. 4th Int. Symp. Advances Visual Comput. (ISVC'08)*, volume 5359 of *Lecture Notes Comput. Sci.*, pages 22–30. Springer-Verlag, 2008.

[dNE03] Hugo A. D. do Nascimento and Peter Eades. User hints for map labelling. In *Proc. 26th Australasian Comput. Sci. Conf.*, ACM Int. Conf. Proc. Series, pages 339–347, 2003.

[DP73] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Can. Cartogr.*, 10(2):112–122, 1973.

[ELMS91] Peter Eades, Wei Lai, Kazuo Misue, and Kozo Sugiyama. Preserving the mental map of a diagram. In *Proc. Compugraphics*, pages 34–43, Sesimbra, Portugal, September 1991.

[FD01] Michael Friendly and Daniel J. Denis. Milestones in the history of thematic cartography, statistical graphics, and data visualization. Web document, http://www.datavis.ca/milestones/. Entry on the first flow map: www.datavis.ca/milestones/index.php?group=1800%2B&mid=ms113, 2001. Accessed: Sept. 2012.

[FHN+13] Martin Fink, Herman Haverkort, Martin Nöllenburg, Maxwell Roberts, Julian Schuhmann, and Alexander Wolff. Drawing metro maps using Bézier curves. In Walter Didimo and Maurizio Patrignani, editors, *Proc. 20th Int. Symp. Graph Drawing (GD'12)*, volume 7704 of *Lecture Notes Comput. Sci.*, pages 463-474. Springer-Verlag, 2013.

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software Pract. Exper.*, 21(11):1129–1164, 1991.

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

[GH10] Emden R. Gansner and Yifan Hu. Efficient, proximity-preserving node overlap removal. *J. Graph Algorithms Appl.*, 14(1):53–74, 2010.

[GHN11] Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. Boundary-labeling algorithms for panorama images. In *Proc. 19th ACM SIGSPATIAL Int. Conf. Advances Geogr. Inform. Syst. (ACM-GIS'11)*, pages 289–298, 2011.

[GMPP91] Peter Gritzmann, Bojan Mohar, János Pach, and Richard Pollack. Embedding a planar triangulation with vertices at specified positions. *Amer. Math. Mon.*, 98:165–166, 1991.

[GN00] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software Pract. Exper.*, 30:1203–1233, 2000.

[GNPR11] Andreas Gemsa, Martin Nöllenburg, Thomas Pajor, and Ignaz Rutter. On d-regular schematization of embedded paths. In Ivana Cerná, Tibor Gyimóthy, Juraj Hromkovic, Keith Jefferey, Rastislav Královic, Marko Vukolic, and Stefan Wolf, editors, *Proc. 37th Int. Conf. Current Trends Theory Practice Comput. Sci. (SOFSEM'11)*, volume 6543 of *Lecture Notes Comput. Sci.*, pages 260–271. Springer-Verlag, 2011.

[Har38] Henry Drury Harness. Atlas to accompany the second report of the railway commissioners, Ireland. Her Majesty's Stationery Office, 1838.

[HGK10] Yifan Hu, Emden R. Gansner, and Stephen Kobourov. Visualizing graphs and clusters as maps. *IEEE Comput. Graphics Appl.*, 30:54–66, 2010.

[HKV11] Yifan Hu, Stephen G. Kobourov, and Sankar Veeramoni. On maximum differential graph coloring. In Ulrik Brandes and Sabine Cornelsen, editors, *Proc. 18th Int. Symp. Graph Drawing (GD'10)*, volume 6502 of *Lecture Notes Comput. Sci.*, pages 274–286. Springer-Verlag, 2011.

[HKV12] Yifan Hu, Stephen G. Kobourov, and Sankar Veeramoni. Embedding, clustering and coloring for dynamic maps. In *Proc. 5th IEEE Pacific Visual. Symp. (PacificVis'12)*, pages 33–40, 2012.

[HMdN06] Seok-Hee Hong, Damian Merrick, and Hugo A. D. do Nascimento. Automatic visualisation of metro maps. *J. Visual Lang. Comput.*, 17(3):203–224, 2006.

[Hol95] Christine Holch. Der vielfältige Falk. Die Zeit (German newspaper), 15 December 1995.

[HS52] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *J. Research Nat. Bur. Standards*, 49(6):409–436, 1952.

[HS92] John Hershberger and Subhash Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32(2):249–267, 1992.

[HS94] John Hershberger and Jack Snoeyink. An $O(n \log n)$ implementation of the Douglas-Peucker algorithm for line simplification. In *Proc. 10th Annu. ACM Symp. Comput. Geom. (SoCG'94)*, pages 383–384, 1994.

[HS11] Jan-Henrik Haunert and Leon Sering. Drawing road networks with focus regions. *IEEE Trans. Vis. Comput. Graphics*, 17(12):2555–2562, 2011.

[HVW09] Danny Holten and Jarke J. Van Wijk. Force-directed edge bundling for graph visualization. *Comput. Graphics Forum*, 28(3):983–990, 2009.

[Jen06] Bernhard Jenny. Geometric distortion of schematic network maps. *Bulletin Soc. Cartogr.*, 40:15–18, 2006.

[Koe36] Paul Koebe. Kontaktprobleme der konformen Abbildung. *Ber. Sächs. Akad. Wiss. Leipzig, Math.-Phys. Klasse*, 88:141–164, 1936.

[KW78] Joseph B. Kruskal and Myron Wish. *Multidimensional Scaling.* Number 07-011 in Sage University Paper series on Quantitative Application in the Social Sciences. Beverly Hills and London: Sage Publications, 1978.

[Law76] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart & Winston, New York, 1976.

[Lin10] Chun-Cheng Lin. Crossing-free many-to-one boundary labeling with hyperleaders. In *Proc. IEEE Pacific Visual. Symp. (PacificVis'10)*, pages 185–192, 2010.

[LKY08] Chun-Cheng Lin, Hao-Jen Kao, and Hsu-Chun Yen. Many-to-one boundary labeling. *J. Graph Algorithms Appl.*, 12(3):319–356, 2008.

[LL99] Longin Jan Latecki and Rolf Lakämper. Convexity rule for shape decomposition based on discrete contour evolution. *Comput. Vision Image Underst.*, 73(3):441–454, 1999.

[Llo82] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inform. Theory*, 28(2):129–137, 1982.

[LPT+11] Chun-Cheng Lin, Sheung-Hung Poon, Shigeo Takahashi, Hsiang-Yun Wu, and Hsu-Chun Yen. One-and-a-half-side boundary labeling. In Weifan Wang, Xuding Zhu, and Ding-Zhu Du, editors, *Proc. 5th Int. Conf. Combin. Optim. Appl. (COCOA'11)*, volume 6831 of *Lecture Notes Comput. Sci.*, pages 387–398. Springer-Verlag, 2011.

[LS02] Ulrich Lauther and Andreas Stübinger. Generating schematic cable plans using springembedder methods. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Proc. 9th Int. Symp. Graph Drawing (GD'01)*, volume 2265 of *Lecture Notes Comput. Sci.*, pages 465–466. Springer-Verlag, 2002.

[LWY09] Chun-Cheng Lin, Hsiang-Yun Wu, and Hsu-Chun Yen. Boundary labeling in text annotation. In *Proc. 13th Int. IEEE Conf. Inform. Vis. (IV'09)*, pages 110–115, 2009.

[MG07] Damian Merrick and Joachim Gudmundsson. Path simplification for metro map layout. In Michael Kaufmann and Dorothea Wagner, editors, *Proc. 14th Int. Symp. Graph Drawing (GD'06)*, volume 4372 of *Lecture Notes Comput. Sci.*, pages 258–269. Springer-Verlag, 2007.

[MHCF96] Tamara Munzner, Eric Hoffman, K. Claffy, and Bill Fenner. Visualizing the global topology of the MBone. In *Proc. IEEE Symp. Inform. Visual. (InfoVis'96)*, pages 85–92, 1996.

[Min69] Charles Joseph Minard. Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812–1813. http://en.wikipedia.org/wiki/File:Minard.png, 1869.

[MKH11] Daisuke Mashima, Stephen G. Kobourov, and Yifan Hu. Visualizing dynamic data with maps. *IEEE Trans. Visual. Comput. Graphics*, 18(9):1424–1437, 2011.

[MNWB08] Damian Merrick, Martin Nöllenburg, Alexander Wolff, and Marc Benkert. Morphing polylines: A step towards continuous generalization. *Comput. Environ. Urban Syst.*, 32(4):248–260, 2008.

[MO01] Joseph S. B. Mitchell and Joseph O'Rourke. Computational geometry column 42. *SIGACT News*, 32(3):63–72, 2001.

[Nes04] Keith V. Nesbitt. Getting to more abstract places using the metro map metaphor. In *Proc. 8th Int. Conf. Inform. Visual. (IV'04)*, pages 488–493. IEEE Computer Society, 2004.

[New06] Mark E. J. Newman. Modularity and community structure in networks. *Proc. Natl. Acad. Sci. USA*, 103:8577–8582, 2006.

[Ney99] Gabriele Neyer. Line simplification with restricted orientations. In Frank K. Dehne, Arvind Gupta, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Proc. 6th Int. Workshop Algorithms Data Struct. (WADS'99)*, volume 1663 of *Lecture Notes Comput. Sci.*, pages 13–24. Springer-Verlag, 1999.

[Nöl05a] Martin Nöllenburg. Automated drawing of metro maps. Master's thesis, Fakultät für Informatik, Universität Karlsruhe, 2005. Available at `http://www.ubka.uni-karlsruhe.de/indexer-vvv/ira/2005/25`.

[Nöl05b] Martin Nöllenburg. Automated drawings of metro maps. Technical Report 2005-25, Fakultät für Informatik, Universität Karlsruhe, 2005. Available at `http://www.ubka.uni-karlsruhe.de/indexer-vvv/ira/2005/25`.

[NPS10] Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. Dynamic one-sided boundary labeling. In *Proc. 18th Int. ACM Symp. Advances Geogr. Inform. Syst. (ACM-GIS'10)*, pages 310–319, 2010.

[NW11] Martin Nöllenburg and Alexander Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Trans. Visual. Comput. Graphics*, 17(5):626–641, 2011.

[O'R03] O'Reilly. Open source route map. `http://www.oreilly.de/artikel/routemap.pdf`, 2003.

[Ove03] Mark Ovenden. *Metro maps of the world*. Harrow Weald: Capital Transport Publishing, 2nd edition, 2003.

[Pol06] Klemens Polatschek. Die Schönheit des Untergrundes. Frankfurter Allgemeine Sonntagszeitung 28, 16 July 2006. Available via `http://fazarchiv.faz.net`.

[PXY+05] Doantam Phan, Ling Xiao, Ron B. Yeh, Pat Hanrahan, and Terry Winograd. Flow map layout. In *Proc. IEEE Symp. Inform. Visual. (InfoVis'05)*, pages 219–224, 2005.

[RNL+11] Maxwell J. Roberts, Elizabeth J. Newton, Fabio D. Lagattolla, Simon Hughes, and Megan C. Hasler. Objective versus subjective measures of metro map usability: Investigating the benefits of breaking design rules. Available at `http://privatewww.essex.ac.uk/~mjr/underground/Roberts_Metro.pdf`, August 2011.

[Rob55] Arthur H. Robinson. The 1837 maps of henry drury harness. *Geogr. J.*, 121(4):440–450, 1955.

[Saa95] Alan Saalfeld. Map generalization as a graph drawing problem. In Roberto Tamassia and Ioannis Tollis, editors, *Proc. 3rd Int. Symp. Graph Drawing (GD'94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 444–451. Springer-Verlag, 1995.

[Saa99] Alan Saalfeld. Topologically consistent line simplification with the Douglas-Peucker algorithm. *Cartogr. Geogr. Inform. Sci.*, 26(1), 1999.

[SGSK01] Elmer S. Sandvad, Kaj Grønbæk, Lennert Sloth, and Jørgen Lindskov Knudsen. A metro map metaphor for guided tours on the Web: the Webvise Guided Tour System. In *Proc. 10th Int. World Wide Web Conf. (WWW'01)*, pages 326–333. ACM Press, 2001.

[SR05] Jonathan M. Stott and Peter Rodgers. Automatic metro map design techniques. In *Proc. 22nd Int. Cartogr. Conf. (ICC'05)*, La Coruña, Spain, 2005.

[SRMW11] Jonathan Stott, Peter Rodgers, Juan Carlos Martínez-Ovando, and Stephen G. Walker. Automatic metro map layout using multicriteria optimization. *IEEE Trans. Visual. Comput. Graphics*, 17(1):101–114, 2011.

[Syd08] `http://www.cityrail.nsw.gov.au/networkmaps/network_map.pdf`, 2008.

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

[Tob87] Waldo Tobler. Experiments in migration mapping by computer. *Amer. Cartogr.*, 14(2):155–163, 1987.

[Tuf01] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, CT, 2nd edition, 2001.

[Tut63] William T. Tutte. How to draw a graph. *Proc. London Math. Soc.*, 13(52):743–768, 1963.

[VBS11] Kevin Verbeek, Kevin Buchin, and Bettina Speckmann. Flow map layout via spiral trees. *IEEE Trans. Visual. Comput. Graphics*, 17(12):2536–2544, 2011.

[VW93] Mahes Visvalingam and J. D. Whyatt. Line generalisation by repeated elimination of points. *Cartogr. J.*, 30(1):46–51, 1993.

[WC11] Yu-Shuen Wang and Ming-Te Chi. Focus+context metro maps. *IEEE Trans. Visual. Comput. Graphics*, 17(12):2528–2535, 2011.

[Wik12] Wikipedia. Harry Beck, 2012. Accessed March 5, 2012.

[Wol07] Alexander Wolff. Drawing subway maps: A survey. *Informatik – Forschung & Entwicklung*, 22(1):23–44, 2007.

[YOT09] Daisuke Yamamoto, Shotaro Ozeki, and Naohisa Takahashi. Focus+Glue+Context: an improved fisheye approach for Web map services. In *Proc. 17th Annu. ACM Symp. Advances Geogr. Inferm. Syst. (ACM-GIS'09)*, pages 101–110, 2009.