# Fast Layout Methods for Timetable Graphs*

Ulrik Brandes[1], Galina Shubina[2], Roberto Tamassia[2], and Dorothea Wagner[1]

[1] University of Konstanz, Dept. of Computer & Information Science,
Box D 188, 78457 Konstanz, Germany.
{Ulrik.Brandes,Dorothea.Wagner}@uni-konstanz.de
[2] Brown University, Dept. of Computer Science, Center for Geometric Computing,
Providence, Rhode Island 02912-1910, USA. {gs,rt}@cs.brown.edu

**Abstract.** Timetable graphs are used to analyze transportation networks. In their visualization, vertex coordinates are fixed to preserve the underlying geography, but due to small angles and overlaps, not all edges should be represented by geodesics (straight lines or great circles).
A previously introduced algorithm represents a subset of the edges by Bézier curves, and places control points of these curves using a force-directed approach [5]. While the results are of very good quality, the running times make the approach impractical for interactive systems.
In this paper, we present a fast layout algorithm using an entirely different approach to edge routing, based on directions of control segments rather than positions of control points. We reveal an interesting theoretical connection with Tutte's barycentric layout method [18], and our computational studies show that this new approach yields satisfactory layouts even for huge timetable graphs within seconds.

## 1 Introduction

We consider timetables comprised of transportation schedules, which may be originating from, e.g., trains, flights, or product shipments. A large amount of such timetable data is provided to us by our industrial partner.[1] These are mostly train schedules from companies all over Europe, but may contain timetables from other public transport authorities (running busses, ferries, etc.) as well. Due to the size of the data (e.g., more than 140,000 trains serving a combined number of 28,000 stations for schedules of trains in Europe), visualization has proven to be a valuable tool for data inspection and maintenance.

The main purpose of the analysis of such data is quality management, since schedules may vary between time periods, and are exchanged in a variety of formats, including hand-written. The basis for several aspects of the analysis [19,

* Research partially supported by the U.S. National Science Foundation under grants CCR-9732327 and CDA-9703080, by the U.S. Army Research Office under grant DAAH04-96-1-0013, and by the German Academic Exchange Service (DAAD/Hochschulsonderprogramm III).

[1] *TLC/EVA* the IT subsidiary of *Deutsche Bahn AG* that is, among other things, responsible for collecting, analyzing, and publishing timetable information.

J. Marks (Ed.): GD 2000, LNCS 1984, pp. 127–138, 2001.

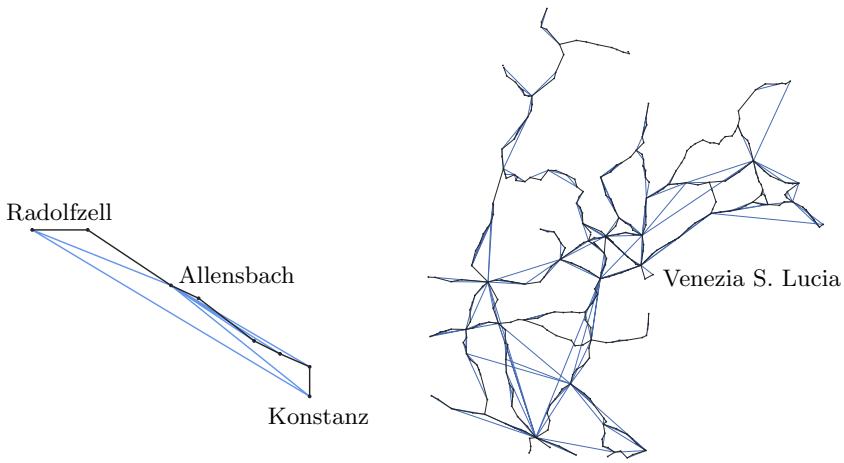



**Fig. 1.** All edges represented by straight-line segments

17,14] are graphs generated from the timetables. For the purpose of this paper, we assume that each station that any train stops at corresponds to a vertex, and an undirected edge is introduced for each pair of stations for which there is a non-stop service in either direction. Consequently, the timetable graphs considered here are undirected and simple.

The links between points of departure and arrival are tied to geographic locations, thus providing an intuitive vertex placement for the corresponding connection graph, leaving us with the problem of routing its edges. In order for the visualization to be effective, the edge routing algorithm has to produce clear and helpful drawings, and do so quickly to make it usable in interactive tools.

In the most common form of geographic network visualization, edges are shown as geodesics (straight lines or great circles, depending on whether the graph is shown in the plane or on the sphere) [2,16]. While such drawings can be produced very quickly, small angles and overlap of edges often hinder their unambiguous identification, as can be seen in Fig. 1.

A method to produce effective timetable graph visualizations is presented in [5]. It uses an automatically classification of the edges into *minimal* and *transitive*, where minimal edges are assumed to correspond to railroads directly connecting pairs of stations, while transitive edges typically correspond to regional or long-distance services that do not stop at each station they pass. Since railroads can be expected to cover a geographic region efficiently, this method represents minimal edges by straight lines. Transitive edges, by their very nature, are bound to cause small angles and overlap. Therefore they are represented by cubic Bézier curves. An elaborate force-directed model places the control points of these curves. While according to the data analysts, the output is very satisfactory, running times are not acceptable (in the range of several minutes for graphs of realistic size). See Fig. 2 for an example.

Since we plan to integrate visualization into an existing interactive query engine that allows to generate timetable graphs from the complete data set based

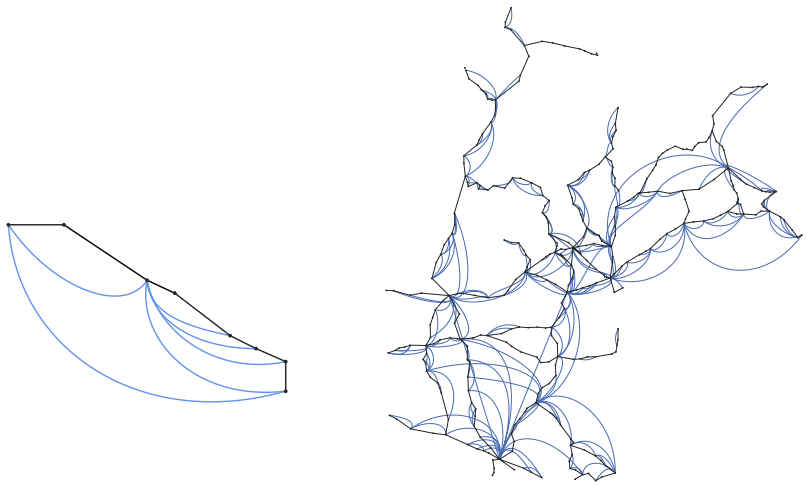


**Fig. 2.** Force-directed placement of Bézier control points for transitive edges [5]

on a variety of attributes (coordinates, train classes, traveling times, service frequencies, etc.), faster, yet still effective, layout methods are sought.

A recently introduced alternative approach to automatic routing of cubic Bézier curves is based on the angles between control segments rather than the positions of control points (*rotation approach* [4]). Though it is extremely fast, its application to timetable graphs produces drawings with several deficiencies, such as excessive crossings and S-shaped curves.

Building on the underlying principle of the general rotation approach, we present a new edge routing model that yields better layouts without sacrificing too much of the running-time advantage. It uses properties of timetables to preprocess the graph so as to make it more susceptible to a rotation-like method. A new objective function is introduced, that combines three criteria: *angular resolution*, *straightness*, and *roundness*. A theoretically interesting aspect of this model is its close relation to the barycentric model of Tutte [18], even though it is entirely based on angles rather than coordinates.

Though several other graph drawing techniques explicitly consider curved or polyline edges, none of them seems applicable in our case, because they either position vertices [12,13,7] or are mainly designed to route edges around obstacles [11,9,1,10]. Our goal here is to disentangle a straight-line drawing in the simplest way possible without moving vertices.

The main features of the angle-based approach for cubic-curve routing are outlined in Sect. 2. Our new model for timetable graphs is introduced in Sect. 3. We conclude with some real-world examples and running time experiments.

## 2 An Alternative Approach to Curved Edge Layout

In this section, we outline some aspects of a recently introduced approach [4] for edge layout, when a graph has fixed vertex positions.

A cubic Bézier curve [3] is determined by its two endpoints, $b_0, b_3$, and two inner *control points* $b_1, b_2$ (see Fig. 3). Note that the same curves are obtained when the order of control points is reversed, while other permutations in general define different curves. We thus call segments $\overline{b_0 b_1}$ and $\overline{b_2 b_3}$ the *initial control segments*, while $\overline{b_1 b_2}$ is called the *inner control segment*. Two other important properties of Bézier curves are *i)* that the entire curve is contained in the convex hull of its defining points, and *ii)* that the tangents at its endpoints are collinear with the first and last control segment. The second property provides an immediate generalization of angles between edges represented by straight lines to angles between edges represented by Bézier curves. From now on, we will neglect the distinction between an edge or a vertex and its graphical representation.

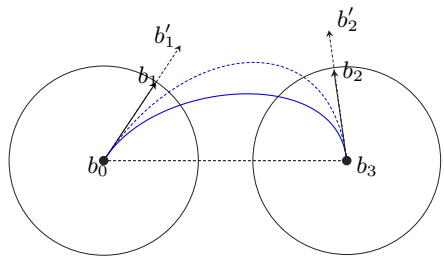



**Fig. 3.** Cubic Bézier curves [3] defined by rotation and length of initial segments

Since vertex positions are fixed, so are the endpoints of each edge. Instead of placing control points directly, the layout can be divided into the following two steps, also illustrated by Fig. 3:

1. determine a direction for each initial control segment
2. determine a length for each initial control segment

The (*local*) *angular resolution* at some vertex is defined as the smallest angle formed by the edges incident to that vertex. The *global* angular resolution of a drawing is the minimum local angular resolution. In the first of the above steps, the angles between incident edges are determined, and therefore the angular resolution at all vertices. The second step can be used to ensure additional properties of a curve, e.g. to reduce its curvature. For our purposes, however, the simple heuristic of choosing a fixed proportion of the distance between the endpoints proved sufficient.

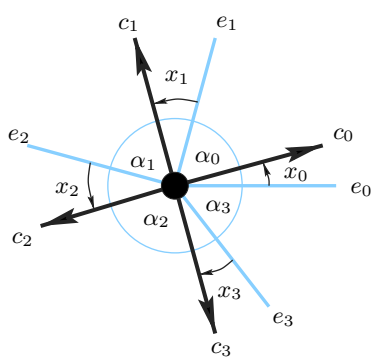



**Fig. 4.** Angles $\alpha_i$ between straight-line edges, and angular differences $x_i$

Cubic curves are the simplest representation that allows to maximize the angular resolution for all vertices. Note that we can treat control segments incident to one vertex independently from control segments incident to other vertices. Therefore, let $e_0, \ldots, e_{d_G(v)-1}$, be a counterclockwise ordering of the edges around some $v \in V$ in the given drawing (with ties broken arbitrarily), and denote by $\alpha_i$, $i = 0, \ldots, d_G(v)-1$, the angle between $e_i$ and its counterclockwise neighbor.

Accordingly, we define $c_0, \ldots, c_{d_G(v)-1}$ to be the corresponding ordering of control segments incident to $v$. The angles between neighboring control segments equal $\frac{2\pi}{d_G(v)}$ because of the optimal angular resolution constraint. Denote by $x_i$ the angle between $e_i$ and $c_i$, $i = 0, \ldots, d_G(v) - 1$, where $x_i > 0$, if $e_i$ comes before, and $x_i < 0$, if $e_i$ comes after $c_i$ in the counterclockwise order around $v$. We call these deviations from straight-line directions the *angular errors*. See Fig. 4 for an illustration and note that $x_3 < 0$.

Any set of control segments satisfying the angle constraints is called a *rotation* at $v$, but arbitrary rotations usually lead to unpleasant "spaghetti" drawings. The *straightness* of a rotation is the degree to which a rotation succeeds in keeping the angular errors small. Several penalty functions can be used to quantify straightness. In particular, the rotation minimizing the squared angular errors $\sum_{i=0}^{d_G(v)-1}(x_i)^2$ is unique, and can be computed in time $\mathcal{O}(d_G(v))$ simply by averaging over the angular errors. It is called *balanced rotation*, because it satisfies $\sum_{i=0}^{d_G(v)-1} x_i = 0$. Figure 5 shows that optimal local angular resolution at all vertices is too strong a criterion. Note, however, that angles between control segments can be specified arbitrarily. Much better drawings are obtained, when angles between control segments at vertices $v$ with a *dominant angle* angle $\alpha_i \geq \pi$ are constrained to $\pi$ (for the dominant angle) and to $\frac{\pi}{d_G(v)-1}$ for the other angles. This set of angle constraints is called the *half-sided template*. For further details we refer to [4].

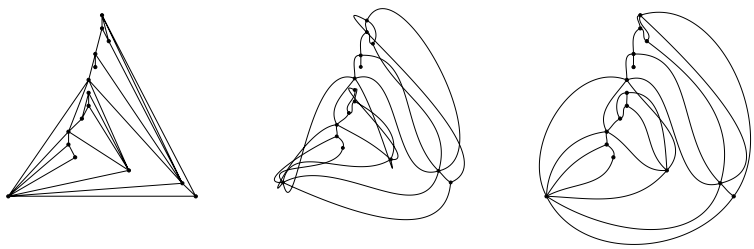


**Fig. 5.** Result of a standard planar graph drawing algorithm [8], and balanced rotations with and without half-sided templates for dominant angles

The rotation approach applied to timetable graphs is very fast (see Sect. 4). Though an obvious improvement over straight-line drawings, the examples in Fig. 6 also show that the results are not entirely satisfactory for our present application. Despite optimal angular resolution and straightness, the drawings in general display sharp turns, S-shaped curves and appear cluttered.

## 3 A New Layout Model for Timetable Graphs

Though the rotation approach described in the previous section yields drawings that are much more readable than straight-line representations, they are still far from the quality obtained by the force-directed layout approach. In this section, we present a new layout model based on the same strategy as the rotation approach but tailored to timetable graphs.

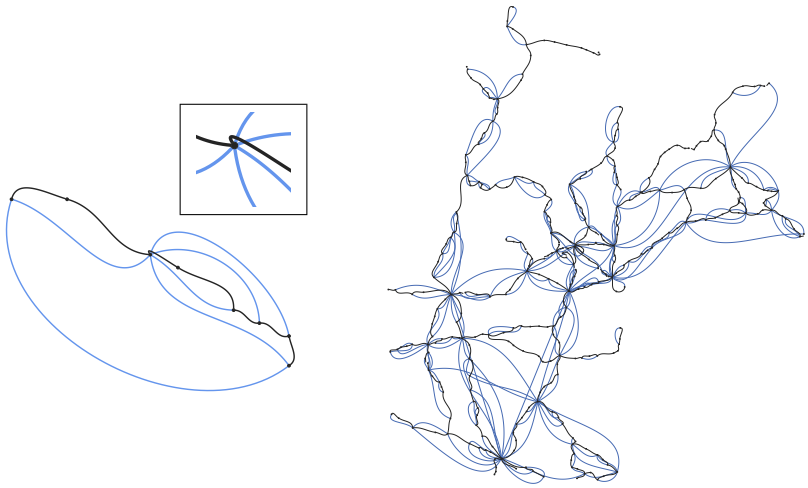


**Fig. 6.** Balanced rotations with the half-sided template for vertices with a dominant angle. Note the unfortunate ordering on the left.

Edges of a timetable graph are routinely classified into minimal and transitive. Since minimal edges typically correspond to actual railways, they seldom cause readability problems and can hence be drawn straight, both to reduce the size of the input and to visually emphasize their role as a support of the network.

We are going to customize the rotation approach to our specific application in several ways. Most importantly, we introduce a preprocessing step that determines a good ordering of edges around a vertex, and our new objective function aims at rounding out S-shaped curves. In consequence, angular resolution must be relaxed from a constraint to an optimization criterion. We show that there still is a unique optimum solution, which, however, is no longer computed as easily as in the rotation approach.

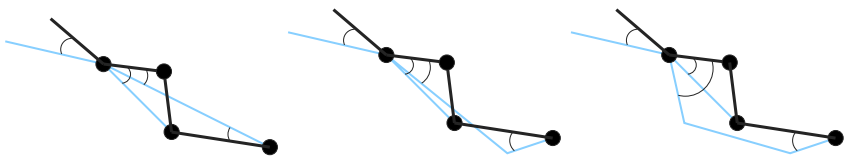
### 3.1 Preprocessing

Since minimal edges are represented by straight lines, the ordering of minimal edges around a vertex is fixed. Transitive edges have an inherent tendency to short-cut paths of minimal edges. Small deviations in the course of the latter cause the straight-line ordering of transitive edges to be somewhat arbitrary (as was demonstrated in Fig. 6). To reduce the crossing problem illustrated in Fig. 6, an ordering of transitive edges is determined in two stages: in the first stage, control segments with similar target directions are grouped together, and in the second stage, the control segments of each group are put in order.

*Grouping.* The initial control segments incident to a vertex are grouped according to the minimal edge their straight-line representation is closest to, and according to the side toward which they depart (i.e. whether they are counterclockwise before or after their closest minimal edge). As illustrated in Fig. 7(a),

each initial control segment of a transitive edge is assigned to the left or right hand side of its closest minimal edge.

*Crossing reduction.* The assignment is then refined to reduce the number of crossings among transitive and minimal edges. If the two initial control segments of a single edge are both classified to lie on, say, the right hand side of their respective minimal edge, the assumption that these minimal edges are linked by a path of minimal edges (the railroad that the long-distance trains inducing this edge travel along) suggests that the inner control segment of the transitive edge will cross this path. We therefore reassign the control segment incident to the vertex of smaller degree (presumably a less important station), or, in case of equal degrees, to the one that deviates further from the straight-line connecting the endpoints, to another group as depicted in Fig. 7(b).



(a) group by closest minimal edge

(b) regroup to avoid crossings

(c) sort by length

**Fig. 7.** Initial ordering of edges. Arcs indicate the side of minimal edge a control segment is assigned to

*Sorting.* Within a group, control segments are sorted according to the length of the straight-line representation of their corresponding edge, such that the shortest ones are closest to their assigned minimal edge. See Fig. 7(c). This order is likely to avoid crossing of adjacent transitive edges.

Thus the initial control segments of transitive edges incident to a common vertex are grouped into a number of groups that is twice the number of minimal edges incident to that vertex. Each group is assigned a wedge that is an angle between a minimal edge and a bisector of the angle between this edge and its clockwise or counterclockwise minimal neighbor and space out evenly. The overall running time of the preprocessing step is $\mathcal{O}\left(\sum_{v \in V} d_G(V) \log d_G(v)\right) = \mathcal{O}\left(|E| \log |E|\right)$.

### 3.2 Layout Objectives

With this heuristic ordering, we are now able to state our objective function formally. It combines the criteria of local angular resolution, straightness, and roundness, subject to straight-line representation of minimal edges. Considering the straight minimal edges to be control segments with fixed direction, let $c_0, \ldots, c_{d_G(v)-1}$ be the directions of control segments incident to a vertex $v$, in the order resulting from the preprocessing.

*Angular Resolution.* The optimal angular differences $a_i$, $i = 0, \ldots, a_{d_G(v)-1}$, between consecutive control segments (which have been a constraint in the rotation approach) are determined by equally dividing up the wedge assigned to the group. Satisfaction of the angular resolution criterion can then be expressed in terms of the squared angular error with respect to the target values,

$$A_v(c) = \sum_{i=0}^{d_G(v)-1} (c_{i+1} - c_i - a_i)^2, \qquad (\textit{angular resolution criterion})$$

where indices are modulo $d_G(v)$, and pairs $c_i, c_{i+1}$ of control segments in different groups are omitted. Recall that control segments of minimal edges lie in two groups.

*Straightness.* For reasons mentioned in the discussion of the rotation approach, the deviation of control segments from straight edges should be penalized. We use the squared angular errors with respect to straight-line directions, i.e. the objective function of the balanced rotation approach,

$$S_v(c) = \sum_{i=0}^{d_G(v)-1} x_i^2 = \sum_{i=0}^{d_G(v)-1} (c_i - e_i)^2. \qquad (\textit{straightness criterion})$$

*Roundness.* Much of the clarity in timetable graph layouts produced with force-directed placement stems from the prevailing symmetry, or *roundness*, of those edges represented by Bézier curves. We measure the roundness of a Bézier curve by the squared difference in deviation of the two initial control segments from the straight-line edges connecting the endpoints. Note that, for a highly desirable curve, the magnitudes of deviation are the same at both ends, but with opposite sign.

As a result of the preprocessing, one initial control segment of a transitive edge is assigned to a group associated with the right hand side of a minimal edge, while the other is assigned to a group associated with the left hand side of a minimal edge. Reversing the sense of direction within each group associated with, say, a left hand side changes the sign of all their angular differences, but does not affect the other two criteria (if the sign of optimal angular differences $a_i$ is reversed as well). Non-roundness is thus defined as

$$R_v(c) = \sum_{i=0}^{d_G(v)-1} \left((c_i - e_i) - (c'_i - e'_i)\right)^2, \qquad (\textit{roundness criterion})$$

where $c'_i$ is the direction of the initial control segment at the opposite end of $e_i$, and $e'_i$ is the reverse straight-line direction of the edge.

The objective function for edge layout of preprocessed timetable graphs is now defined as a weighted sum of the above criteria, $U(c) = \sum_{v \in V} \omega_a \cdot A_v(c) + \omega_s \cdot S_v(c) + \omega_r \cdot R_v(c)$.

### 3.3 Optimizing the Objective Function

Next we show that the layout objective function $U(c)$ defined in the previous section is a generalized version of the objective function of the barycentric layout model, and has a unique minimum under equivalent assumptions.

Consider the following transformation of a preprocessed timetable graph. We construct a new undirected graph $G = (V, E)$ that has a vertex for each of the two initial control segments, and for each of the two directions of a transitive edge. Two vertices in $G$ are adjacent, if one is a straight-line direction and the other is the corresponding control segment ($e_i$ and $c_i$ of some vertex), if they are consecutive control segments in some group (recall that groups and order are defined in the preprocessing), or if they are initial control segments of the same transitive edge.

Assume that for each edge $e = \{u, v\} \in E$ there are weights $\omega_e > 0$ and target differences $\theta_{uv} = -\theta_{vu}$. Then it is easy to see that our objective function can be restated as

$$U(c) = \sum_{e=\{u,v\}\in E} \omega_e \cdot (c_v - c_u - \theta_{uv})^2.$$

Since the essential properties of this function are the same as those of the barycentric layout model $\sum_{e=\{u,v\}\in E}(c_v - c_u)^2$, the following parallels Tutte's analysis [18]. For a vector $c = (c_v)_{v\in V}$ minimizing this function, the partial derivatives

$$\frac{\partial}{\partial c_v} U(c) = \sum_{u\,:\,e=\{u,v\}\in E} 2\omega_e \cdot (c_v - c_u - \theta_{uv})$$

must equal zero for all $v \in V$. Cancelling the constant factor of 2, this system of linear equations can be reordered into the form

$$(D(G) - A(G)) \cdot c = L(G) \cdot c = b,$$

where $D(G)$ is a diagonal matrix with weighted degrees $d_{vv} = \sum_{u\,:\,e=\{u,v\}\in E} \omega_e$ on the diagonal, $A(G)$ is the weighted adjacency matrix with entries $a_{uv} = \omega_{\{u,v\}}$ if $\{u, v\} \in E$ and $a_{uv} = 0$ otherwise, and $b$ is a vector with constant entries $b_v = \sum_{u\,:\,e=\{u,v\}\in E} \omega_e \theta_{uv}$. The resulting matrix $L(G)$ is called the Laplacian of the graph.

**Lemma 1 ([6]).** *The determinant of any submatrix of $L(G)$ obtained by omitting any pair of a row and a column corresponding to a vertex in $G$ equals*

$$\sum_{T} \prod_{e\in E(T)} \omega_e$$

*where the sum is over all spanning trees of $G$, and $E(T)$ denotes the edge set of a tree $T$.*

Note that fixing any entry in $c$ corresponds to omitting its row and column from $L(G)$ and adjusting $b$. Fixing the entries of more than one vertex of $G$ corresponds to contracting these vertices, and then omitting the row and column of this vertex and adjusting $b$. Consequently, the determinant of the resulting submatrix is positive, if the value of at least one vertex in each connected component of $G$ is fixed. Since, by definition, no station is incident only to transitive edges, every component of $G$ has at least one vertex that corresponds to a fixed control segment of a minimal edge.

**Theorem 1.** *The timetable graph layout objective function $U(c)$ has a unique minimum that can be determined by solving a system of linear equations with twice as many unknowns as there are transitive edges.*

Due to the size of typical systems (cf. Tab. 1), we cannot afford to solve it exactly in time still acceptable for an interactive system. Since the matrix $L(G)$ is weakly diagonally dominant, we instead use Gauss-Seidel iteration to quickly approximate the optimal directions. Note that this nicely corresponds to a one-dimensional spring embedder, that does an optimal move at each step.

Initial directions are determined by equally dividing the angle formed by the bounding pair of a minimal edge and an angle bisector for each group. Clearly, these layouts optimize the angular resolution criterion subject to the heuristic ordering and grouping constraint.

## 4 Results and Discussion

Figure 8 gives the result of our new approach as applied to the running example. The larger examples given in the appendix show that our new method clearly outperforms the general rotation approach in terms of visual quality, though it still does not quite match the quality of force-directed placements.

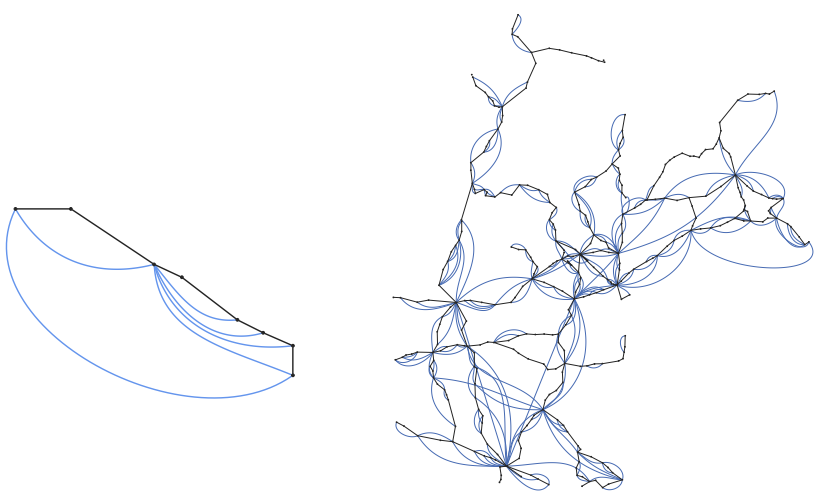


**Fig. 8.** Layout after 7 iterations ($\omega_a = 2$, $\omega_s = 0$, $\omega_r = 1$)

By necessity, the angular error of control segments with respect to straight edges is usually much larger than the angle between neighboring control segments, so that $\omega_s$ should be chosen significantly smaller than the other two weights. Since it turns out that straightness is sufficiently taken care of by the preprocessing step, we generally omit this criterion altogether. The relative choice of angular resolution vs. roundness depends on personal preferences. The examples in the appendix use $\omega_a = 2 \cdot \omega_r$. We also found that the initial directions do fairly well for any reasonable choice of weights, and since the system is rather sparse, the maximum rotation of a control segment is below 0.01 radians after 3–10 iterations.

All three approaches (force-directed layout, rotation approach, and the approach of this paper) have been implemented in C++ using LEDA [15]. Since we compare proof-of-concept implementations, our running-time experiments should be understood as qualitative. The indication is nevertheless quite clear. Though most of the time is spent on a preprocessing step that determines the "neighborhood" of each control point [5], the force-directed approach is very slow, and will probably remain so even with a sophisticated implementation. While the rotation approach is the fastest, even our current implementation of the approach presented in this paper performs at interactive speed,[2] but produces drawings of much better quality.

**Table 1.** Running times on a Sun Ultra 5 workstation (360 Mhz, 192 MBytes). Times given in parentheses are without preprocessing

| instance | nodes | edges (transitive) | force-directed | rotation | new |
|---|---|---|---|---|---|
| switzerland | 2218 | 3203 ( 536) | 53 (10) sec | 0.36 sec | **1.31 (0.17) sec** |
| italy | 2386 | 4370 (1849) | 309 (42) sec | 0.51 sec | **2.21 (0.57) sec** |
| france | 4551 | 7793 (2408) | 621 (54) sec | 0.80 sec | **3.44 (0.73) sec** |
| germany | 7083 | 9713 (1956) | 582 (38) sec | 1.18 sec | **4.21 (0.60) sec** |

There are several avenues for future work. With respect to the present application, we have yet no way of modeling the second most effective feature after roundness, i.e. *binding*. By introducing dummy edges between a pair of control points when their initial segments are incident to the same vertex, the force-directed approach succeeds in dragging consecutive or nested transitive edges to the same side of a path of minimal edges. Is there a way to integrate this feature in the present approach?

Similar problems are encountered for geographic networks whose vertices are placed on a globe. We are working on a three-dimensional interpretation in the mold of [4] that would take into account lengths of geodesics to better untangle the edges.

We are also investigating useful strategies for control segment length assignments that would satisfy certain properties of the resulting curves, like low curvature, but potentially also to preserve certain features, like planarity.

We devised a fast and effective layout method for timetable graphs by taking a different view on edge routing and utilizing the underlying network structure.

[2] Note that the runnings times compare favorably with the time that LEDA's graph editor needs to render the results.

It will be interesting to devise similar extensions of the rotation approach for other applications like internet traffic, flight routes, or non-geographic networks.

**Acknowledgment.** We thank Michael Baur and Marc Benkert for implementing the preprocessing step, and Stina Bridgeman for help in additional experiments.

## References

1. J. Abello and E.R. Gansner. Short and smooth polygonal paths. *Proc. LATIN '98*, Springer LNCS 1380, pp. 151–162, 1998.
2. R.A. Becker, S.G. Eick, and A.R. Wilks. Visualizing network data. *IEEE Transactions on Visualization and Graphics*, 1(1):16–28, 1995.
3. P. Bézier. *Numerical Control*. John Wiley & Sons, 1972.
4. U. Brandes, G. Shubina, and R. Tamassia. Improving angular resolution in visualizations of geographic networks. *Data Visualization 2000. Proc. VisSym '00*, pp. 23–32. Springer, 2000.
5. U. Brandes and D. Wagner. Using graph layout to visualize train interconnection data. *Journal of Graph Algorithms and Applications*, 2000. To appear.
6. R.L. Brooks, C.A.B. Smith, A.H. Stone, and W.T. Tutte. The dissection of rectangles into squares. *Duke Mathematical Journal*, 7:312–340, 1940.
7. C.C. Cheng, C.A. Duncan, M.T. Goodrich, and S.G. Kobourov. Drawing planar graphs with circular arcs. *Proc. GD '99*, Springer LNCS 1731, pp. 117–126, 1999.
8. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
9. D.P. Dobkin, E.R. Gansner, E. Koutsofios, and S.C. North. Implementing a general-purpose edge router. *Proc. GD '97*, Springer LNCS 1353, pp. 262–271, 1997.
10. E.R. Gansner and S.C. North. Improved force-directed layouts. *Proc. GD '98*, Springer LNCS 1547, pp. 364–373, 1998.
11. E.R. Gansner, S.C. North, and K.-P. Vo. DAG – A program that draws directed graphs. *Software—Practice and Experience*, 17(1):1047–1062, 1988.
12. M.T. Goodrich and C.G. Wagner. A framework for drawing planar graphs with curves and polylines. *Proc. GD '98*, Springer LNCS 1547, pp. 153–166, 1998.
13. C. Gutwenger and P. Mutzel. Planar polyline drawings with good angular resolution. *Proc. GD '98*, Springer LNCS 1547, pp. 167–182, 1998.
14. A. Liebers, D. Wagner, and K. Weihe. On the hardness of recognizing bundels in time table graphs. *Proc. WG '99*, Springer LNCS 1665, pp. 325–337, 1999.
15. K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
16. T. Munzner, E. Hoffman, K. Claffy, and B. Fenner. Visualizing the global topology of the MBone. *Proc. IEEE InfoVis '96*, pp. 85–92, 1996.
17. F. Schulz, D. Wagner, and K. Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *Proc. WAE '99*, Springer LNCS 1668, pp. 110–123, 1990.
18. W.T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society, Third Series*, 13:743–768, 1963.
19. K. Weihe. Covering trains by stations or the power of data reduction. Electronic *Proc. ALEX '98*, pp. 1–8, 1998.
`http://rtm.science.unitn.it/alex98/proceedings.html`.