Travel Planning With Self-Made Maps*

Ulrik Brandes, Frank Schulz, Dorothea Wagner, and Thomas Willhalm

University of Konstanz, Department of Computer & Information Science
Box D 188, 78457 Konstanz, Germany.
Firstname.Lastname@uni-konstanz.de

Abstract. Speed-up techniques that exploit given node coordinates have proven useful for shortest-path computations in transportation networks and geographic information systems. To facilitate the use of such techniques when coordinates are missing from some, or even all, of the nodes in a network we generate artificial coordinates using methods from graph drawing. Experiments on a large set of German train timetables indicate that the speed-up achieved with coordinates from our network drawings is close to that achieved with the actual coordinates.

1 Introduction

In travel-planning systems, shortest-path computations are essential for answering connection queries. While still computing the optimal paths, heuristic speed-up techniques tailored to geographic networks have been shown to reduce response times considerably [20, 19] and are, in fact, used in many such systems.

The problem we consider has been posed by an industrial partner[1] who is a leading provider of travel planning services for public transportation. They are faced with the fact that quite often, much of the underlying geography, i.e. the location of nodes in a network, is unknown, since not all transport authorities provide this information to travel service providers or competitors. Coordinate information is costly to obtain and to maintain, but since the reduction in query response time is important, other ways to make the successful geometric speed-up heuristics applicable are sought.

The existing, yet unknown, underlying geography is reflected in part by travel times, which in turn are given in the form of timetables. Therefore, we can construct a simple undirected weighted graph in the following way. Each station represents a vertex, and two vertices are adjacent, if there is a non-stop connection between the two corresponding stations. Edge weights are determined from travel times, thus representing our distance estimates. Reasonable (relative) location estimates are then obtained by embedding this graph in the plane such that edge lengths are approximately preserved.

* Research partially supported by the Deutsche Forschungsgemeinschaft (DFG) under grant WA 654/10-4.

[1] HaCon Ingenieurgesellschaft mbh, Hannover.

Our specific scenario and geometric speed-up heuristics for shortest-path computations are reviewed briefly in Sect. 2. In Sect. 3, we consider the special case in which the locations of a few stations are known and show that a simple and efficient graph drawing technique yields excellent substitutes for the real coordinates. This approach is refined in Sect. 4 to be applicable in more general situations. In Sect. 5, both approaches are experimentally evaluated on timetables from the German public train network using a snapshot of half a million connection queries.

2 Preliminaries

Travel information systems for, e.g., car navigation [21, 12] or public transport [17, 18, 22], often make use of geometric speed-up techniques for shortest-path computations. We consider the (simplified) scenario of a travel information system for public railroad transport used in a recent pilot study [19]. It is based solely on *timetables*; for each train there is one table, which contains the departure and arrival times of that train at each of its halts. In particular, we assume that every train operates daily.

The system evaluates *connection queries* of the following kind: Given a departure station A, a destination station B, and an earliest departure time, find a connection from A to B with the minimum travel time (i.e., the difference between the arrival time at B and the departure time at A).

To this end, a (directed) *timetable graph* is constructed from timetables in a preprocessing step. For each departure and arrival of a train there is one vertex in the graph. So, each vertex is naturally associated with a station, and with a time label (the time the departure or arrival of the train takes place). There are two different kinds of edges in the graph:

- *stay edges:* The vertices associated with the same station are ordered according to their time label. Then, there is a directed edge from every vertex to its successor (for the last vertex there is an edge to the first vertex). Each of these edges represents a stay at the station, and the edge length is defined by the duration of that stay.
- *travel edges:* For every departure of a train there is a directed edge to the very next arrival of that train. Here, the edge length is defined to be the time difference between arrival and departure.

Answering a connection query now amounts to finding a shortest path from a source to one out of several target vertices: The source vertex is the first vertex at the start station representing a departure that takes place not earlier than the earliest departure time, and each vertex at the destination station is a feasible target vertex.

2.1 Geometric speed-up techniques

In [19], Dijkstra's algorithm is used as a basis for these shortest-path computations and several speed-up techniques are investigated. We focus on the purely geometric ones, i.e. those based directly on the coordinates of the stations, which can be combined with other techniques.

Goal-directed search. This strategy is found in many textbooks (e.g., see [1, 16]). For every vertex v, a lower bound $b(v)$ satisfying a certain consistency criterion is required for the length of a shortest path to the target. In a timetable graph, a suitable lower bound can be obtained by dividing the Euclidean distance to the target by the maximum speed of the fastest train. Using these lower bounds, the length $\lambda_{\{u,v\}}$ of each edge is modified to $\lambda'_{\{u,v\}} = \lambda_{\{u,v\}} - b(u) + b(v)$. It can be shown that a shortest path in the original graph is a shortest path in the graph with the modified edge lengths, and vice versa. If Dijkstra's algorithm is applied to the modified graph, the search will be directed towards a correct target.

Angle restriction. In contrast to the goal-directed search, this technique requires a preprocessing step, which has to be carried out once for the timetable graph and is independent of the subsequent queries. For every vertex v representing the departure of a train, a circle sector $C(v)$ with origin at the location of the vertex is computed. That circle sector is stored using its two bounding angles, and has the following interpretation: If a station A is *not* inside the circle sector $C(v)$, then there is a shortest path from v to A, which starts with the outgoing stay edge.

Hence, if Dijkstra's algorithm is applied to compute a shortest path to some destination station D, if some vertex u is processed, and D is not inside the circle sector $C(u)$, then the outgoing travel edge can be ignored, because there is a shortest path from u to D starting with the stay edge.

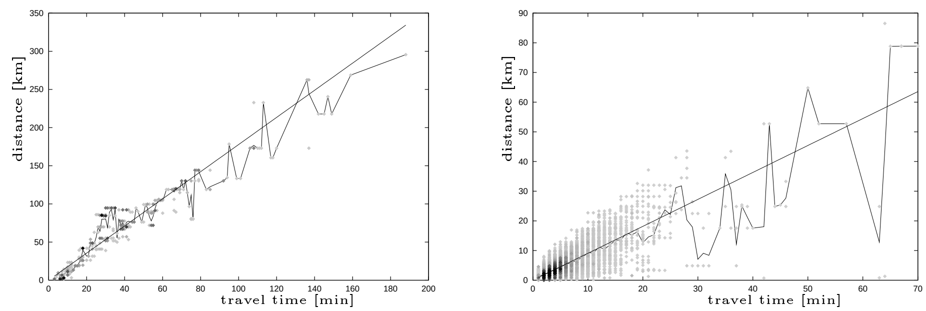
2.2 Estimating distances from travel times

The location of stations is needed to determine lower bounds for goal-directed search, or circle sectors for the angle-restriction heuristic. If the actual geographic locations are not provided, the only related information available from the timetables are travel times. We use them to estimate distances between stations that have a non-stop connection, which in turn are used to generate locations suitable for the geometric heuristics, though in general far from being geographically accurate.

The (undirected, simple) *station graph* of a set of timetables contains a vertex for each station listed, and an edge between every pair of stations connected by a train not stopping inbetween. The *length* λ_e of an edge e in the station graph will represent our estimate of the distance between its endpoints.

Distance between two stations can be expected to be roughly a linear function in the travel time. However, for different classes of trains the constant involved will be different, and closely related to mean velocity of trains in this class. We therefore estimate the length of an edge e in the station graph, i.e. the distance between two stations, using the mean over all non-stop connections inducing this edge of their travel time times the mean velocity of the vehicle serving them.

Mean velocities have been extracted from the data set described in Sect. 5, for which station coordinates are known. For two train categories, the data are depicted in Fig. 1, indicating that no other function is obviously better than our simple linear approximation. Note that all travel times are integer, since they are computed from arrival and departure times. As a consequence, slow trains are often estimated to have unrealistically high maximum velocities, thus affecting the modified edge lengths in the goal-directed search heuristic.



(a) high-speed trains

(b) local trains

Fig. 1. Euclidean distance vs. travel time for non-stop connections. For both service categories, all data points are shown along with the average distance per travel time and a linear interpolation

3 Networks with Partially Known Geography

In our particular application, it may occasionally be the case that the geographic locations of at least some of the major hubs of the network are known, or can be obtained easily. We therefore first describe a simple method to generate coordinates for the other stations that exploits the fact that such hubs are typically well-distributed and thus form a scaffold for the overall network. Our approach for the more general case, described in the next section, can be viewed as an extension of this method.

Let $p = (p_v)_{v \in V}$ be a vector of vertex positions, then the potential function

$$U_B(p) = \sum_{\{u,v\} \in E} \omega_{\{u,v\}} \cdot \|p_u - p_v\|^2 \tag{1}$$

where $\omega_e = \frac{1}{\lambda_e}$, $e \in E$, weights the influence of an edge according to its estimated length λ_e, defines a weighted *barycentric layout model* [25]. This model has an interesting physical analogy, since each of the terms in (1) can be interpreted as the potential energy of a spring with spring constant ω_e and ideal length zero.

A necessary condition for a local minimum of $U_B(p)$ is that all partial derivatives vanish. That is, for all $p_v = (x_v, y_v)$, $v \in V$, we have

$$x_v = \frac{1}{\sum\limits_{u:\{u,v\}\in E} \omega_{\{u,v\}}} \cdot \sum_{u:\{u,v\}\in E} \omega_{\{u,v\}} \cdot x_u$$

$$y_v = \frac{1}{\sum\limits_{u:\{u,v\}\in E} \omega_{\{u,v\}}} \cdot \sum_{u:\{u,v\}\in E} \omega_{\{u,v\}} \cdot y_u .$$

In other words, each vertex must be positioned in the weighted barycenter of its neighbors. It is well known that this system of linear equations has a unique solution, if at least one p_v in each connected component of G is given (and the equations induced by v are omitted) [3]. Note that, in the physical analogy, this corresponds to fixing some of the points in the spring system. Moreover, the matrix corresponding to this system of equations is weakly diagonally dominant, so that iterative equation solvers can be used to approximate a solution quickly (see, e.g., [10]).

Assuming that the given set of vertex positions provides the cornerstones necessary to unfold the network appropriately, we can thus generate coordinates for the other vertices using, e.g., Gauss-Seidel iteration, i.e. by iteratively placing them at the weighted barycenter of their neighbors. Figure 2 indicates that this approach is highly dependent on the set of given positions. As is discussed in Sect. 5, it nevertheless has some practical merits.

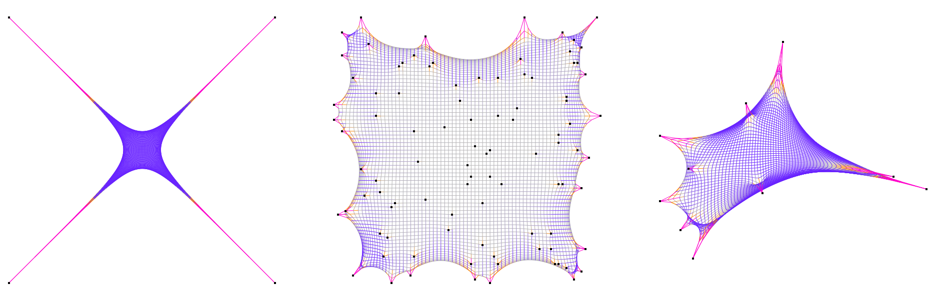


Fig. 2. Barycentric layout of an 72×72 grid with the four corners fixed, and the same grid with 95 and with 10 randomly selected vertices fixed

4 A Specific Layout Model for Connection Networks

The main drawbacks of the barycentric approach are that all vertices are positioned inside of the convex hull of the vertices with given positions, and that the estimated distances are not preserved. In this section, we modify the potential (1) to take these estimates into account.

Recall that each of the terms in the barycentric model corresponds to the potential energy of a spring of length zero between pairs of adjacent vertices. Kamada and Kawai [13] use springs of length $\lambda_{\{u,v\}} = d_G(u, v)$, i.e. equal to the length of a shortest path between u and v, between every pair of vertices. The potential then becomes

$$U_{KK}(p) = \sum_{u,v \in V} \omega_{\{u,v\}} \cdot \left(\|p_u - p_v\| - \lambda_{\{u,v\}} \right)^2 , \qquad (2)$$

the idea being that constituent edges of a shortest path in the graph should form a straight line in the drawing of the graph. To preserve local structure, spring constants are chosen as $\omega_e = \frac{1}{\lambda_e^2}$, so that long springs are more flexible than short ones. (The longer a path in the graph, the less likely are we able to represent it straight.) Note that this is a special case of multidimensional scaling, where the input matrix contains all pairwise distances in the graph.

This model certainly does reflect our layout objectives more precisely. Note, however, that it is $\mathcal{NP}$-hard to determine whether a graph has an embedding with given edge lengths, even for planar graphs [7]. In contrast to the barycentric model, the necessary condition of vanishing partial derivatives leads to a system of non-linear equations, with dependencies between x- and y-coordinates. Therefore, we can no longer iteratively position vertices optimally with respect to the temporarily fixed other vertices as in the barycentric model. As a substitute, a modified Newton-Raphson method can be used to approximate an optimal move for a single vertex [13]. Since this method does not scale to graphs with thousands of vertices, we next describe our modifications to make it work on connection graphs.[2]

Sparsening. If springs are introduced between every pair of vertices, a single iteration takes time quadratic in the number of vertices. Since at least a linear number of iterations is needed, this is clearly not feasible. Since, moreover, we are not interested in a readable layout of the graph, but in supporting the geometric speed-up heuristics for shortest-path computations, there is no need to introduce springs between all pairs of vertices.

We cannot omit springs corresponding to edges, but in connection graphs, the number of edges is of the order of the number of vertices, so most of the pairs in (2) are connected by a shortest path with at least two edges. If a train runs along a path of k edges, we call this path a *k-connection*. To model the plausible assumption that, locally, trains run fairly straight, we include only terms corresponding to edges (or 1-connections) and to 2- and 3-connections into the potential. Whenever there are two or more springs for a single pair of

[2] In the graph drawing literature, similar objective functions have been subjected to simulated annealing [5, 4] and genetic algorithms [14, 2]. These methods seem to scale even worse.

vertices, they are replaced by a single spring of the average length. For realistic data, the total number of springs thus introduced is linear in the number of vertices.

Long-range dependencies. Since we omit most of the long-range dependencies (i.e. springs connecting distant pairs of vertices), an iterative method starting from a random layout is almost surely trapped in a very poor local minimum.

We therefore determine an initial layout by computing a local minimum of the potential on an even sparser graph, that does only include the long-range dependencies relevant for our approach. That is, we consider the subgraph consisting of all stations that have a fixed position or are a terminal station of some train, and introduce springs only between the two terminal stations of each train, and between pairs of the selected vertices that are consecutive on the path of any train. We refer to these additional pairs as *long-range connections*. In case the resulting graph has more components than the connection graph, we heuristically add some stations touched by trains inducing different components and the respective springs. After running our layout algorithm on this graph (initialized with a barycentric layout), the initial position for all other vertices is determined from a barycentric layout in which those positions that have already been computed are fixed.

Iterative improvement. We compute a local minimum of a potential $U(p)$ by relocating one vertex at a time according to the forces acting on it, i.e. the negative of the gradient, $-\nabla U(p)$.

For each node v (in arbitrary order) we move only this node in dependence of $U(p_v)$. The node is shifted in the opposite direction of

$$d := \nabla U(p_v) := \left\langle \frac{\partial U(p_v)}{\partial x_v}, \frac{\partial U(p_v)}{\partial y_v}, \frac{\partial U(p_v)}{\partial z_v} \right\rangle$$

A substantial parameter of a gradient descent is the size of each step. For small graphs it is often sufficient to take a fixed multiple of the gradient (see the classic example of [6]), while others suggest some sort of step size reduction schedule (e.g., see [8]). We applied a more elaborate method that is robust against change of scale, namely the method of Wolfe and Powell (see, e.g., [23, 15]). The step size $\sigma \in (0, \infty)$ is determined by

$$\frac{\nabla U(p_v - \sigma d)d}{\nabla U(p_v)d} \leq \kappa$$

$$\frac{U(p_v) - U(p_v - \sigma d)}{\sigma \cdot \nabla U(p_v)d} \geq \delta$$

for given parameters $\delta \in (0, 0.5)$ and $\kappa \in (\delta, 1)$. Roughly speaking, this guarantees that the potential is reduced and that the step is not too small. In our experiments, this method clearly outperformed the simpler methods both in terms of convergence and overall running time.

Another dimension. Generally speaking, a set of desired edge lengths can be resembled more closely in higher-dimensional Euclidean space. Several models make use of this observation by temporarily allowing additional coordinates and then penalizing their deviation from zero [24] or projecting down [9].

We use a third coordinate during all phases of the layout algorithm, but ignore it in the final layout. Since projections do not preserve the edge lengths, we use a penalty function $\sum_{v \in V} c_t \cdot z_v^2$, where c_t is the penalty weight at the tth iteration, to gradually reduce the value of the z-coordinate towards the end of the layout computation. Experimentation showed that the final value of the potential is reduced by more than 10% with respect to an exclusively two-dimensional approach.

In summary, our layout algorithm consists of the following four steps:

1. barycentric layout of graph of long-range connections
2. iterative improvement
3. barycentric layout of graph of 2-, 3-, and long-range connections
4. iterative improvement with increasing z-coordinate penalties

In each of these steps, the iteration is stopped when none of the stations was moved by more than a fixed distance. Figure 3 shows the results of this approach when applied to the graph of Fig. 2.

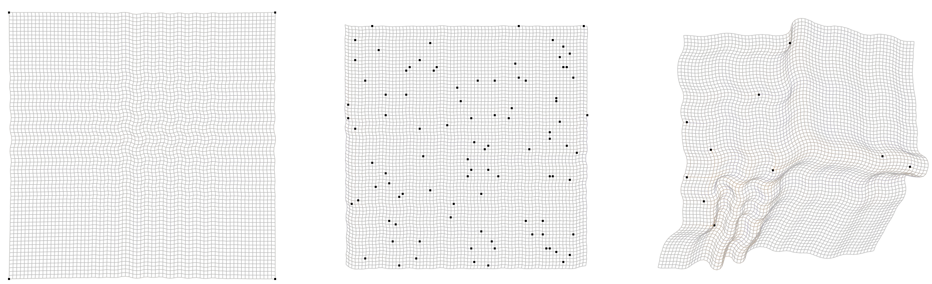


Fig. 3. Layouts of the graph of Fig. 2, where fictitious trains run along grid lines, under specific model

5 Results and Discussion

Our computational experiments are based on the timetables of the Deutsche Bahn AG, Germany's national train and railroad company, for the winter period 1996/97. It contains a total of 933,066 arrivals and departures on 6,961 stations, for which we have complete coordinate information.

To assess the quality of coordinates generated by the layout algorithms described in Sects. 3 and 4, we used a snapshot from the central travel information server of Deutsche Bahn AG. This data consists of 544,181 queries collected over several hours of a regular working day.

This benchmark data are unique in the sense that it is the only real network for which we have both coordinates and query data.

In the experiments, shortest paths are computed for these queries using our own implementation of Dijkstra's algorithm and the angle-restriction and goal-directed search heuristics. All implementations are in C or C++, compiled with gcc version 2.95.2.

From the timetables we generated the following instances:

- `de-org` (coordinates known for all stations)
- `de-22-important` (coordinates known for the 22 most important[3] stations)
- `de-22-random` (coordinates known for 22 randomly selected stations)
- `de` (no coordinates given)

For these instances, we generated layouts using the barycentric model of Sect. 3 and the specific model of Sect. 4, and measured the average core CPU time spent on answering the queries, as well as the number of vertices touched by the modified versions of Dijkstra's algorithm. Each experiment was performed on a single 336 Mhz UltraSparc-II processor of a Sun Enterprise 4000/5000 workstation with 1024 MB of main memory. The results are given in Tab. 1, and the layouts are shown in Figs. 4–5.

Table 1. Average query response times and number of nodes touched by Dijkstra's algorithm. Without coordinates, the average response time is **104.0** ms (18406 nodes)

	heuristic:	angles		goal		both	
instance	layout model	∅ ms	nodes	∅ ms	nodes	∅ ms	nodes
`de-org` (Fig. 4)		**16.4**	4585	**82.5**	11744	**13.9**	3317
`de` (Fig. 5)	specific	**41.1**	9034	**108.2**	15053	**40.7**	7559
`de-22-important`	barycenteric	**18.9**	5062	**117.1**	17699	**20.1**	4918
(Fig. 6)	specific	**19.5**	5158	**84.6**	11992	**16.8**	3762
`de-22-random`	barycentric	**18.1**	5017	**111.4**	17029	**19.3**	4745
(Fig. 7)	specific	**20.7**	5383	**88.0**	12388	**17.9**	3985

The results show that the barycentric model seems to match very well with the angle-restriction heuristic when important stations are fixed. The somewhat surprising usefulness of this simple model even for the randomly selected stations seems to be due to the fact that our sample spreads out quite well. It will be interesting to study this phenomenon in more detail, since the properties that make a set of stations most useful are important for practical purposes.

The specific model appears to work well in all cases. Note that the average response time for connection queries is reduced by 60%, even without any

[3] Together with the coordinate information, there is an value associated with each station that indicates its importance as a hub. The 22 selected stations have the highest attained value.

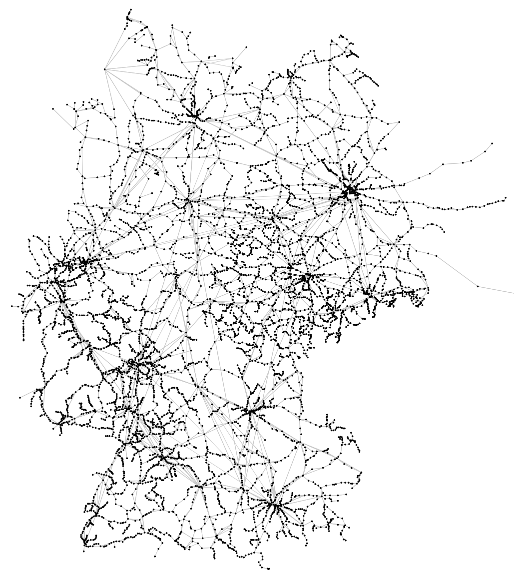


Fig. 4. `de-org`

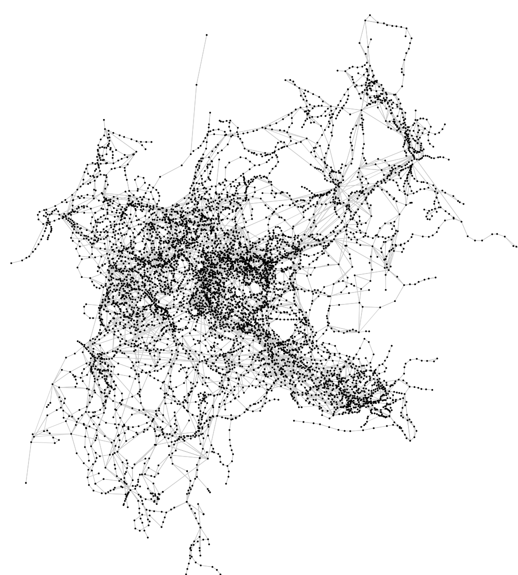
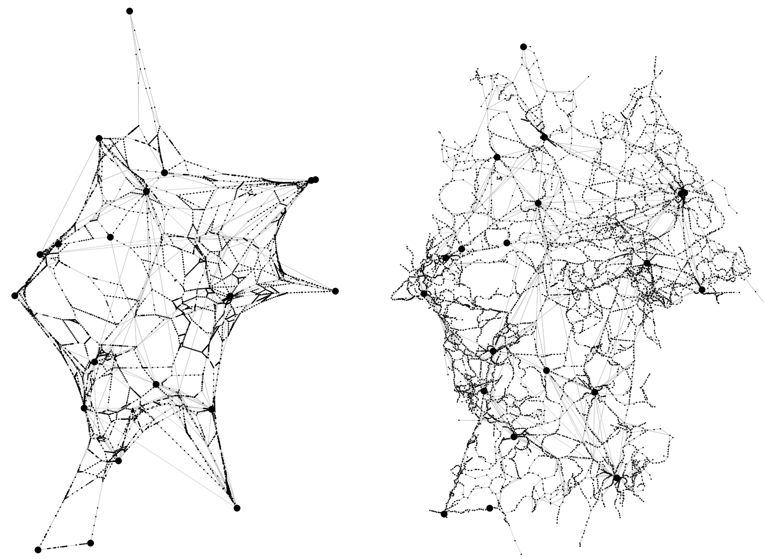
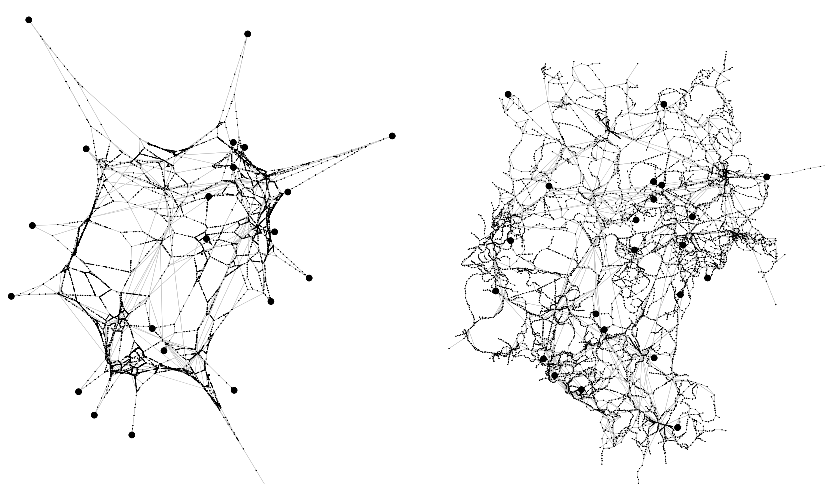


Fig. 5. `de`



barycentric specific

Fig. 6. `de-22-important`



barycentric specific

Fig. 7. `de-22-random`

knowledge of the underlying geography. With the fairly realistic assumption that the location of a limited number of important stations is known, the speed-up obtained with the actual coordinates is almost matched.

To evaluate whether the specific model achieves the objective to preserve given edge length, we generated additional instances from `de-org` by dropping a fixed percentage of station coordinates ranging from 0–100%, while setting λ_e to its true value. As can be seen in Fig. 8, these distances are reconstructed quite well.

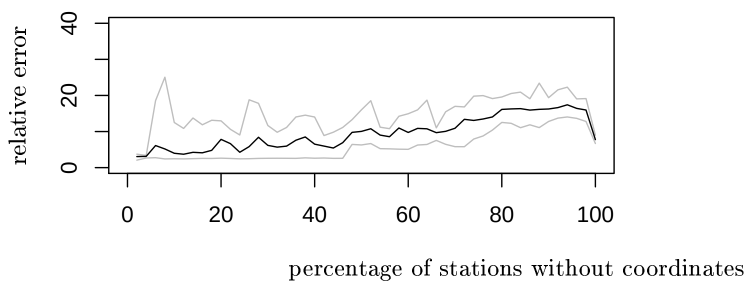


Fig. 8. Minimum, mean, and maximum relative error in edge lengths, averaged over ten instances each

As of now, we do not know how our method compares with the most recent developments in force-directed placement for very large graphs [9, 26], in particular the method of [11].

Acknowledgments. We thank Steffen Mecke for help with preliminary experiments, and companies TLC/EVA and HaCon for providing timetables and connection query data, respectively.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows.* Prentice-Hall, 1993.
2. J. Branke, F. Bucher, and H. Schmeck. A genetic algorithm for drawing undirected graphs. *Proc. 3rd Nordic Workshop on Genetic Algorithms and their Applications*, pp. 193–206, 1997.
3. 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.
4. I. F. Cruz and J. P. Twarog. 3D graph drawing with simulated annealing. *Proc. 3rd Intl. Symp. Graph Drawing (GD '95)*, Springer LNCS 1027, pp. 162–165, 1996.
5. R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996.

6. P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
7. P. Eades and N. C. Wormald. Fixed edge-length graph drawing is np-hard. *Discrete Applied Mathematics*, 28:111–134, 1990.
8. T. M. Fruchterman and E. M. Reingold. Graph-drawing by force-directed placement. *Software—Practice and Experience*, 21(11):1129–1164, 1991.
9. P. Gajer, M. T. Goodrich, and S. G. Kobourov. A fast multi-dimensional algorithm for drawing large graphs. *Proc. Graph Drawing 2000.* To appear.
10. G. H. Golub and C. F. van Loan. *Matrix Computations.* Johns Hopkins University Press, 3rd edition, 1996.
11. D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *Proc. Graph Drawing 2000.* To appear.
12. S. Jung and S. Pramanik. HiTi graph model of topographical road maps in navigation systems. *Proc. 12th IEEE Int. Conf. Data Eng.*, pp. 76–84, 1996.
13. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
14. C. Kosak, J. Marks, and S. Shieber. Automating the layout of network diagrams with specified visual organization. *IEEE Transactions on Systems, Man and Cybernetics*, 24(3):440–454, 1994.
15. P. Kosmol. *Methoden zur numerischen Behandlung nichtlinearer Gleichungen und Optimierungsaufgaben.* Teubner Verlag, 1993.
16. T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley, 1990.
17. K. Nachtigall. Time depending shortest-path problems with applications to railway networks. *European Journal of Operational Research* 83:154–166, 1995.
18. T. Preuß and J.-H. Syrbe. An integrated traffic information system. *Proc. 6th Intl. EuropIA Conf. Appl. Computer Networking in Architecture, Construction, Design, Civil Eng., and Urban Planning.* Europia Productions, 1997.
19. F. Schulz, D. Wagner, and K. Weihe. Dijkstra's algorithm on-line: an empirical case study from public railroad transport. *Proc. 3rd Workshop on Algorithm Engineering (WAE '99)*, Springer LNCS 1668, pp. 110–123, 1998.
20. R. Sedgewick and J. S. Vitter. Shortest paths in euclidean space. *Algorithmica* 1:31–48, 1986.
21. S. Shekhar, A. Kohli, and M. Coyle. Path computation algorithms for advanced traveler information system (*ATIS*). *Proc. 9th IEEE Intl. Conf. Data Eng.*, pp. 31–39, 1993.
22. L. Siklóssy and E. Tulp. TRAINS, an active time-table searcher. *Proc. 8th European Conf. Artificial Intelligence*, pp. 170–175, 1988.
23. P. Spellucci. *Numerische Verfahren der nichtlinearen Optimierung.* Birkhäuser Verlag, 1993.
24. D. Tunkelang. JIGGLE: Java interactive general graph layout environment. *Proc. 6th Intl. Symp. Graph Drawing (GD '98)*, Springer LNCS 1547, pp. 413–422, 1998.
25. W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society, Third Series*, 13:743–768, 1963.
26. C. Walshaw. A multilevel algorithm for force-directed graph drawing. *Proc. Graph Drawing 2000.* To appear.