Generating Node Coordinates for Shortest-Path Computations in Transportation Networks

ULRIK BRANDES
University of Konstanz
FRANK SCHULZ, DOROTHEA WAGNER, and THOMAS WILLHALM
University of Karlsruhe

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 drawings is close to that achieved with the true coordinates—and in some special cases even better.

Categories and Subject Descriptors: G.2.3 [**Discrete Mathematics**]: Applications

General Terms: Graph

Additional Key Words and Phrases: Graph drawing, shortest paths, transportation networks, travel planning

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 [Sedgewick and Vitter 1986; Schulz et al. 2000; Schulz et al. 2002] and are, in fact, used in many such systems.

A previous version appeared as *Travel Planning With Self-Made Maps*, at the Workshop on Algorithm Engineering and Experiments (ALENEX 2001).
This work was partially supported by the Deutsche Forschungsgemeinschaft (DFG) under grant WA 654/12-1 and the Human Potential Programme of the European Union under contract no. HPRN-CT-1999-00104 (AMORE).
Authors' addresses: Ulrik Brandes, Department of Computer & Information Science, University of Konstanz, Box D 67, 78457 Konstanz, Germany; email: ulrik.brandes@uni-konstanz.de, http://www.inf.uni-konstanz.de/algo/; Frank Schulz, Dorothea Wagner, and Thomas Willhalm, Department of Computer Sciences, University of Karlsruhe, Box 6980, 76128 Karlsruhe, Germany; email: {fschulz,dwagner,willhalm}@ira.uka.de, http://i11www.ira.uka.de/.

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, that is, the location of nodes in a network, is unknown, since not all transport authorities provide this information to travel service providers or competitors. This is particularly true if the provider of the travel information system differs from the transportation company, for example, for foreign trains or local bus companies. The situation will become even more serious in future with the liberalization of the European railway network and the integration of local buses in door-to-door queries. Furthermore, producers of a timetable information system need to demonstrate their travel-planning software to new customers who usually do not have coordinates in a ready-to-use state. 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 nonstop 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. This problem is closely related to drawing Internet latency maps and positioning algorithms for wireless ad hoc networks. However, it is *not* our goal to reconstruct the original coordinates, but to produce coordinates with which the speed-up techniques perform well.

Our specific scenario and geometric speed-up heuristics for shortest-path computations are reviewed briefly in Section 2. In Section 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 Section 4 to be applicable in more general situations. In Section 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-planning systems for, for example, car navigation [Shekhar et al. 1993; Jung and Pramanik 1996] or public transport [Nachtigall 1995; Preuss and Syrbe 1997; Siklóssy and Tulp 1988], often make use of geometric speed-up techniques for shortest-path computations. We consider the (simplified) scenario of a travel-planning system for public railroad transport used in a recent pilot study [Schulz et al. 2000]. 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.

[1] HaCon Ingenieurgesellschaft mbh, Hannover.

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 which introduces cycles in the graph). 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. (Travel edges introduce more complex cycles.)

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 Schulz et al. [2000], 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, that is, those based directly on the coordinates of the stations, which can be combined with other techniques. Although artificial coordinates can be used with other heuristic algorithms, this paper deals only with speed-up techniques that assure the correctness of the result.

2.1.1 *Goal-Directed Search.* This strategy is found in many textbooks (e.g., see Ahuja et al. 1993; Lengauer 1990). 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.

In our case, the maximum speed relative to the generated coordinate set is determined beforehand by a linear scan over all edges in the graph. This guarantees that goal-directed search finds the correct result whatever the

coordinates are. “Bad” coordinates however lead to a slow performance of the algorithm.

2.1.2 *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, that starts with the stay edge from v.

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 nonstop connection, which in turn are used to generate locations suitable for the geometric heuristics, though in general far from being geographically accurate.

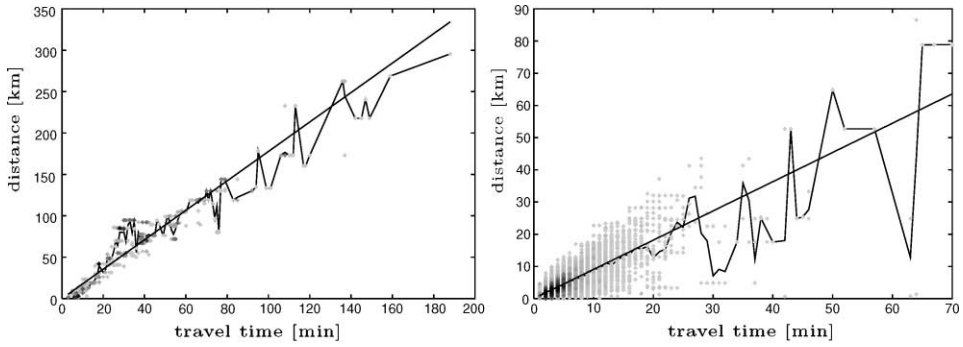
The (undirected, simple) *station graph* of a set of timetables contains a vertex for each station listed, and an edge between every pair of stations connected by a train not stopping in between. The *length* λ_e of an edge e in the station graph will represent our estimate of the distance between its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

Distance between two stations can be expected to be roughly linear in the travel time. However, for different classes of trains the constant involved will be different, and closely related to the mean velocity of trains in this class. We therefore estimate the length of an edge e in the station graph, that is, the distance between two stations: Consider all nonstop connections that induce this edge. We use as estimated distance the mean value of their travel time times the average velocity of the vehicle serving the connection.

Mean velocities have been extracted from the data set described in Section 5, for which station coordinates are known. For two train categories, the data are depicted in Figure 1, indicating that linear approximations yield a fairly good estimation.

Note that all travel times are integers, 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.

Apart from the estimation of geographic distances, a second set of edge lengths has been tested: a distance that is proportional to the average travel time of all vehicles including this edge. (It is the same as assuming that all



(a) high-speed trains (b) local trains

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

trains have the same average velocity.) This approach can be justified by the insight that the travel-planning system minimizes only the travel time.

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 remaining 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

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

where $\omega_e = 1/\lambda_e$, $e \in E$, weights the influence of an edge according to its estimated length λ_e, defines a weighted *barycentric layout model* [Tutte 1963]. 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

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

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

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

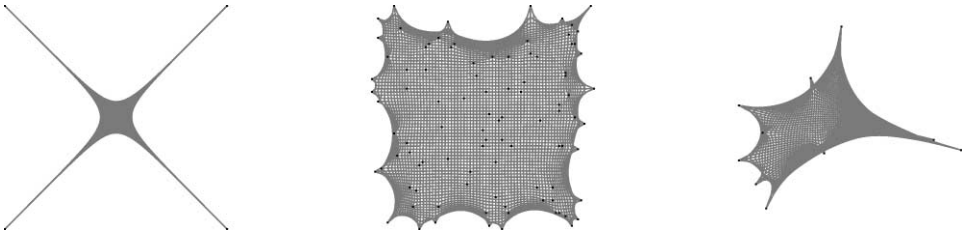


Fig. 2. Barycentric layout of an 72×72 grid with the four corners fixed, and the same grid with 95 and with 10 randomly selected vertices fixed.

unique solution, if at least one p_v in each connected component of G is given (and the equations induced by v are omitted) [Brooks et al. 1940]. 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., Golub and van Loan 1996).

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, for example, Gauss–Seidel iteration—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 Section 5, it nevertheless has some practical merits.

4. A TAILOR-MADE 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 [1989] use springs of length $\lambda_{\{u,v\}} = d_G(u, v)$, which is equal to the length of a shortest path between u and v, between every pair of vertices. The potential then becomes

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

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 $\omega_e = 1/\lambda_e^2$, 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 [Eades and Wormald 1990]. In contrast to the barycentric model, the necessary condition of vanishing partial

derivatives leads to a system of nonlinear 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 [Kamada and Kawai 1989; Kumar and Fowler 1994]. 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]

4.1 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 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.

4.2 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 poor local minimum.

We therefore determine an initial layout by computing a local minimum of the potential on an even sparser graph that includes only 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

[2]In the graph drawing literature, similar objective functions have been subjected to simulated annealing [Davidson and Harel 1996; Cruz and Twarog 1996] and genetic algorithms [Kosak et al. 1994; Branke et al. 1997]. These methods seem to scale even worse.

vertices is determined from a barycentric layout in which those positions that have already been computed are fixed.

To measure the impact of this modification, we computed 100 layouts with and without modification and compared the average of the final potential values. It turns out that the result with this initial layout has a 9% smaller value of the objective function.

4.3 Nodes of High Degree

The method to find an initial layout of the sparser graph in the previous section can be seen as a two-level approach of the embedding algorithm: The first level is the full graph, the second level is the graph of long-range connections, which is embedded first. Although the second level already improves the final potential function, in case no station at all has a fixed position, the iterative method is still likely to be trapped in a local minimum. The introduction of a third, even smaller graph therefore leads to another improvement of the potential function of 21%. (In case of 22 given coordinates for major hubs, the improvement is only 0.1%, however.)

To determine the third level, we use again the structure of the graph: We iteratively replace all nodes of degree three or less in the second level by edges between their neighbors. More precisely, nodes

—of degree 0 or 1 are removed,

—of degree 2 are replaced by an edge with length equal to the sum of the lengths of incident edges,

—of degree 3 are replaced by a triangle of edges, where the edge lengths are computed as if the angles between the incident edges of the node had been 60°.

4.4 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, that is, 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 method 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 Eades [1984]), while others suggest some sort of step size reduction schedule (e.g., see Fruchterman and Reingold 1991).

We applied a more elaborate method that is robust against change of scale, namely the method of Wolfe and Powell (see, e.g., Spellucci 1993; Kosmol 1993). 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.

We also implemented the Newton–Raphson method, but it turned out to be an order of magnitude slower to achieve the same minima. This is mainly due to the fact that we worked in three dimensions (see the next section), where it is necessary to invert a 3×3-matrix and to compute six instead of three second derivatives. The matrix inversion was performed by our own implementation as well as by LAPACK [Anderson et al. 1999]. Both versions were not competitive with the method of Wolfe and Powell.

4.5 Another Dimension

Generally speaking, a set of desired edge lengths can be realized more accurately when the number of dimensions of the Euclidean space is increased.

Several models make use of this observation by temporarily allowing additional coordinates and then penalizing their deviation from zero [Tunkelang 1998] or projecting down [Gajer et al. 2001].

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,[3] we use a penalty function $\sum_{v \in V} c_t z_v^2$, 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.

The penalty has to be chosen large enough such that the graph is finally pressed into the plane and small enough that the result differs from the projection.[4] Let P_0 denote the potential at the beginning of the last phase. Experiments showed that a multiple of the unmodified potential that increases like the fourth power of the time gives good results: $c_t = CP_0t^4$.

The average final potential of 100 runs of the algorithm is reduced by 16% with respect to an exclusively two-dimensional approach.

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

(1) Barycentric layout of graph of long-range connections for nodes of high degree.
(2) Iterative improvement.
(3) Barycentric layout of graph of long-range connections.
(4) Iterative improvement.

[3] In our experiments with 100 layouts, the objective function triples if the graph is simply projected into the plane.
[4] In our experiments with 100 layouts, the objective function of a projection into the plane and a subsequent optimization in the plane is 7% worse.

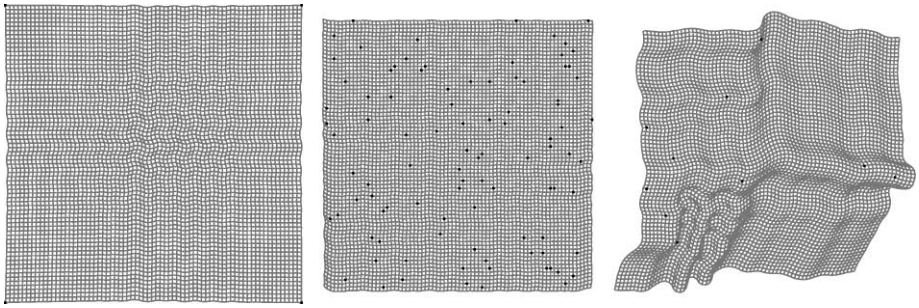


Fig. 3. Layouts of the graph of Figure 2, where fictitious trains run along grid lines, under tailored model.

(5) Barycentric layout of entire graph including 2-, 3-, and long-range connections.
(6) 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 Figure 2.

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 1996/1997.

It contains a total of 933,280 arrivals and departures on 6,884 stations, for which we have the complete coordinate information.

To assess the quality of coordinates generated by the layout algorithms described in Sections 3 and 4, we used a snapshot of queries against 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.

These 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 the above 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[5] stations);
—`de-22-random` (coordinates known for 22 randomly selected stations);
—`de` (no coordinates given);

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

Table I. Average Query Response Times and Number of Nodes Touched by Dijkstra's Algorithm. Without Coordinates, the Average Response Time is **105** ms (33,704 edges)

		Speed-up Technique					
		Angles		Goal		Both	
Instance	Layout Model	ms	Edges	ms	Edges	ms	Edges
de-org		**17**	9,177	**79**	20,995	**14**	6,496
de (Figure 5)	Tailored (dist. est.)	**40**	17,553	**107**	28,591	**43**	15,167
	Tailored (time avg.)	**43**	18,228	**92**	24,634	**38**	13,888
de-22-important (Figure 6)	Barycenteric	**20**	10,464	**100**	26,669	**19**	8,722
	Tailored (dist. est.)	**19**	9,844	**93**	24,334	**19**	7,763
	Tailored (time avg.)	**22**	10,994	**75**	20,052	**17**	7,412
de-22-random (Figure 7)	Barycentric	**27**	13,415	**124**	32,628	**31**	13,066
	Tailored (dist. est.)	**21**	10,597	**87**	23,033	**18**	7,973
	Tailored (time avg.)	**27**	11,671	**77**	20,853	**22**	8,055

Fig. 4. de-org.

For these instances, we generated layouts using the barycentric model of Section 3, the tailored model of Section 4 with estimated distances, or the tailored model with average travel time as "distance," and measured the average core CPU time spent on answering the queries, as well as the number of edges 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 1,024 MB of main memory. The results are given in Table I, and the layouts are shown in Figures 4 and 5.

The results show that the barycentric model seems to pair 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.

Fig. 5. de.

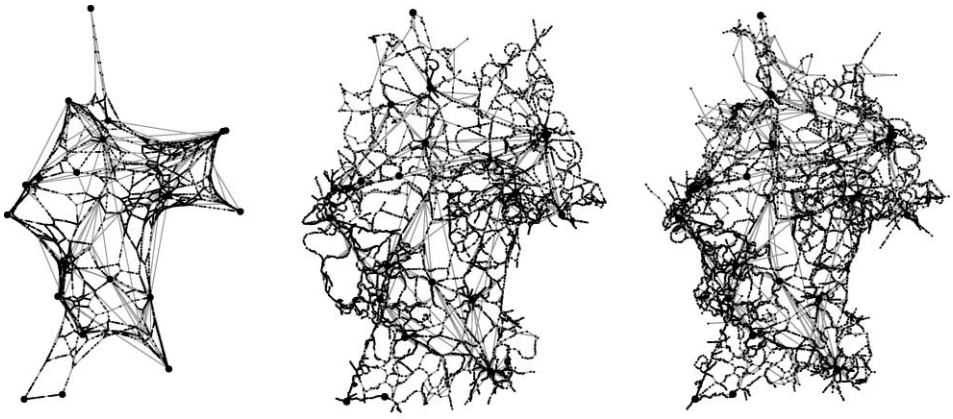


Fig. 6. de-22-important.

Another interesting observation is that the layouts according to average travel times are better for goal-directed search than the layouts that use estimated distances. This can be explained by the fact that goal-directed search uses the highest speed to calculate the lower bound to the destination, which is tighter in this case.

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

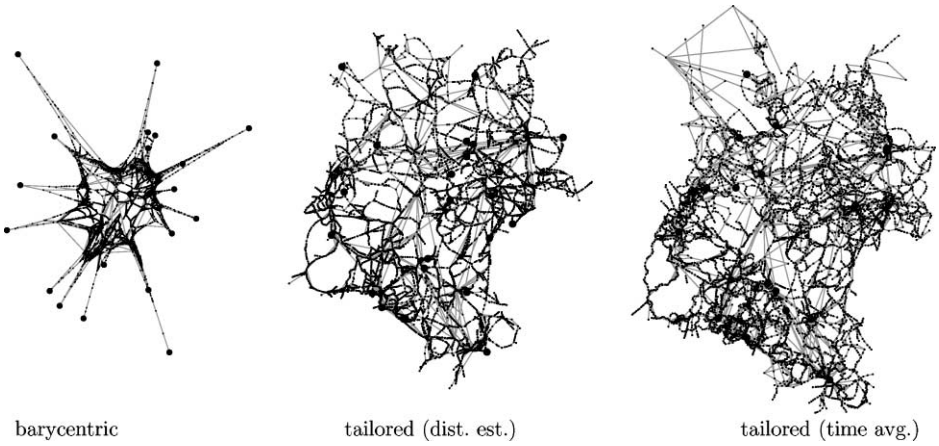


Fig. 7. de-22-random.

Fig. 8. de-org-t.

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 tailored model achieves the objective of preserving given edge lengths, we generated additional instances from de-org by dropping a fixed percentage ranging from 0% to 100% of station coordinates, while setting λ_e to its true value. As can be seen in Figure 9, these distances are reconstructed quite well.

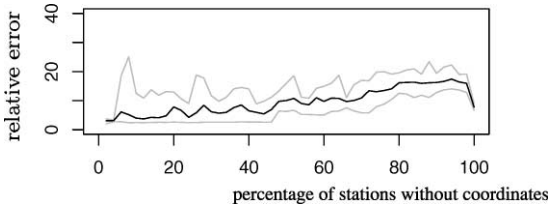


Fig. 9. Evaluation whether the tailored layout model preserves edge lengths. Minimum, mean, and maximum relative error in edge lengths, averaged over 10 instances each.

Table II. Average Query Response Times and Number of Nodes Touched by Dijkstra's Algorithm for the Layout Based on Original Coordinates

		speed-up technique					
		Angles		Goal		Both	
Instance	Layout Model	ms	Edges	ms	Edges	ms	Edges
de-org-t	modified org	**18**	9,384	**71**	18,782	**14**	6,086

As an attempt to avoid the small but existing shortcomings of the layout algorithms as much as possible, we generated a further set of coordinates de-org-t that uses the original coordinates as initialization. The layout was then modified locally to fit the travel times as much as possible.

The result is depicted in Figure 8 and the resulting query times are shown in Table II. With de-org-t, the results are slightly better than the results with any other generated coordinates. This suggests that there is still some space for improvement of the layout algorithm, but probably not much. The main result of this last experiment, however, is the observation that the original coordinates can be improved with respect to the goal-directed search.

6. OUTLOOK

The results and pictures suggest that our layout algorithm produces a reasonably good reconstruction of the traffic network with respect to the travel-planning system.

The query response times for the modified original coordinates de-org-t show that it is possible to create layouts that are tailored to shortest-path problems. It will be interesting to study this phenomenon in more detail and search for characteristics of layouts that pair well with specific geometric speed-up techniques.

When it comes to ship and flight schedules it is not possible anymore to ignore the fact that the earth is not flat. Fortunately, it is easy to modify the algorithm in such a way that it works on a sphere. It is sufficient

—to use a metric that reflects the slope of the earth,

—to modify penalty function the projection.

Whereas the latter can be done in a canonical way, the new distance should avoid expensive trigonometric calculations. A fifth order approximation of the arcus sine multiplied by the Euclidean distance turns out to work well in practice.

ACKNOWLEDGMENTS

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

REFERENCES

AHUJA, R., MAGNANTI, T., AND ORLIN, J. 1993. *Network Flows*. Prentice-Hall, Englewood Cliffs, NJ.

ANDERSON, E., BAI, Z., BISCHOF, C., BLACKFORD, S., DEMMEL, J., DONGARRA, J., DU CROZ, J., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., AND SORENSEN, D. 1999. *LAPACK User's Guide*, 3rd ed. Society for Industrial and Applied Mathematics. Available at `http://www.netlib.org/lapack/`.

BRANKE, J., BUCHER, F., AND SCHMECK, H. 1997. A genetic algorithm for drawing undirected graphs. In *Proceedings of the Third Nordic Workshop on Genetic Algorithms and their Applications*. 193–206.

BROOKS, R. L., SMITH, C. A. B., STONE, A. H., AND TUTTE, W. T. 1940. The dissection of rectangles into squares. *Duke Math. J. 7*, 312–340.

CRUZ, I. F. AND TWAROG, J. P. 1996. 3D graph drawing with simulated annealing. In *Proceedings of the Third International Symposium on Graph Drawing (GD '95)*, F. J. Brandenburg, ed. Lecture Notes in Computer Science, vol. 1027. Springer, Berlin, 162–165.

DAVIDSON, R. AND HAREL, D. 1996. Drawing graphs nicely using simulated annealing. *ACM Trans. Graphics 15*, 4, 301–331.

EADES, P. 1984. A heuristic for graph drawing. *Congressus Numerantium 42*, 149–160.

EADES, P. AND WORMALD, N. C. 1990. Fixed edge-length graph drawing is np-hard. *Discrete Appl. Math. 28*, 111–134.

FRUCHTERMAN, T. M. AND REINGOLD, E. M. 1991. Graph-drawing by force-directed placement. *Softw. - Pract. Exp. 21*, 11, 1129–1164.

GAJER, P., GOODRICH, M. T., AND KOBOUROV, S. G. 2001. A fast multi-dimensional algorithm for drawing large graphs. In *Proceedings of the Eighth International Symposium on Graph Drawing (GD 2000)*, J. Marks, ed. Lecture Notes in Computer Science, vol. 1984. Springer, Berlin, 211–221.

GOLUB, G. H. AND VAN LOAN, C. F. 1996. *Matrix Computations*, 3rd ed. Johns Hopkins University Press, Baltimore, MD.

JUNG, S. AND PRAMANIK, S. 1996. Hiti graph model of topographical road maps in navigation systems. In *Proceedings of the 12th IEEE International Conference Data Engineering*. 76–84.

KAMADA, T. AND KAWAI, S. 1989. An algorithm for drawing general undirected graphs. *Inf. Process. Lett. 31*, 7–15.

KOSAK, C., MARKS, J., AND SHIEBER, S. 1994. Automating the layout of network diagrams with specified visual organization. *IEEE Trans. Syst. Man Cybern. 24*, 3, 440–454.

KOSMOL, P. 1993. *Methoden zur numerischen Behandlung nichtlinearer Gleichungen und Optimierungsaufgaben*. Teubner Verlag.

KUMAR, A. AND FOWLER, R. 1994. *A Spring Modeling Algorithm to Position Nodes of an Undirected Graph in Three Dimensions*. Tech. rep., Department of Computer Science, University of Texas, Pan American, Edinburgh.

LENGAUER, T. 1990. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley, New York.

NACHTIGALL, K. 1995. Time depending shortest-path problems with applications to railway networks. *Eur. J. Oper. Res. 83*, 1, 154–166.

PREUSS, T. AND SYRBE, J.-H. 1997. An integrated traffic information system. In *Proceedings of the Sixth International Conference on Applied Computer Networking in Architecture, Construction, Design, Civil Engineering, and Urban Planning (europIA '97)*.

SCHULZ, F., WAGNER, D., AND WEIHE, K. 2000. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *J. Exp. Algorithmics 5*, 12.

SCHULZ, F., WAGNER, D., AND ZAROLIAGIS, C. 2002. Using multi-level graphs for timetable information. In *Proceedings of the Algorithm Engineering and Experiments (ALENEX '02)*. Lecture Notes in Computer Science, Springer, Berlin, in press.

SEDGEWICK, R. AND VITTER, J. S. 1986. Shortest paths in Euclidean space. *Algorithmica 1*, 1, 31–48.

SHEKHAR, S., KOHLI, A., AND COYLE, M. 1993. Path computation algorithms for advanced traveler information system (ATIS). In *Proceedings of the Ninth IEEE International Conference on Data Engineering*. 31–39.

SIKLÓSSY, L. AND TULP, E. 1988. Trains, an active time-table searcher. In *Proceedings of the 8th European Conference on Artificial Intelligence*. 170–175.

SPELLUCCI, P. 1993. *Numerische Verfahren der nichtlinearen Optimierung*. Birkhäuser Verlag.

TUNKELANG, D. 1998. JIGGLE: Java interactive general graph layout environment. In *Proceedings of the Sixth International Symposium on Graph Drawing (GD '98)*, S. H. Whitesides, ed. Lecture Notes in Computer Science, vol. 1547. Springer, Berlin, 413–422.

TUTTE, W. T. 1963. How to draw a graph? *Proc. London Math. Soc., Third Series 13*, 743–768.

Received April 2002; revised March 2004; accepted April 2004