In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Dallas, TX, November 2014

Automated Tabular Itinerary Visualization*

Marco D. Adelfio Hanan Samet
Center for Automation Research, Institute for Advanced Computer Studies
Department of Computer Science, University of Maryland
College Park, MD 20742 USA
{marco, hjs}@cs.umd.edu

ABSTRACT

Advances in geographic information extraction have exposed previously untapped resources, such as many travel itineraries found in HTML tables and spreadsheets on the Web. In the general sense, itineraries differ from the related concepts of routes and trajectories in that the precise paths between stopping points are of far less importance than the locations of the stopping points and their order. This characteristic allows for some flexibility when visualizing itineraries. A method for automatically generating itinerary visualizations is presented, which utilizes principles from graph-drawing and map labeling, along with additional criteria designed specifically for the itinerary visualization task. We describe a system based on this method that can perform automated layout for arbitrary itineraries at varying scales.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Design

Keywords

Itinerary layout, map labeling, geographic visualization

1. INTRODUCTION

We define an *itinerary* as an ordered set of waypoints (also known as *stops*). Waypoint locations are commonly specified using latitude/longitude pairs or Cartesian coordinates. Each point may have additional information associated with it, such as a place name or a date and time of visit.

Itineraries are similar in some ways to both *routes* and *trajectories*, which are abstractions that are also used to represent movement of an object or person over time. However, the three terms can be used to describe three different emphases when dealing with paths, which in turn suggest that distinct attributes should be emphasized in their visualizations. In particular, when a path is mostly concerned with laying out a particular sequence of steps or segments between consecutive pairs of waypoints, we call it a route. A trajectory, on the other hand, aims to represent a recorded or estimated path and edges should aim to accurately represent an interpolated path between waypoints. In contrast to both of these, an itinerary's aim is to communicate the location and order of the waypoints, with a minimal expectation of precision communicated via the edge paths. In other words, under these definitions, a route emphasizes decision points and the paths connecting them, a trajectory emphasizes geographical or spatial precision, and an itinerary emphasizes the topology connecting a series of waypoints.

Our goal is to overlay an itinerary visualization over a map in a way that allows viewers to easily discern the ordering and location of points on the itinerary. The process of creating an itinerary visualization is equal parts graph drawing and map labeling, with additional constraints and optimization criteria that differentiate it from both. A fundamental difference from many graph-drawing scenarios is that the locations of the waypoints are fixed to correspond to their locations on a map, whereas graph-drawing techniques typically allow for moving nodes to optimize the layout. Another difference is that we allow for the usage of curved edges for itineraries, which is uncommon in graph-drawing contexts.

The automated itinerary visualization problem is motivated by our work in browsing spatial data [6, 17, 18]) and performing geographic information extraction on HTML tables and spreadsheets [2, 3, 14], which has exposed many geographic datasets, including itineraries. However, presenting itineraries from these datasets requires a more useful visualization than a simple map mash-up, which does not communicate the connectivity or the sequence of visited locations.

2. RELATED WORK

Automated itinerary layout bears a resemblance to several existing problem domains in computer science, including the well-studied areas of graph drawing and map labeling and the more specific sub-area of route rendering.

The canonical graph drawing problem [5, 7] takes as input a graph $G = (V, E)$ (with vertices V and edges E) and seeks to find an appropriate visual representation of the graph, usually by assigning each vertex $v \in V$ coordinates in the 2D plane. While finding an optimal visual representation of a graph is a somewhat subjective task, there are common criteria that are used to approximate visual and aesthetic priorities that are emphasized in manual graph drawing. Some of the primary considerations are to have an even distribution of nodes and edges, to use uniform or near-uniform edge lengths, to display isomorphic substructures uniformly, and to minimize the number of edge crossings [8].

A visualization's utility depends on the amount of information it communicates to viewers. Usability studies have evaluated user preferences for graph layouts that emphasize

*This work was supported in part by the NSF under Grants IIS-10-18475, IIS-12-19023, and IIS-13-20791 and by Google Research and NVIDIA.

SIGSPATIAL'14, November 04-07 2014, Dallas/Fort Worth, TX, USA

http://dx.doi.org/10.1145/2666310.2666377.

specific priorities. Purchase [15] concluded that "reducing the crossings is by far the most important aesthetic, while minimizing the number of bends and maximizing symmetry have a lesser effect." This is especially relevant in the context of the itinerary layout problem since translating waypoints is not possible, meaning the only avenue for reducing edge crossings is by adding bends (i.e., curvature) to the edges. We take this approach, as described in Sections 3 and 4.

One approach to drawing general graphs is to use a force-directed layout in which node and edge pairs are assigned attractive and repulsive forces and a physical simulation is performed to obtain an appropriate layout [7]. Another uses simulated annealing, whereby layouts are iteratively chosen to either improve upon a previous layout or with some probability that approaches zero as the process continues [9]. Simulated annealing is designed to model the physical process of heating a material, then cooling it until it reaches a stable equilibrium state.

Several efforts have looked at automated map layouts for specialized purposes. Route map generalization [4] combines aspects of both graph drawing and map labeling and aims to improve the usability of computer-generated route maps by applying cartographic principles to their design. The focus of these maps is to communicate information that is helpful for navigating between places. LineDrive [4] uses simulated annealing to optimize the route representation, based on a variety of cartographic criteria. Similar efforts have been made with bike maps [16]. More general "origin-destination" maps, including the bicycle flow maps [20] strive to show dense connectivity between nodes overlaid on a map, using asymmetric Bézier curves to emphasize imbalances in bike traffic between two stations. Unlike in itinerary visualization, these maps do not attempt to avoid overlaps between edges or to provide labels for important locations.

3. BACKGROUND

In its basic form, the itinerary visualization problem is to produce a visual encoding of a graph $G = (V, E)$ where V represents the waypoints and E represents edges between consecutive waypoints. Each waypoint $w \in V$ is represented as a tuple (x, y, i, name) where x and y are the Cartesian coordinates describing the location of the waypoint (usually in projected or screen coordinates), i is the index of the waypoint in the current itinerary, and *name* is a string value containing the place name (or other description) for the waypoint that should be included in the visualization. Note that we can treat all point locations as Cartesian coordinates. In case of coordinates encoded as latitude/longitude pairs, we convert to Cartesian coordinates using a suitable geographic projection (e.g., the Mercator).

An interesting aspect of itinerary visualization is that, since edges connecting consecutive waypoints are not meant to track a precise route, the intentional imprecision can be made clear to viewers. One way to achieve this is with simple curves that mimic artistic renderings of itineraries. So, rather than straight edges used in many types of graph visualization, we allow curved edges, where the curvature of the edges is determined by layout parameters.

As shown in Figure 1, the degrees of freedom in our itinerary layout algorithm involve the positioning of the text label associated with each waypoint and the curvature of paths representing journeys between consecutive pairs of waypoints. Thus, we have two parameters (θ and d) for each label and one parameter (r) for each edge, resulting in a total of $3n-1$ parameters for an itinerary consisting of n waypoints.

1. *Label position.* The positions of text labels are determined based on a direction (θ) and distance (d) from the waypoint location. The label is positioned such that the centroid of the label is along the line extending in direction θ from the waypoint, but so that the nearest point on the label's bounding box is at distance d from the waypoint.
2. *Curvature.* Each segment between consecutive waypoints is drawn as a quadratic Bézier curve. The control point of the curve is placed along the perpendicular bisector of the straight line segment connecting the waypoints. For consecutive waypoints w_1 and w_2 and curvature parameter r, control point c is positioned on the perpendicular bisector of $\overrightarrow{w_1w_2}$ at a distance of $r\|\overrightarrow{w_1w_2}\|$ from the segment's midpoint. For positive (negative) values of r, the control point is positioned to the left (right) while traveling from w_1 to w_2.

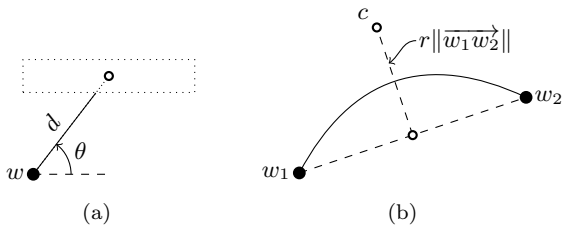


Figure 1: Geometric effects of layout parameters (a) d, θ, and (b) r. Label placement is determined by parameters d and θ, while link curvature is determined by parameter r.

Other potentially usable parameters are discussed in Section 6. However, segment curvature and label position address the most common adjustments that we noticed in a collection of itineraries found on the web that were created by human cartographers. Additionally, the size of the layout search space is exponentially related to the number of available parameters, so we prefer to keep that number small.

The quality of a layout is measured based on the presence or absence of several factors. We observe that the following factors detract from the suitability of an itinerary layout.

1. Labels outside the visible map area
2. Edges overlapping non-incident waypoints
3. Labels overlapping other labels
4. Edges overlapping other edges
5. Labels overlapping edges
6. Labels overlapping waypoints
7. Small angles between incident edges
8. Deviation of segment curvature ratios from target
9. Distance of labels from corresponding waypoints

Our goal is to identify a layout that minimizes the presence of these factors by varying the available parameters, so it can be treated as an optimization problem. As discussed in Section 2, simulated annealing is an option in many layout contexts, including ours, where numerous local minima may act as attractors for a greedy approach. So, we developed a method based on simulated annealing.

4. METHOD

In the simulated annealing context, the value of the objective function for a specific candidate is called its *energy*. We compute the energy of a candidate layout as a weighted sum of negative factors. Formally, we measure energy $e = \sum_{i=1}^{9} \alpha_i f_i$, where each f_i corresponds to one of the undesirable factors from Section 3 (e.g., the number of edge-edge intersections) and α_i represents the corresponding weight. Weights were chosen by evaluating user preferences on a small set of sample itineraries and recording an order of weights assigning the highest penalty to the least desirable layouts

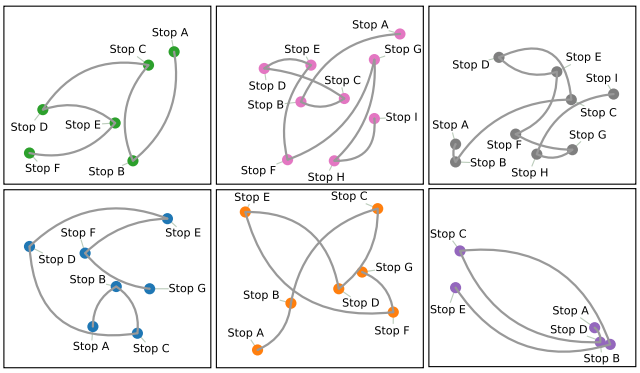


Figure 2: Six sample itinerary layouts. Each itinerary visits a collection of between 5 and 10 randomly-located waypoints. Each itinerary's parameter values are computed independently using our algorithm.

(factors are listed in order, starting with most undesirable). Lower energy values indicate more desirable layouts.

The most computationally expensive components of the energy computation procedure involve detecting intersections between Bézier curves that represent edges and other edges, as well as the intersection of these curves with waypoint nodes and text labels. One method is interval subdivision, which evaluates the curve equation at several points and uses the convex hull properties of Bézier curves to test the bounding box for intersections. Other methods for detecting intersections between Bézier curves include curve implicitization and clipping [19]. However, these methods introduce complexity into the layout algorithm for only modest speed improvements over interval subdivision when the Bézier curves are of low degree (such as the quadratic curves used in our method).

We use the standard simulated annealing formulation, whereby the progression of the algorithm is controlled by a *temperature* variable. A high initial temperature is iteratively reduced, simulating the cooling process that takes place in physical annealing scenarios. As the temperature decreases, parameter changes that result in higher energy layouts are less likely to be chosen as the next state of the system. The acceptance probability for a particular parameter change is based on the Metropolis criterion, a standard simulated annealing acceptance test [9]. The process terminates when the temperature falls below a specified threshold.

The simulated annealing algorithm is shown in Algorithm 1. It takes the input graph, represented as a set of waypoints P and a set of edges E, and augments the input graph with the layout parameters (for curvature and label position) outlined in Section 3. The algorithm begins by initializing the temperature variable t and energy variable e (line 2). For several iterations, the temperature decreases by ratio t_{decay} (line 10) until the temperature falls below the t_{accept} threshold (line 3). In each iteration, a candidate state is chosen (line 4) by picking parameter values for the layout using the CANDIDATE function, which applies a stochastic update to a single layout parameter. The candidate state's energy is computed by COMPUTELAYOUTENERGY (line 5). Candidate layouts that reduce the energy are always accepted, while those that increase the energy are accepted according to the Metropolis criterion (line 6). When a candidate layout is accepted, its parameters and energy are copied to be used for subsequent comparisons (lines 7 and 8). Finally, the resulting layout is returned (lines 12).

The number of iterations taken by simulated annealing here is $log_{t_{decay}}(t_{accept}/t_0)$. Since we expect that each layout parameter requires a consistent number of stochastic updates

Algorithm 1 Augment a collection of waypoints and edges with layout parameter values.

```
1:  procedure FINDLAYOUT(P, E)
    input: List of waypoints P, list of edges E
    output: Lists P, E, augmented with layout parameters
2:      t ← t_0; e ← ∞
3:      while t > t_accept do
4:          P_C, E_C ← CANDIDATE(P, E)
5:          e' ← COMPUTELAYOUTENERGY(P_C, E_C)
6:          if e' < e OR exp((e − e')/t) < RANDOM() then
7:              P, E ← P_C, E_C
8:              e ← e'
9:          end if
10:         t ← t × t_decay
11:     end while
12:     return (P, E)
13: end procedure
```

to arrive at an acceptable value, we update the t_{decay} value in order to maintain a consistent ratio of iterations to layout parameters. In particular, we set $t_{accept} = \phi^{1/n}$, where ϕ is the desired t_{decay} value for an itinerary with a single waypoint and n is the number of waypoints.

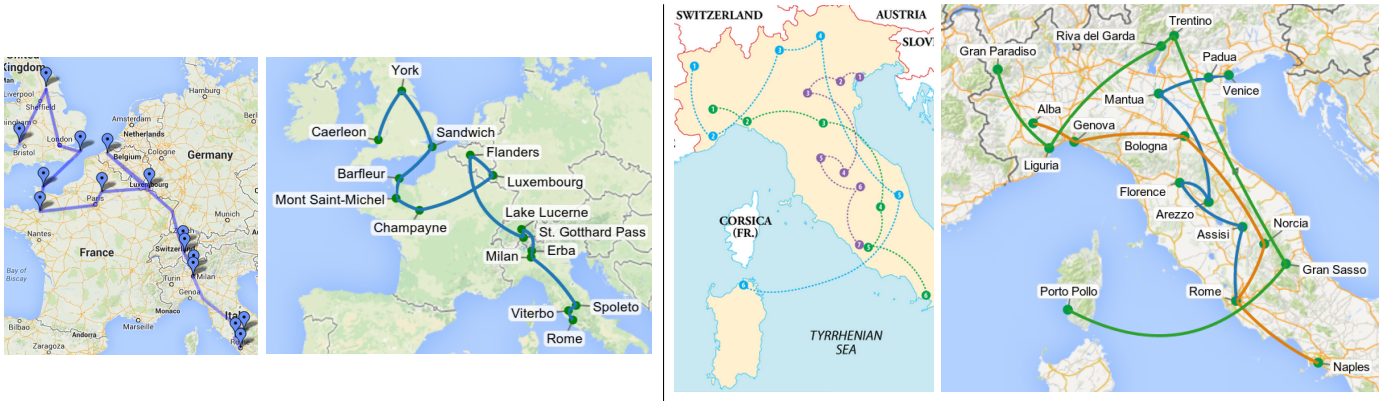
Figure 2 shows the results of our algorithm on six randomly-generated itineraries. The visualizations minimize the least desirable layout factors, with no occluded labels and minimal overlap of labels, edges, and waypoints. Minor layout issues are visible, such as the placement of labels for Stop B and Stop D in the bottom right diagram. Here, a better layout would involve switching the positions of the two labels. However, swapping them would have required several fortuitous parameter changes by the CANDIDATE function in order to climb out of a local minimum in the energy function, which did not occur.

5. DEMONSTRATION SYSTEM

A demonstration system illustrates the effectiveness of our itinerary layout algorithm. It consists of three primary components: (1) a geotagging module, (2) a mapping module, and (3) an itinerary visualization module.

The geotagging module takes an input itinerary, i.e., a set of place names (or *toponyms*) that refer to waypoint locations. It returns geographic interpretations of those place names that have geographic coordinates (i.e., latitude and longitude values). A geotagging [10, 11, 12, 13] step is needed for any textually-specified itinerary to convert the human-readable specification into machine-readable form. Our system has two geotagging modes. The first uses a geotagger service (Google's Geocoder API) to geotag each place name individually. The second uses GeoWhiz, a system based on the structured toponym resolution technique that we have developed for geotagging coherent lists of places [1, 3]. Neither was specifically designed for itinerary geotagging, but both perform well on sample itineraries. However, in some cases, both modes generate results that are unexpected given our knowledge that inputs are itineraries. In the case of Google's Geocoder API, the places lack the general geographic consistency that we expect from itineraries, while for GeoWhiz, the emphasis on consistency can result in incorrect interpretations when the itinerary spans a large geographical area.

The mapping module renders the map base layer, upon which the itinerary visualization is displayed. The system currently allows for Google Maps (with the standard Mercator projection) or one of several static map projections. A challenge for this module is picking appropriate bounds for the geographic window. To avoid a specialized solution for



(a) (b)

Figure 3: Two itineraries taken from image search results and their reproductions using our itinerary layout method. The reproductions are shown to the right of the originals. The map in (a) was created by a blogger to display her European itinerary. The order of stops and precise stop locations are difficult to discern in the original, but our automatically generated version addresses both of these issues by adding labels and using curves for edges. The map in (b) shows three suggested itineraries in northern Italy. Our method supports laying out multiple disconnected itineraries, and the simulated annealing algorithm settles on a layout that avoids label overlap even in a somewhat dense itinerary diagram such as this.

each projection, we use a generic algorithm that projects each waypoint into screen coordinates, computes a bounding box in projected coordinates, then scales and transforms the active region in the projection to fill the available space for the map. The resultant screen locations are used as inputs to the next module, which generates the itinerary layout.

The visualization module takes the projection waypoint coordinates, along with the waypoint names and edge topology, and generates layout parameters d, θ, and r for the corresponding waypoints and edges using the algorithm in Section 4. The system uses the actual screen sizes of waypoint label text to do accurate overlap tests. Once the simulated annealing algorithm's iterations are done, the resultant layout parameters are used to render the itinerary on the map.

The system accepts direct input of itineraries, by allowing users to enter names of waypoints, but any itinerary gathering technique could be substituted, such as extracting itineraries from a Web crawl or supplying structured itineraries based on travel site content. Figure 3 shows two itineraries taken from the internet along with reproduced visualizations generated by our system.

6. CONCLUSIONS AND FUTURE WORK

We presented an algorithm and system for generating automated itinerary visualizations. In the future, itinerary layout could be adapted to allow increased realism through options for paths along great circles (approximating flight paths) or along various land or sea transportation routes. The high-level framework that we introduced allows for the addition of such constraints. Additionally, the layout procedure itself could be adapted to include a force-directed component that performs with lower latency (although force-directed approaches are prone to settling at local minima). Further changes involving spatial distortion around waypoint locations, as in LineDrive [4], could also be used.

7. REFERENCES

[1] M. D. Adelfio and H. Samet. GeoWhiz: Toponym resolution using common categories. In *SIGSPATIAL*, pages 542–545, Orlando, FL, Nov. 2013.

[2] M. D. Adelfio and H. Samet. Schema extraction for tabular data on the web. *PVLDB*, 6(6):421–432, 2013.

[3] M. D. Adelfio and H. Samet. Structured toponym resolution using combined hierarchical place categories. In *GIR*, pages 49–56, Orlando, FL, Nov. 2013.

[4] M. Agrawala and C. Stolte. Rendering effective route maps: Improving usability through generalization. In *SIGGRAPH'01*, pages 241–249, 2001.

[5] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph drawing: Algorithms for the visualization of graphs*. Prentice Hall PTR, 1998.

[6] C. Esperança and H. Samet. Experience with SAND/Tcl: A scripting tool for spatial databases. *JVLC*, 13(2):229–255, Apr. 2002.

[7] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[8] I. Herman, G. Melancon, and M. Marshall. Graph visualization and navigation in information visualization: A survey. *TVCG*, 6(1):24–43, Jan 2000.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[10] M. D. Lieberman and H. Samet. Multifaceted toponym recognition for streaming news. In *SIGIR*, pages 843–852, Beijing, China, July 2011.

[11] M. D. Lieberman and H. Samet. Adaptive context features for toponym resolution in streaming news. In *SIGIR*, pages 731–740, Portland, OR, Aug. 2012.

[12] M. D. Lieberman, H. Samet, and J. Sankaranarayanan. Geotagging: Using proximity, sibling, and prominence clues to understand comma groups. In *GIR*, Zurich, Switzerland, Feb. 2010.

[13] M. D. Lieberman, H. Samet, and J. Sankaranarayanan. Geotagging with local lexicons to build indexes for textually-specified spatial data. In *ICDE*, pages 201–212, Long Beach, CA, Mar. 2010.

[14] M. D. Lieberman, H. Samet, J. Sankaranarayanan, and J. Sperling. Spatio-textual spreadsheets: Geotagging via spatial coherence. In *GIS*, pages 524–527, Seattle, WA, Nov. 2009.

[15] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In *Graph Drawing*, pages 248–261, London, UK, 1997.

[16] S. Reddy, K. Shilton, G. Denisov, C. Cenizal, D. Estrin, and M. Srivastava. Biketastic: Sensing and mapping for better biking. In *SIGCHI*, pages 1817–1820, 2010.

[17] H. Samet, H. Alborzi, F. Brabec, C. Esperança, G. R. Hjaltason, F. Morgan, and E. Tanin. Use of the SAND spatial browser for digital government applications. *CACM*, 46(1):63–66, Jan. 2003.

[18] H. Samet, A. Rosenfeld, C. A. Shaffer, and R. E. Webber. A geographic information system using quadtrees. *Pattern Recognition*, 17(6):647–656, November/December 1984.

[19] T. W. Sederberg and S. R. Parry. Comparison of three curve intersection algorithms. *Computer Aided Design*, 18(1):58–63, Jan. 1986.

[20] J. Wood, A. Slingsby, and J. Dykes. Visualizing the dynamics of London's bicycle-hire scheme. *Cartographica*, 46(4):239–251, 2011.