Stub Bundling and Confluent Spirals for Geographic Networks*

Arlind Nocaj and Ulrik Brandes

Department of Computer & Information Science, University of Konstanz

Abstract. Edge bundling is a technique to reduce clutter by routing parts of several edges along a shared path. In particular, it is used for visualization of geographic networks where vertices have fixed coordinates. Two main drawbacks of the common approach of bundling the interior of edges are that (i) tangents at endpoints deviate from the line connecting the two endpoints in an uncontrolled way and (ii) there is ambiguity as to which pairs of vertices are actually connected. Both severely reduce the interpretability of geographic network visualizations.

We therefore propose methods that bundle edges at their ends rather than their interior. This way, tangents at vertices point in the general direction of all neighbors of edges in the bundle, and ambiguity is avoided altogether. For undirected graphs our approach yields curves with no more than one turning point. For directed graphs we introduce a new drawing style, confluent spiral drawings, in which the direction of edges can be inferred from monotonically increasing curvature along each spiral segment.

1 Introduction

We are interested in visualizing geographic networks given as a graph with fixed vertex coordinates and possibly other attributes. Although, for substantive reasons, there is often a relationship between the graph's adjacency structure and the spatial arrangement of its vertices, straight-line drawings are generally cluttered with areas of high edge-density and small-angle crossings.

A technique to reduce such clutter is edge bundling. Generalizing the idea of edge concentrators [17], edge bundles have been introduced in the context of hierarchically clustered graphs [11]. Sets of related edges are routed so that they meet, run concurrently, and then separate again, where edges are considered related if their projections on the cluster tree share a subpath. Note that the nodes of the cluster tree directly yield shared edge control points. Different bundling strategies have been introduced in force-directed layout of general graphs [12,24,19] and layered layout of directed acyclic graphs [23].

For graphs with given vertex coordinates, relatedness of edges is usually defined in terms of similarity of their straight-line realizations. Examples include similarities obtained from grid approximations [6,16,15], visibility graphs [22], or clusters in the four-dimensional space of pairs of vertex coordinates [9]. In the extreme, bundling techniques operate on the pixel level [25,8,14,27,13].

* Research supported in part by DFG under grant GRK 1024. We are grateful to Sabine Cornelsen for valuable comments and suggestions on earlier drafts of this paper.

S. Wismath and A. Wolff (Eds.): GD 2013, LNCS 8242, pp. 388–399, 2013.

Because of the shared inner segments, it cannot be inferred from the drawing which subgraph of a bipartite clique a bundle actually represents, i.e., we cannot know whether the drawing is faithful [18]. Moreover, having edges meet requires that they deviate from the line through their vertices in a way that has no substantive meaning.

Both these problems can be avoided by bundling edges at their ends rather than in their interior. This idea has indeed been introduced in the context of geographic networks [4,20] and also forms the basis of flow maps [21,5,26] which can be seen as drawings of in- or out-stars.

We present novel such methods for drawing geographic networks with edges bundled at their ends. For undirected graphs, we refine the approach of Peng et al. [20]. Our main contribution is an approach for directed graphs based on a new drawing style for in-/out-stars called *confluent spiral drawings*. Confluent spirals consist of smooth drawings of each bundle in which edge directions are represented by increasing curvature so that no ambiguity is created in a combined drawing of all, say, in-stars of a directed graph.

The remainder of this paper is divided into three sections. In Sect. 2, we outline how edges are assigned to bundles. Our approaches for undirected and directed graphs are then described in Sects. 3 and 4 with a short discussion in Sect. 5.

2 Stub Bundling

We consider geographic networks consisting of a graph $G = (V, E)$ with fixed vertex coordinates $p = (p_v)_{v \in V}$ where $p_v = \binom{x_v}{y_v} \in \mathbb{R}^2$. Coordinates might be defined extrinsically by, say, geographic locations, or derived from, say, a precomputed layout.

Our goal is to route the edges in such a way that readability of the network is improved over the corresponding straight-line drawing. The means in this work are bundled edges, curved routing, and color gradients.

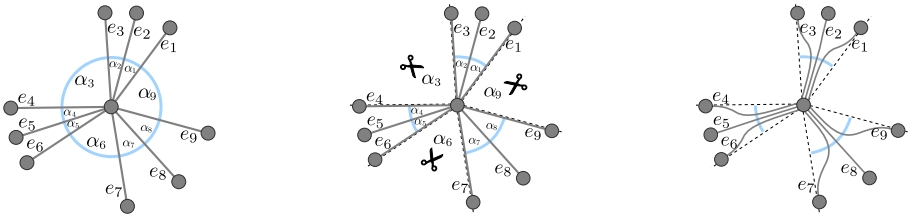
A common objective of edge bundling is to reduce the total length of edges drawn. Since multiplicity along shared paths is ignored, bundling of interior segments of edges is attempted. For geographic networks, however, a more substantively relevant criterion is to be able to read off the general direction in which adjacent vertices are located. To represent this more accurately, and in addition to provide a faithful representation of adjacency, we bundle edges only at their ends, i.e., only with edges that share an endpoint. This type of bundling is referred to here as *stub bundling* to distinguish it from the bundling of edge interiors, or *interior bundling*.

The first step is to find a partition of the edges around each vertex into bundles (see fig. 1). To preserve their general direction, we use the angles between consecutive straight-line edges as our partition criterion. Each bundle is a set of half-edges incident on the same vertex and with similar direction.

For given angles α, γ, an (α, γ)-bundling is a coarsest partition such that

- the angle between any two half-edges in a bundle is at most α, and
- the angle between two consecutive edges in a bundle is at most γ.

Such bundlings are obtained easily by iteratively splitting adjacency lists at maximum angles (between consecutive edges). For each vertex $v \in V$, we start with a bundle containing all incident half-edges. The bundle is split at all occurrences of the maximum angle between consecutive edges in this bundle; in case of equiangular half-edges,



(a) sorting (b) partitioning (c) routing

Fig. 1. Stub bundling: a cyclic sequence of edges is (a) split at maximum angles ($\alpha_3, \alpha_6, \alpha_9$) until (b) the angle range of each bundle is below a given threshold; then, (c) the first segment of each half-edge in a bundle is routed with the same tangent

where the consecutive angles are equal inside a bundle we split symmetrically into two (for bundles with even cardinality) or three (odd cardinality) smaller bundles. In the latter case the resulting bundles may have different (but symmetrical) cardinalities. This process is iterated until we obtain an (α, γ)-bundling. Note that, in contrast to the counterclockwise greedy splitting of [20], we do not accidentally split at small angles and we maintain a higher degree of symmetry.

The entire bundling step is thus carried out in time $\mathcal{O}(m \log \Delta)$, where m is the number of edges and Δ the maximum degree of a vertex, by sorting adjacency lists and splitting them hierarchically. Clearly, other bundling strategies, e.g. also based on distances rather than just directions, may be more appropriate for specific applications. It remains to show how to route the edges beyond the constraint that half-edges in the same bundle share an initial path.

3 Undirected Graphs

After bundling as described in the previous section, we need to decide on two things: in which direction to route the stub of a bundle, and how to connect the two extremal segments of each edge.

To ensure that edges can be followed easily, we allow only one turning point in the routing of an edge. More precisely, we draw each half-edge as a cubic Bézier curve (without turning point) to gain more control over the curve shape. Bézier curves are especially convenient because their tangents at endpoints can be prescribed so that edges in a bundle start in parallel and the two half edges of an edge can be linked smoothly.

Stub directions are determined from the straight-line segment connecting a vertex with the centroid of all neighbors in a bundle. This incorporates not only their directions but also their distances. For the present purpose this is considered a good approximation to the general direction of all edges in a bundle, but more general nodal templates for outgoing edges could be used [3].

We now describe in detail how to choose the control points of the Bézier curves. See fig. 2 for illustration. Let $e = \{v, w\} \in E$ be an edge and let $\Gamma(e, v)$ and $\Gamma(e, w)$ be

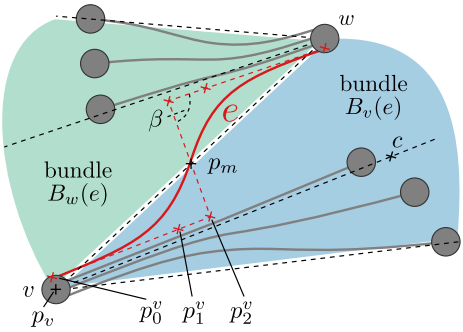


Fig. 2. Control points for routing undirected edge e

the two Bézier curves representing the half-edges of e. Further, let $B_v(e)$ and $B_w(e)$ be the bundles around v and w containing e. Let

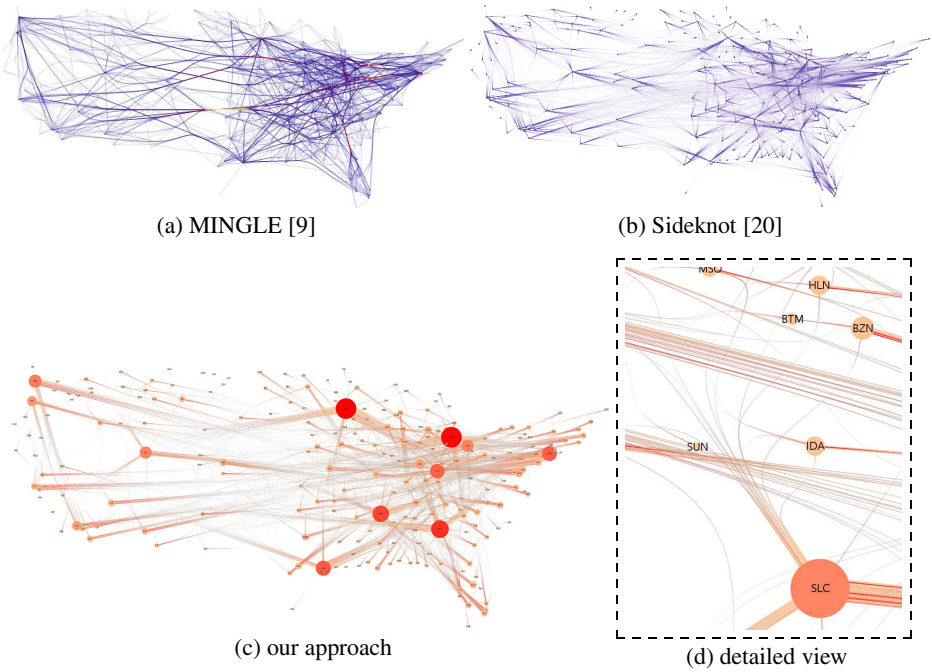
$$p_m = \frac{1}{2}(p_v + p_w) + \left(\frac{|B_v(e)|}{|B_v(e)| + |B_w(e)|} - \frac{1}{2}\right) \cdot t_{\text{shift}} \cdot (p_w - p_v)$$

be the weighted midpoint on the segment between p_v and p_w and $t_{\text{shift}} \in [0, 1]$. More intuitively p_m is closer to p_v if $B_w(e)$ contains more edges then $B_v(e)$. This has the effect that larger bundles have longer parallel parts. Let c be the centroid of the end vertices of the edges bundled in $B_v(e)$ (excluding v). In order to define the control points of $\Gamma(e, v)$ we consider the *baseline* going through p_v and the centroid c. We first compute temporary control points, which lie on the baseline. Later these points will be shifted left and right to obtain a parallel routing.

We choose a branching angle β which is the same for every edge. This angle determines how long the edge will stay with the bundle until it branches off to enter the other bundle, and thus the smoothness of edges. Denote by p_2^v the point on the baseline such that the angle $\sphericalangle(p_v, p_2^v, p_m)$ is β. Another intermediate control point p_1^v is chosen on the segment between p_v and p_2^v with a smoothing parameter $t \in (0, 1)$, i.e., $p_1^v = p_v + t \cdot (p_2^v - p_v)$.

So far we have determined temporary control points p_v, p_1^v, p_2^v, and $\frac{1}{2}(p_2^v + p_2^w)$ for $\Gamma(e, v)$ and symmetrically for $\Gamma(e, w)$. These are refined to avoid overlap without introducing many crossings. Ordering edges around each vertex is a special case of the more general metro-line crossing minimization problem [2] but we find the simple heuristic of ordering stubs in a bundle $B_v(e)$ according to the opposite control point p_2^w to work sufficiently well. Control points are shifted left and right according to this ordering. Determining control points and ordering stubs in the same bundle does not increase the asymptotic running time of $\mathcal{O}(m \log \Delta)$ already caused by bundling.

Stub bundling is motivated by faithfulness and the substantive interest in directions at the ends of edges. Therefore, non-uniform rendering of edges can be used to highlight bundles and reduce the visual dominance of the less important interior of an edge by fading out colors toward the middle of an edge. Note the emphasis this creates in fig. 3 without eliminating the possibility to trace individual edges. In addition to the alternate bundling strategy, non-overlapping stub routing distinguishes our approach from that



(a) MINGLE [9] (b) Sideknot [20]

(c) our approach (d) detailed view

Fig. 3. US airlines graph: In our approach main edge directions and their strengths are visible from an overview perspective (c) but single edges can still be traced in a detailed view (d). Unlike the approach of (b), ours uses parallel routing of edges to facilitate the display of additional data attributes by varying width or color.

of [20] and facilitates the use of different widths and colors for edges in the same bundle to convey additional attribute information such as volume, frequency, time, and so on.

The total bundling process, edge partitioning and control point computation, of the US airlines graph took 0.07 seconds on an Intel Core i7-2600K CPU@3.40GHz with a single core (impl. in Java 6).

4 Directed Graphs: Confluent Spiral Drawings

To visualize directed geographic networks we break the symmetry of the previous approach as arrows and color gradients do not seem to work well for displaying orientation of stub-bundled edges. Depending on the meaning of edge orientations, we bundle only incoming or outgoing stubs. The problem of drawing a directed graph is thus reduced to the problem of drawing one in- or out-star per vertex.

We introduce a new drawing convention for such star-configurations. It is a variation of spiral trees [26] which have been introduced for flow maps [21], but based on confluent logarithmic spirals for smoother appearance and easier inference of orientation.

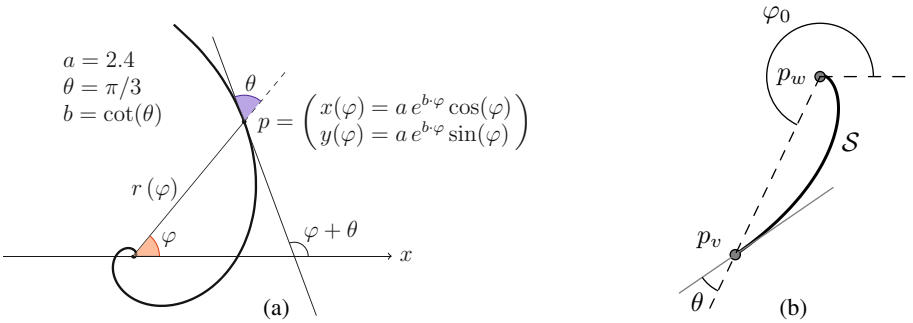


Fig. 4. (a) Logarithmic spiral with angle $\theta = \pi/3$, centered at origin, going through a point p. (b) Spiral used to represent a directed edge (v, w). The constantly increasing curvature on the path from p_v towards p_w unambiguously indicates the orientation.

4.1 Logarithmic Spirals

A *logarithmic* (or *equiangular*) *spiral* is a curve which winds around a center, or vortex, and approaches it with an exponentially increasing curvature. The increase in curvature is determined by a constant θ. In effect, all rays out of the vortex are at angle θ to the tangents of intersection points with the spiral.

Formally, a logarithmic spiral in Euclidean space can be defined in polar coordinates (r, φ) relative to its vortex by $r(\varphi) = a \cdot e^{b\cdot\varphi}$, where $b = \cot\theta$ and $a \in \mathbb{R} \setminus \{0\}$ are fixed, and $\varphi \in \mathbb{R}$. Figure 4(a) shows an example.

We use a sequence of spiral segments to represent an edge between two vertices, and the vortex of each spiral corresponds to a target vertex. We define a spiral segment $\mathcal{S}$ from a start point $p_v \in \mathbb{R}^2$ to an end point $p_w \in \mathbb{R}^2$ with tangent angle $\theta \in (-\frac{\pi}{2}, 0) \cup (0, \frac{\pi}{2})$ as

$$\mathcal{S}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = p_w + |p_v - p_w|\, e^{-|b|\cdot t} \begin{pmatrix} \cos\left(\frac{b}{|b|}(\varphi_0 + t)\right) \\ \sin\left(\frac{b}{|b|}(\varphi_0 + t)\right) \end{pmatrix}, t \in [0, \infty),$$

where $b = \cot\theta$ and $\varphi_0 = \measuredangle\left(\overrightarrow{p_w p_v}, \binom{1}{0}\right)$ is counterclockwise around p_w.

Note that $\mathcal{S}(0) = p_v$. Although the curve has finite length, it never reaches p_w. Practically this is not a problem, since vertex w is represented by a graphical element with non-zero dimensions such as a disc. The spiral $\mathcal{S}$ goes clockwise around p_w if $\theta < 0$ and counterclockwise if $\theta > 0$. Figure 4(b) shows how a logarithmic spiral can be used to represent a directed edge from v to w.

Fig. 5. Two drawings of the same graph. The absence of edge (w_1, w_2) is apparent because the curve from p_{w_1} to p_{w_2} is not smooth.

Confluent Spirals. The term *confluent* was introduced in Dickerson et al. [7] for a drawing style that allows to draw larger classes of graphs in a planar way. We here use it more loosely, not requiring planarity. We say a drawing is a *confluent spiral drawing*, if each edge $e = (v, w) \in E$ is represented as follows:

- There is a continuously differentiable curve from p_v to p_w consisting of logarithmic-spiral segments.
- The logarithmic spiral of the last segment has vortex $\binom{x_w}{y_w}$ and the segment starts at p where
  - either $p = \binom{x_v}{y_v}$ or
  - p lies in the interior of another edge $(v, w') \in E$ with $w' \neq w$.

Furthermore, we do require that the curves representing outgoing edges of the same vertex do not intersect but in a shared prefix. Figure 5 shows a small graph and a corresponding drawing with confluent spirals.

4.2 Determining Confluent Spiral Trees

In this section, we introduce an algorithm to compute a confluent spiral drawing by computing a confluent spiral tree for each vertex. Later, we extend this algorithm to handle obstacles by adding further constraints to it.

The main difference compared to the spiral trees suggested by Buchin et al. [5] is that we want to have confluent drawings, which means that the intersection angle between two spirals is zero. This means that following a path from the root to some other vertex one never has to make a sharp turn. Due to this property it is not possible to apply the method of Buchin et al. [5]. Later the same authors [26] use a spiral tree as a basis to generate flow maps by minimizing a complex cost function to smooth the curves.

In contrast to the previous approach, we require the vortex of spirals not to be on the source but on the target vertex of an edge, which directly results in smooth curves, see fig. 6 for an example.

As a first step we apply the edge partitioning, as described in section 2. The result is for each vertex $v \in V$ a set of bundles containing outgoing edges of v. Each bundle is handled separately. Let B be a bundle with outgoing edges of v. For every edge $e = (v, w) \in B$ we determine a logarithmic spiral $\mathcal{S}$ that is centered at p_w and either starts at p_v or branches out of another spiral in a confluent way such that $\sum_{e \in B} \text{length}(\mathcal{S}_e)$ is minimal. Note that although a spiral never reaches the vortex its length is finite:

$$\text{length}(\mathcal{S}) = \int_0^\infty |\mathcal{S}'(t)|\, dt = ||p_v - p_w|| \frac{\sqrt{1+b^2}}{|b|}.$$

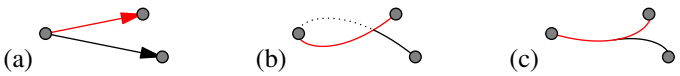


Fig. 6. (a) directed graph, (b) spiral tree approach of Buchin et al. [5]: spiral vortices at source, postprocessing required for smoothness, (c) our confluent spiral tree approach: spiral vortices at targets, smoothness inherent in confluent design

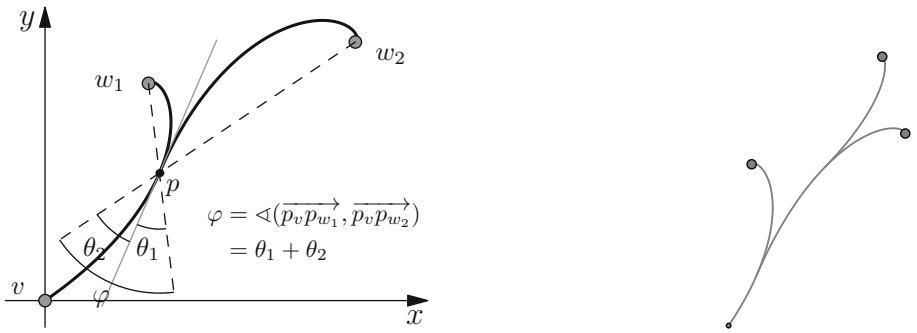


Fig. 7. (a) Construction of a confluent spiral segment with vortex w_2 branching off the parent spiral for (v, w_1) at p. The branching spiral is defined by p, p_{w_2} and the angle θ_2 which is determined by the parent spiral and the angle φ at p. (b) When a parent spiral is changed due to vertex movement, local adaptation by the other spirals is immediate. (NB: in the electronic version, this figure can be animated)

If a spiral $\mathcal{S}$ branches off another spiral $\mathcal{T}$ we refer to the latter as the *parent* spiral. The following heuristic is used to decide on the tree structure. Intuitively, we want edges leading to local targets to branch out earlier in the tree. Thus, consider the edges of a bundle B ordered by the distance of their targets from p_v. Other meaningful orderings, e.g., from data attributes, could be used too here, resulting in different trees.

We start with the first edge $(v, w) \in B$ as the *trunk*. This edge is represented by a logarithmic spiral with a predefined *trunk angle* $0 < \theta_0 < \pm\pi/6$. The upper bound $\pi/6$ ensures that the spiral approaches its vortex more directly, without orbitting around it.

Spiral segments for the other edges are allowed to branch off any already existing spiral segment $\mathcal{T}$ subject to two constraints on the new spiral $\mathcal{S}$:

- $\mathcal{S}$ must have an angle $\theta \in [\theta_{\min}, \theta_{\max}]$ (typically $\theta \in (0, \pm\pi/6]$).
- The tangent of $\mathcal{S}$ at p must have the same slope and direction as that of $\mathcal{T}$.

The branching point p on the parent spiral $\mathcal{T}$ is determined by trying k candidate points $(p_i = \mathcal{T}(i\,\frac{\pi}{k}), i \in 0, ..., k-1)$ on $\mathcal{T}$ and choosing the point that satisfies all constraints and results in the shortest spiral length. Note that $\mathcal{T}$ and p completely determine $\mathcal{S}$ as illustrated in Figure 7(a). If we cannot find a spiral satisfying the constraints, we postpone the current edge temporarily. Our experience so far is that edges need to be postponed rarely so that the overall runtime is in $\mathcal{O}(\sum_{B \in \mathcal{B}} k \cdot |B|^2) \subset \mathcal{O}(k\,n\,\Delta^2)$, where $\mathcal{B}$ is the set of all stub bundles and $n = |V|$.

4.3 Avoiding Obstacles and Crossings

Avoiding edge crossings is very important to reduce visual complexity and improve readability. Furthermore, it is very important that edges not connected to a vertex have a certain distance to that vertex. This can be modelled by placing obstacles on the vertex positions. We extend our framework to deal with obstacles and crossings by adding them as constraints during the search of a parent spiral. We use an R-tree [10] as spatial index and add the vertices with their shape as obstacles into it. The spirals of the

already finished edges are approximated by s line segments and stored in the index too. For a possible branching point p we query the spatial index with s segments of the corresponding spiral to check whether they intersect with obstacles or other edges in the index. The creation of the R-tree index needs $\mathcal{O}(n \log n)$ time on average while maintaining and querying it takes $\mathcal{O}(s \ \log(n + s \ \Delta))$.

The intuitive interpretation of this method is that, if there is an obstacle for a desired branching point we will branch out in an earlier or later phase of the parent spiral to miss that obstacle. Although the resulting spirals will be longer, the readability will be improved. See fig. 8 for an illustration.

4.4 Edge Ordering and Parallel Routing

At this stage we have a confluent spiral tree for each bundle B. To reflect the data, in this case the different number of edges in the bundle, we route them in parallel until they branch to their targets.

Intuitively, walking along the outer contour of our tree gives us the required edge ordering. We determine this ordering by sorting the edges according to the branching point and branching side when traversing the underlying tree from the root vertex. With this ordering we then compute an offset curve to the approximated spiral, which is very similar to polygon offsetting. The offset will determine the thickness of the edge, which in turn can be used to represent, e.g., an edge attribute. See fig. 10 and fig. 9 for an example. Note that after applying an offset the result is not a true logarithmic spiral anymore. In practice this is not a problem as logarithmic spirals are eventually only approximated with cubic splines anyway [1].

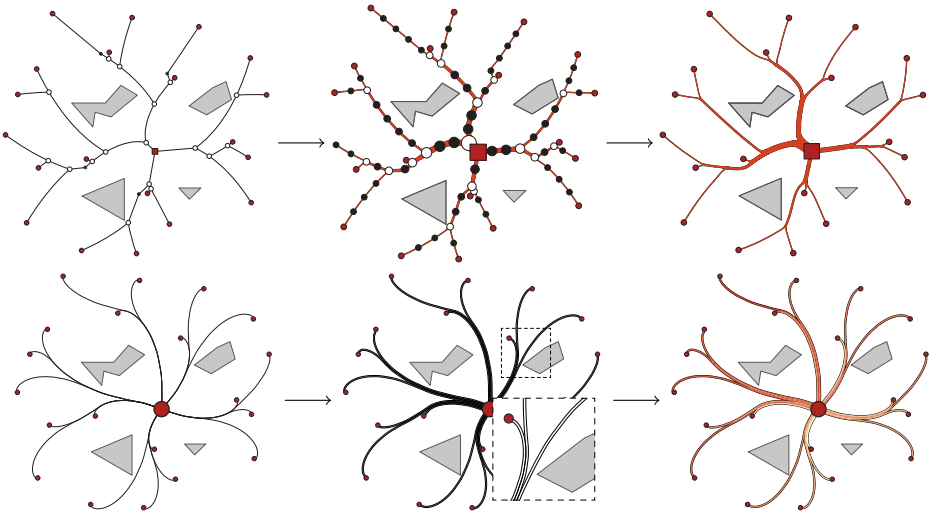
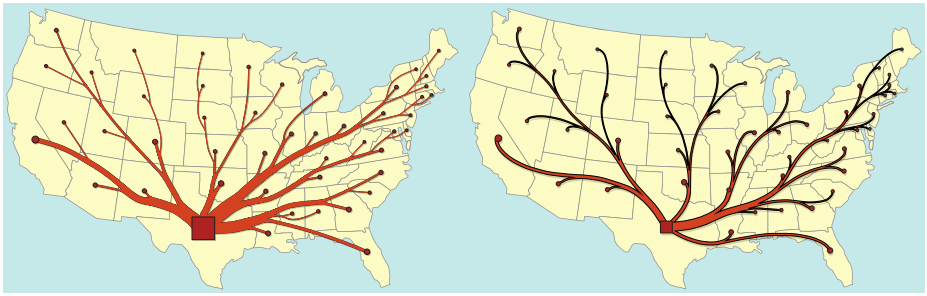


Fig. 8. Avoiding obstacles: Approach of Verbeek et al. [26] (top) and ours (bottom); spirals branch out *smoothely* from trunk at appropriate point to miss the obstacles. Parallel edge routing allows to map an edge attribute to the color; here node distance to the root is mapped (bottom-right).



(a) spiral tree [26] (b) confluent spiral tree

(c) zoom (d) migration volume (e) zoom

Fig. 9. Flow map of migration from Texas (1995-2000). The smooth linkage of confluent spirals (b) eliminates turns and thus not only yields more pleasing drawings but also facilitates the display of edge attributes (d). Note that edge direction can be inferred locally from every segment (e).

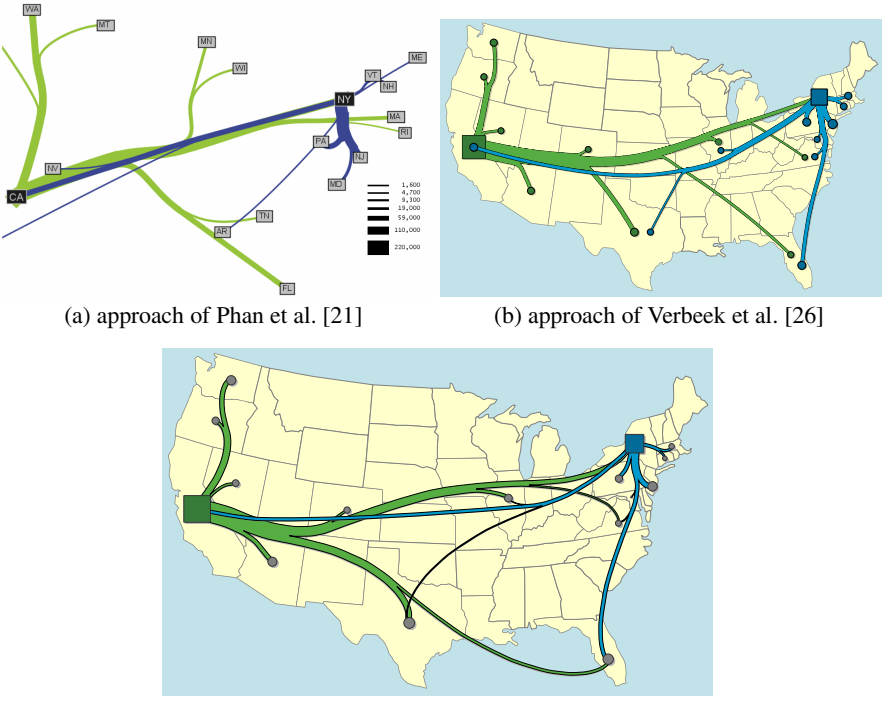
5 Discussion

We have presented drawing styles for the routing of undirected and directed edges in geographic networks using edge bundling at ends rather than interiors, and logarithmic-spiral trees that yield confluent drawings. Their main benefits are

- faithfulness (unambiguous representation of edges)
- stubs point in general direction of destinations
- edge widths and colors are still available for data attributes
- confluent flow maps of in- or out-stars

While initial feedback indicates that confluent spiral drawings are visually appealing, controlled user studies will have to show that they are effective.

As future work we plan to explore other, more data-driven, approaches to partition stubs into bundles, and we would like to prove guarantees on the tree structure of spiral segments and on avoidance of obstacles. For now, our application-oriented implementation is based largely on heuristics, but does layout networks with several thousands of edges essentially at interactive speed.



(a) approach of Phan et al. [21]

(b) approach of Verbeek et al. [26]

(c) our approach

Fig. 10. Flow map of migration to California and New York (1995-2000, top 10 states of origin). Line widths indicate migration volume and are to scale across figures.

References

1. Baumgarten, C., Farin, G.: Approximation of logarithmic spirals. Computer Aided Geometric Design 14(6), 515–532 (1997)
2. Benkert, M., Nöllenburg, M., Uno, T., Wolff, A.: Minimizing intra-edge crossings in wiring diagrams and public transportation maps. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 270–281. Springer, Heidelberg (2007)
3. Brandes, U., Shubina, G., Tamassia, R.: Improving angular resolution in visualizations of geographic networks. In: de Leeuw, W.C., van Liere, R. (eds.) VisSym 2000, pp. 23–32. Springer (2000)
4. Brandes, U., Wagner, D.: Using graph layout to visualize train interconnection data. Journal of Graph Algorithms and Applications 4(3), 135–155 (2000)
5. Buchin, K., Speckmann, B., Verbeek, K.: Angle-restricted steiner arborescences for flow map layout. In: Asano, T., Nakano, S.-I., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 250–259. Springer, Heidelberg (2011)
6. Cui, W., Zhou, H., Qu, H., Wong, P.C., Li, X.: Geometry-based edge clustering for graph visualization. IEEE Trans. on Visualization and Computer Graphics 14(6), 1277–1284 (2008)
7. Dickerson, M., Eppstein, D., Goodrich, M.T., Meng, J.Y.: Confluent drawings: Visualizing non-planar diagrams in a planar way. Journal of Graph Algorithms and Applications 9(1), 31–52 (2005)

8. Ersoy, O., Hurter, C., Paulovich, F.V., Cantareiro, G., Telea, A.: Skeleton-based edge bundling for graph visualization. IEEE Trans. on Visualization and Computer Graphics 17(12), 2364–2373 (2011)
9. Gansner, E.R., Hu, Y., North, S.C., Scheidegger, C.E.: Multilevel agglomerative edge bundling for visualizing large graphs. In: Proc. of the IEEE Pacific Visualization Symposium 2011, pp. 187–194. IEEE (2011)
10. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD 1984, pp. 47–57. ACM Press (1984)
11. Holten, D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. IEEE Trans. on Visualization and Computer Graphics 12, 741–748 (2006)
12. Holten, D., van Wijk, J.J.: Force-directed edge bundling for graph visualization. Computer Graphics Forum 28(3), 983–990 (2009)
13. Hurter, C., Ersoy, O., Telea, A.: Smooth bundling of large streaming and sequence graphs. In: Proc. of the IEEE Pacific Visualization Symposium. IEEE (to appear, 2013)
14. Hurter, C., Ersoy, O., Telea, A.: Graph bundling by kernel density estimation. Computer Graphics Forum 31(3pt. 1), 865–874 (2012)
15. Lambert, A., Bourqui, R., Auber, D.: 3d edge bundling for geographical data visualization. IEEE Trans. on Visualization and Computer Graphics, 329–335 (2010)
16. Lambert, A., Bourqui, R., Auber, D.: Winding roads: Routing edges into bundles. Computer Graphics Forum 29(3), 853–862 (2010)
17. Newberry, F.J.: Edge concentration: A method for clustering directed graphs. SIGSOFT Software Engineering Notes 14(7), 76–85 (1989)
18. Nguyen, Q.H., Eades, P., Hong, S.: On the faithfulness of graph visualizations. In: Proc. of the IEEE Pacific Visualization Symposium. IEEE (to appear, 2013)
19. Nguyen, Q., Hong, S.-H., Eades, P.: TGI-EB: A new framework for edge bundling integrating topology, geometry and importance. In: Speckmann, B. (ed.) GD 2011. LNCS, vol. 7034, pp. 123–135. Springer, Heidelberg (2011)
20. Peng, D., Lu, N., Chen, W., Peng, Q.: Sideknot: Revealing relation patterns for graph visualization. In: Proc. of the IEEE Pacific Visualization Symposium 2012, pp. 65–72. IEEE (2012)
21. Phan, D., Xiao, L., Yeh, R., Hanrahan, P., Winograd, T.: Flow map layout. In: Proc. of IEEE Symposium of Information Visualization 2005, p. 29. IEEE (2005)
22. Pupyrev, S., Nachmanson, L., Bereg, S., Holroyd, A.E.: Edge routing with ordered bundles. In: Speckmann, B. (ed.) GD 2011. LNCS, vol. 7034, pp. 136–147. Springer, Heidelberg (2011)
23. Pupyrev, S., Nachmanson, L., Kaufmann, M.: Improving layered graph layouts with edge bundling. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 329–340. Springer, Heidelberg (2011)
24. Selassie, D., Heller, B., Heer, J.: Divided edge bundling for directional network data. IEEE Trans. on Visualization and Computer Graphics 17 (2011)
25. Telea, A., Ersoy, O.: Image-based edge bundles: Simplified visualization of large graphs. Computer Graphics Forum 29(3), 843–852 (2010)
26. Verbeek, K., Buchin, K., Speckmann, B.: Flow map layout via spiral trees. IEEE Trans. on Visualization and Computer Graphics 17(12), 2536–2544 (2011)
27. Zinsmaier, M., Brandes, U., Deussen, O., Strobelt, H.: Interactive level-of-detail rendering of large graphs. IEEE Trans. on Visualization and Computer Graphics 18(12), 2486–2495 (2012)