Eurographics/ IEEE-VGTC Symposium on Visualization 2009
H.-C. Hege, I. Hotz, and T. Munzner
(Guest Editors)

Volume 28 (2009), Number 3

Force-Directed Edge Bundling for Graph Visualization

Danny Holten[1] and Jarke J. van Wijk[1]

[1]Eindhoven University of Technology

Abstract

Graphs depicted as node-link diagrams are widely used to show relationships between entities. However, node-link diagrams comprised of a large number of nodes and edges often suffer from visual clutter. The use of edge bundling remedies this and reveals high-level edge patterns. Previous methods require the graph to contain a hierarchy for this, or they construct a control mesh to guide the edge bundling process, which often results in bundles that show considerable variation in curvature along the overall bundle direction. We present a new edge bundling method that uses a self-organizing approach to bundling in which edges are modeled as flexible springs that can attract each other. In contrast to previous methods, no hierarchy is used and no control mesh. The resulting bundled graphs show significant clutter reduction and clearly visible high-level edge patterns. Curvature variation is furthermore minimized, resulting in smooth bundles that are easy to follow. Finally, we present a rendering technique that can be used to emphasize the bundling.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and Curve Generation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically Based Modeling

1. Introduction

Graphs are widely used to depict data in which information is comprised of objects and the relationship between these objects. Typical examples are the following:

- Social networks, in which nodes depict individual people and edges depict if people are acquainted;
- Software systems, in which nodes depict source-code elements and edges depict dependency relations;
- Traffic networks, in which nodes depict locations and edges depict the amount of traffic between locations.

Graphs are generally visualized as node-link diagrams, in which dots depict the nodes, joined by lines or curves for the edges. Although node-link diagrams provide an intuitive way to represent graphs, visual clutter quickly becomes a problem when graphs comprised of a large number of nodes and edges are visualized.

This could be remedied by using a different representation instead of a node-link diagram to depict large graphs. For instance, a matrix-based representation can be used as an alternative to node-link diagrams [vH03]. However, although matrix views offer a clean and uncluttered layout, they are less intuitive than node-link diagrams [GFC04]. We therefore want to retain the node-link diagram while reducing the visual clutter that is generally associated with this representation in case of large graphs.

When addressing visual clutter in node-link diagrams, one can focus on the nodes, the edges, or both. We choose to focus on the representation of the edges in this paper, since visual clutter in node-link diagrams is generally the direct result of edge congestion. Furthermore, since node positions often have a clearly defined meaning, e.g., in case of nodes depicting locations in traffic networks, it is not always possible to modify node positions to reduce visual clutter.

Improved rendering and interaction provide two ways to address edge clutter. Rendering can be improved by drawing node-link diagrams at high resolutions while making use of anti-aliasing and alpha blending. Anti-aliasing reduces staircase effects when drawing lines, while alpha blending increases the visibility of individual edges in areas with high edge density. Interaction can be used to address edge congestion by providing a user with the ability to zoom in on certain parts of the graph. Furthermore, interaction techniques such as EdgeLens and Edge Plucking can be used to curve graph edges away from a user's focus of attention without changing node positions [WC05, WCG03].

However, rendering does not take advantage of the possibility to change the shape of edges and interaction is not an option in case of static representations of graphs. It is possible to change the shape of edges by visually bundling them together analogous to the way electrical wires and network cables are merged into bundles along their joint paths and fanned out again at the end, in order to make a tangled web of wires and cables more manageable.

As previous methods have demonstrated, such edge bundling effectively reduces visual clutter. Bundled flow map layouts have been used for single-source graphs, while Hierarchical Edge Bundling has been used to visualize graphs that contain a hierarchical structure [Hol06, PXY*05]. Cui et al. propose a Geometry-Based Edge Bundling (GBEB) method suitable for general graphs [CZQ*08]. GBEB is partially based on previous work by Zhou et al. and Qu et al. and relies on the generation of a control mesh to guide the bundling [QZW06, ZYC*08].

To bundle edges in an intuitive way without requiring a control mesh or hierarchy, we propose to use a self-organizing approach. Edges are modeled as flexible springs that can attract each other while node positions remain fixed. A force-directed technique is used to calculate the bundling. The resulting graphs show significant clutter reduction and clearly visible high-level edge patterns. The variation in curvature is minimized as well, resulting in smooth bundles that are easy to follow. We also provide an interactive and continuous way to change the bundling strength.

Since GBEB is the only edge bundling method for general graphs as far as we know, we compare the results of our Force-Directed Edge Bundling (FDEB) technique to the GBEB results throughout this paper [CZQ*08].

The remainder of this paper is organized as follows. In Section 2 we give an overview of node-link-based visualization techniques and techniques for visual-clutter reduction. Section 3 describes the FDEB technique in detail, followed by Section 4, in which we present example visualizations and compare the results of FDEB to the results of GBEB. Section 4 also introduces a rendering technique to further emphasize the bundling. Finally, Sections 5 and 6 present conclusions and directions for future work, respectively.

2. Related Work

Since we make use of straight-line edges that are already present in the node-link diagram as the starting point for our FDEB approach, we first give an overview of techniques that are commonly used for visualizing graphs as node-link diagrams. This is followed by an overview of general methods that can be used to reduce visual clutter resulting from edge congestion. Current methods for reducing edge clutter by means of bundling are subsequently presented.

2.1. Node-Link-Based Graph Visualization

The most well-known class of node-link-based visualization techniques for general graphs is the class of force-directed methods and its derivatives [FR91, KK89]. An overview of additional techniques for specific kinds of graphs, such as directed acyclic graphs (DAGs) or trees, is presented in [BETT99, HMM00, KW01]. Force-directed methods position graph nodes so that all edges are of more or less equal length and there are as few edge crossings as possible. This is achieved by assigning forces as if the edges are springs and the nodes are electrically charged particles. The entire graph is then simulated as a physical system. Issues with regard to computational complexity and layout stability have recently been treated by various approaches [HK02, HMM00, KCH03]. GPU implementations can significantly reduce computation time even further [FT07].

2.2. General Edge Clutter Reduction

Various techniques are available for the reduction of edge clutter. A general taxonomy of techniques for reducing visual clutter, not specifically concerned with edge clutter reduction, is presented in [ED07].

Visualizing graphs in a clustered way reduces edge clutter by drawing edges between clusters of nodes instead of individual edges between all nodes. Clustered graphs contain a hierarchical component in the form of a recursive clustering structure as well as non-hierarchical connections between the nodes of the clusters. Methods for drawing clustered graphs are presented by Eades et al. [EFL96, Fen97]. A more general survey on drawing clustered graphs is provided by Kaufmann et al. [KW01]. A drawback of these methods is the necessity of (the generation of) a hierarchy and the fact that many low-level edges are fully merged into inter-cluster edges, making it impossible to discern individual edges.

Interaction-based techniques such as EdgeLens and Edge Plucking can be used to curve graph edges away from a user's focus of attention without changing node positions [WC05, WCG03]. However, the use of such techniques is not an option in case of non-interactive graph visualization.

Becker et al. use half-lines for the visualization of edges [BEW95]. A directed edge from source node A to target node B visualized as a half-line only shows the first half of the line between both nodes. Although this reduces edge clutter, it is generally hard to identify the target node of a half-line.

2.3. Edge Clutter Reduction using Edge Bundling

Edge clutter reduction techniques based on edge bundling have recently been gaining interest in graph visualization.

Confluent graph drawing is a technique for visualizing non-planar graphs in a planar way by allowing groups of edges to be merged and drawn together [DEGM03]. However, not every graph is confluently drawable and in general, it appears difficult to quickly determine whether or not a graph can be drawn confluently.

Dwyer et al. propose to add edge routing to force-directed layouts [DMW07]. However, no bundling is performed and

edges are route around nodes in this approach. Hence, minimization of curvature-variation to generate smooth bundles that are easy to follow is not addressed by edge routing.

Flow map layouts use hierarchical, binary clustering on a set of nodes, positions, and flow data to route edges [PXY*05]. As mentioned by Phan et al., the biggest drawback is that all edge splits are binary [PXY*05].

Gansner et al. present a technique that reduces edge clutter by merging groups of edges as bundled splines that share part of their route [GK06]. This method is limited to graphs that use a circular layout, however.

Hierarchical Edge Bundling (HEB) bundles edges together by bending each edge, modeled as a B-spline curve, toward the polyline path defined by the available hierarchy [Hol06]. This method is not applicable to general graphs, since it requires a graph to contain a hierarchy.

Cui et al. present a GBEB method suitable for general graphs [CZQ*08]. However, their method relies on the generation of a control mesh to guide the bundling, which frequently results in bundles that display considerable curvature-variation. This can make such bundles hard to follow (see Section 4.2).

3. Force-Directed Edge Bundling

A straightforward way to bundle edges together in a general graph would be to first create a hierarchy and then use HEB to perform the bundling [Hol06]. However, creating a suitable HEB-hierarchy for a general graph is not trivial. The bundles induced by the hierarchy should faithfully reflect the high-level edge patterns that are present in the graph. It is not evident which hierarchical-clustering scheme or spanning-tree generation method would be suitable for such a task.

Furthermore, a self-organizing, force-directed approach is beneficial for the following reasons. Its behavior is easy to understand because of the straightforward physics model, the core algorithm can be implemented in a few lines of code, and it can easily be extended to accommodate for additional layout criteria (the compatibility measures introduced in Section 3.2 are an example of this).

3.1. Main Technique

The initial input for our FDEB method is a straight-line node-link diagram of a general graph. This node-link diagram can be generated using any available graph layout technique. In case of graphs that represent geographic information such as traffic between locations, the node positions are determined by geographic coordinates instead.

To enable straight-line edges to change shape while bundling, edges are subdivided into segments. Figure 1 shows an example in which two interacting edges P and Q are subdivided using four subdivision points per edge. The position of edge end-points P_0, P_1, Q_0, and Q_1 remains fixed.

For each edge, a linear, attracting spring force $\mathbf{F_s}$ is

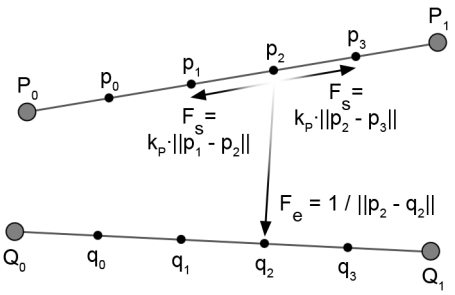

Figure 1: *Two interacting edges P and Q. The spring forces $\mathbf{F_s}$ and the electrostatic force $\mathbf{F_e}$ that are exerted on subdivision point p_2 by p_1, p_3, and q_2 are shown.*

used between each pair of consecutive subdivision points (see Figure 1). Springs between two points are zero-length springs, i.e., springs that exert zero force when they have zero length. A global spring constant K is used to control the amount of edge bundling in a graph by determining the stiffness of the edges. Since edges have different initial lengths and are subdivided into segments (the springs between consecutive subdivision points), a local spring constant k_P is calculated for each segment of edge P. k_P is identical for each segment of P and is calculated as $k_P = K/|P|$(number of segments), where $|P|$ is the initial length of edge P.

Furthermore, an attracting electrostatic force $\mathbf{F_e}$ is used between each pair of corresponding subdivision points of a pair of interacting edges. Thus there are four $\mathbf{F_e}$ interactions in Figure 1: between p_0 and q_0, p_1 and q_1, p_2 and q_2, and p_3 and q_3. Using an inverse-square model instead of an inverse-linear model results in stronger, more localized bundling (see Section 4). In this section the inverse-linear model is used in the figures and equations.

$\mathbf{F_e}$ could also be calculated for each combination of subdivision points (p,q) with $p \in P$ and $q \in Q$. However, this significantly increases the computational complexity from $O(N)$ to $O(N^2)$ for the interaction between a pair of edges with N subdivision points per edge. Furthermore, we have observed that both approaches do not differ much from a visual point of view. We therefore chose the least computationally expensive approach.

During each calculation step of the iterative simulation, the combined force exerted on each subdivision point of each of the edges is calculated. The position of each subdivision point is updated by moving it a small distance in the direction of the combined force that is exerted on it. For a subdivision point p_i on edge P, the combined force $\mathbf{F_{p_i}}$ exerted on this point is a combination of the two neighboring spring forces $\mathbf{F_s}$ exerted by p_{i-1} and p_{i+1} and the sum of all electrostatic forces $\mathbf{F_e}$. It is defined as

$$\mathbf{F_{p_i}} = k_P \cdot (\|p_{i-1} - p_i\| + \|p_i - p_{i+1}\|) + \sum_{Q \in E} \frac{1}{\|p_i - q_i\|},$$

with

k_p : spring constant for each segment of edge P,
E : set of all interacting edges except edge P.

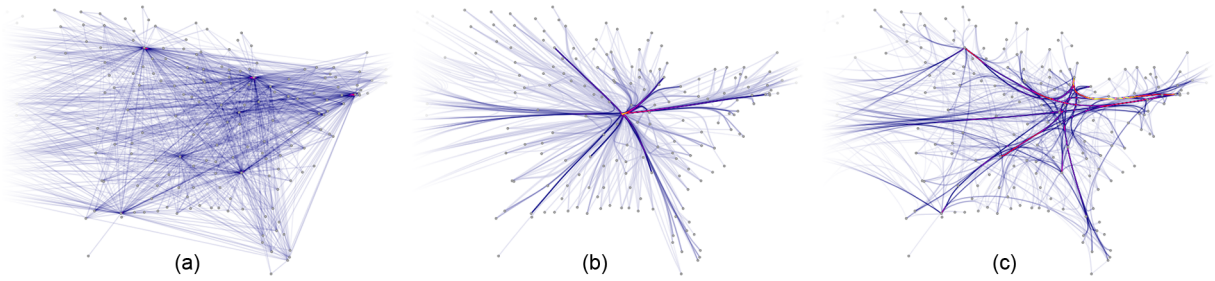

Figure 2: *Part of (a) a straight-line graph that is bundled (b) without and (c) with edge compatibility measures. These measures reduce the amount of bundling between incompatible edges while retaining it in parts of the graph where this is desirable.*

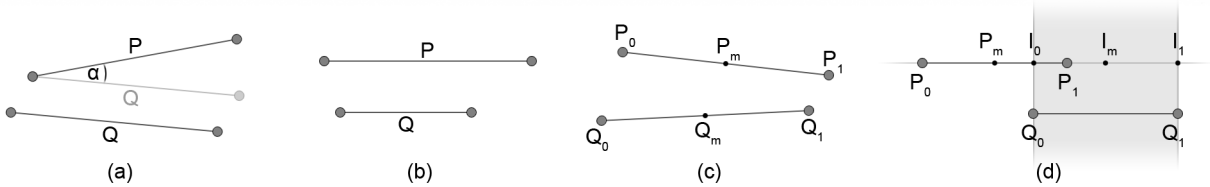

Figure 3: *Geometric concepts and situations necessary to illustrate the edge compatibility measures (a) angle compatibility C_a, (b) scale compatibility C_s, (c) position compatibility C_p, and (d) visibility compatibility C_v.*

3.2. Edge Compatibility Measures

The model presented in the previous section generates an edge-bundled graph, but the amount of bundling is often too high (see Figure 2b). Although this could be remedied by increasing the value of K, i.e., by making edges stiffer, this generally also results in less bundling in parts of the graph where a high amount of bundling is still desirable. To address this, we introduce a set of edge compatibility measures to control the amount of interaction between edges.

In general, edges that are almost perpendicular should not be bundled together. We therefore introduce the concept of angle compatibility $C_a(P,Q) \in [0,1]$ as

$$C_a(P,Q) = |\cos(\alpha)|,$$

with

$$\alpha \quad : \quad \arccos\left(\frac{P \cdot Q}{|P||Q|}\right).$$

This is illustrated in Figure 3a. The larger the angle between edges P and Q, the smaller $C_a(P,Q)$. $C_a(P,Q)$ is 0 if P and Q are orthogonal and 1 if P and Q are parallel.

Furthermore, edges that differ considerably in length should not be bundled together either; doing so might result in noticeable stretching and curving of short edges to accommodate to the shape of long edges, as is illustrated in Figure 4). We therefore introduce the concept of scale compatibility $C_s(P,Q) \in [0,1]$ as

$$C_s(P,Q) = \frac{2}{l_{avg} \cdot \min(|P|,|Q|) + \max(|P|,|Q|)/l_{avg}},$$

with

$$l_{avg} \quad : \quad \frac{|P|+|Q|}{2}.$$

This is illustrated in Figure 3b. $C_s(P,Q)$ is 1 if P and Q have equal length and approaches 0 if the ratio between the longest and the shortest edge approaches ∞.

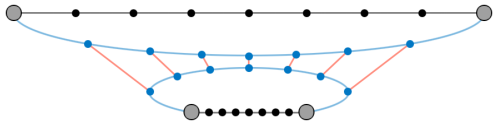

Figure 4: *Bundling edges that differ considerably in length can result in noticeable stretching and curving of short edges. Original edges, curved edges and attracting forces are shown in black, blue, and red, respectively.*

Edges that are far apart should not be bundled together either. We therefore introduce the concept of position compatibility $C_p(P,Q) \in [0,1]$ as

$$C_p(P,Q) = l_{avg}/(l_{avg} + \|P_m - Q_m\|),$$

with

$$P_m \text{ and } Q_m \quad : \quad \text{midpoints of edges } P \text{ and } Q.$$

This is illustrated in Figure 3c. $C_p(P,Q)$ is 1 if P_m and Q_m coincide and approaches 0 if $\|P_m - Q_m\|$ approaches ∞.

However, it is possible that edges are parallel, equal in length, and close together, but should nevertheless have a fairly low bundling-compatibility. The opposite edges of a (highly) skewed parallelogram are an example of this (see Figure 3d). To address this, we also introduce the concept of visibility compatibility $C_v(P,Q) \in [0,1]$ as

$$C_v(P,Q) = \min(V(P,Q), V(Q,P)),$$

with

$$V(P,Q) \quad : \quad \max\left(1 - \frac{2\|P_m - I_m\|}{\|I_0 - I_1\|}, 0\right),$$

$$I_m \quad : \quad \text{midpoint of } I_0 \text{ and } I_1.$$

This is illustrated in Figure 3d. $V(P,Q)$ is the visibility of P to Q and is determined by extending a "band of sight" from Q and calculating the intersection points I_0 and I_1 of this

band with the extended line P. $C_v(P,Q)$ is 1 if P_m coincides with the intersection midpoint I_m (ideal position). $C_v(P,Q)$ becomes 0 if P is moved outside the band of sight along its extended line.

We define the total edge compatibility $C_e(P,Q) \in [0,1]$ between two edges P and Q as

$$C_e(P,Q) = C_a(P,Q) \cdot C_s(P,Q) \cdot C_p(P,Q) \cdot C_v(P,Q).$$

The combined force $\mathbf{F_{p_i}}$ is now redefined as

$$\mathbf{F_{p_i}} = k_P \cdot (\|p_{i-1} - p_i\| + \|p_i - p_{i+1}\|) + \sum_{Q \in E} \frac{C_e(P,Q)}{\|p_i - q_i\|}.$$

Figure 2c shows the effect of the total edge compatibility $C_e(P,Q)$ when bundling a graph of US airlines. The amount of bundling between incompatible edges is significantly reduced while a high amount of bundling is retained in parts of the graph where this is desirable.

3.3. Calculation

An iterative refinement scheme is used to calculate the bundling to improve performance. The simulation starts with an initial number of subdivision points P_0 for each edge and an initial step size S_0. The step size S determines the distance a point is moved at each iteration step in the direction of the combined force that is exerted on it. Furthermore, a fixed number of simulation cycles C is performed. A specific number of iteration steps I is performed during each cycle. I_0 is the number of iteration steps during the first cycle.

After performing a cycle, the number of subdivision points P is doubled and the step size S is halved before initiating the next cycle. The number of iteration steps I per cycle is decreased as well. After experimenting with different values, we settled for $P_0 = 1$, $S_0 = 0.04$, $C = 6$, and $I_0 = 50$. The factor by which I is decreased was set to $\frac{2}{3}$. This leads to the following scheme:

cycle	0	1	2	3	4	5
P	1	2	4	8	16	32
S	.04	.02	.01	.005	.0025	.00125
I	50	33	22	15	9	7

The scheme above performs 141 iteration steps in total and ends with edges that contain 32 subdivision points. This proved to be satisfactory for the collection of graphs that we used as test input for our FDEB method. The graphs were bundled sufficiently and 32 subdivision points were enough to guarantee smooth edges. Furthermore, S_0 was set as high as possible without destabilizing the simulation. Possible oscillating movement of subdivision points was dampened by using a local cooling scheme. Actual running times are given in Section 4 for each of the depicted graphs.

We also found that performance can be significantly increased without compromising the bundling result by using a threshold for the total edge compatibility $C_e(P,Q)$. A pair of edges P and Q is only considered for calculation if $C_e(P,Q)$ is above a certain threshold. The total number of edge interactions often drops to $25\% - 50\%$ of the total as a result of this, even for small threshold values such as 0.05.

As a final post-processing step, a variable amount of smoothing can be applied to all edges by convolving the positions of the subdivision points using a Gaussian kernel. The maximum width of the Gaussian kernel and hence, the maximum amount of smoothing, is determined by the number of subdivision points per edge. A small amount of smoothing gives the edges a less jagged appearance resulting in bundles that are easy to follow. Figure 5 shows the effect of using smoothing amounts of 0%, 25%, and 50%.

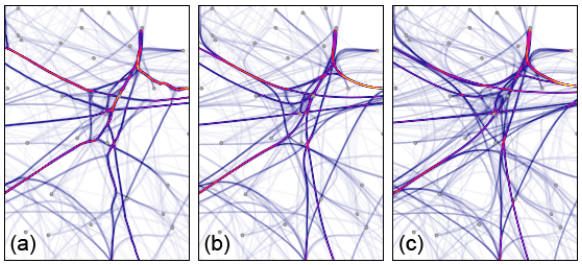

Figure 5: *Part of a bundled graph at (a) 0%, (b) 25%, and (c) 50% smoothing. A small amount of smoothing makes edges less jagged resulting in bundles that are easy to follow.*

4. Results

This section illustrates the results of our FDEB approach. Section 4.1 describes the rendering technique that is used to emphasize the bundling. Example visualizations and a comparison of FDEB and GBEB are provided in Section 4.2. Finally, Section 4.3 presents an interactive and continuous way to change the bundling strength.

4.1. Rendering

A GPU-based, OpenGL rendering technique is used to further emphasize the bundling. The edges are first drawn into a floating-point accumulation buffer to determine the number of edges passing through each pixel (by measuring the amount of overdraw). After this, the minimum and maximum number of edges passing through a pixel is determined. This information is subsequently used to assign a user-defined color gradient to the edges in the accumulation buffer, which can be done in a linear or logarithmic way. The graphs shown here use the default low-to-high color gradient depicted in Figure 6 to perform the gradient-based rendering.

Figure 6: *The default low-to-high color gradient that is used to perform the gradient-based rendering.*

4.2. Visualization Examples and Comparison

The bundled graphs shown in this section depict US airlines and US migration information. All of the visualizations that

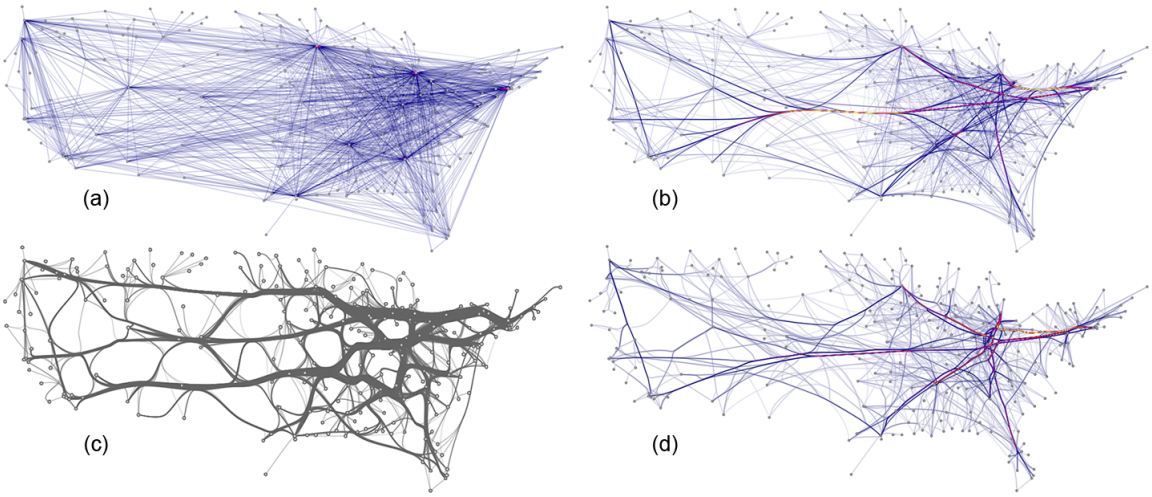

Figure 7: *US airlines graph (235 nodes, 2101 edges) (a) not bundled and bundled using (b) FDEB with inverse-linear model, (c) GBEB, and (d) FDEB with inverse-quadratic model.*

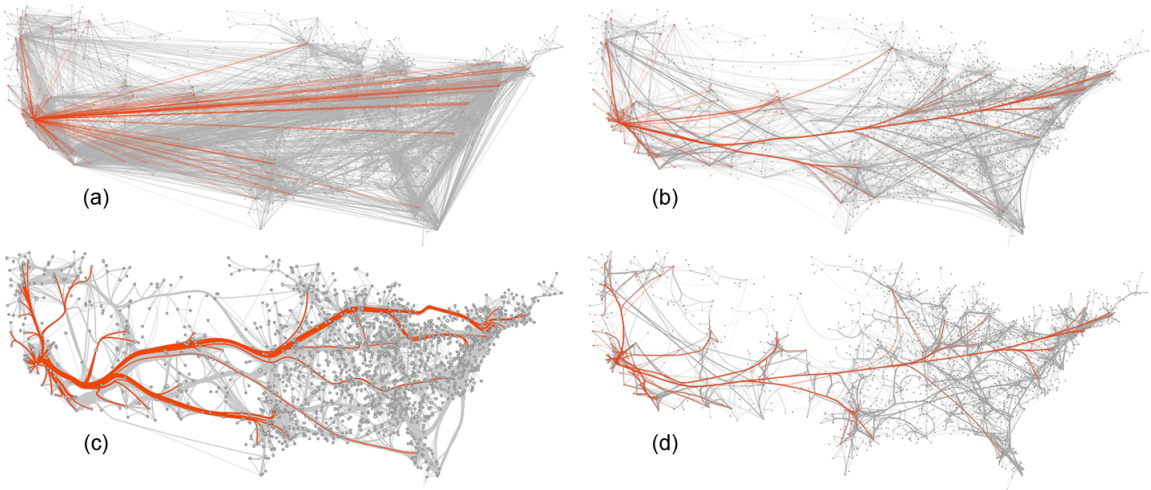

Figure 8: *US migration graph (1715 nodes, 9780 edges) (a) not bundled and bundled using (b) FDEB with inverse-linear model, (c) GBEB, and (d) FDEB with inverse-quadratic model. The same migration flow is highlighted in each graph.*

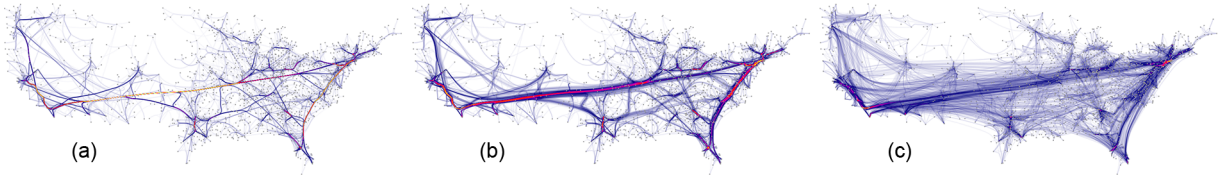

Figure 9: *A low amount of straightening provides an indication of the number of edges comprising a bundle by widening the bundle. (a) $s = 0$, (b) $s = 10$, and (c) $s = 40$. If s is 0, color more clearly indicates the number of edges comprising a bundle.*

we generated use the rendering technique described in Section 4.1. To facilitate the comparison of migration flow in Figure 8, we use a similar rendering technique as the one that Cui et al. [CZQ*08] used to generate Figure 8c.

The airlines graph is comprised of 235 nodes and 2101 edges. It took 19 seconds to calculate the bundled airlines graphs (Figures 7b and 7d) using the calculation scheme presented in Section 3.3. The migration graph is comprised of 1715 nodes and 9780 edges. It took 80 seconds to calculate the bundled migration graphs (Figures 8b and 8d) using the same calculation scheme. All measurements were performed on an Intel Core 2 Duo 2.66GHz PC running Windows XP with 2GB of RAM and a GeForce 8800GT graphics card. Our prototype was implemented in Borland Delphi 7.

Figures 7c and 8c show that the bundling performed by GBEB results in less "webbing"; almost all edges are merged into strong bundles and the space between these bundles is nearly empty. However, GBEB often has to bend edges considerably to bundle them together, resulting in bundles with high curvature-variation (clearly visible in Figure 8c). This makes such bundles hard to follow, i.e., it is not always easy to (roughly) identify the start and end points of edges that comprise a bundle.

The compatibility measures introduced by FDEB prevent incompatible edges to be bundled. As a result, graphs bundled using FDEB display more webbing, since not all edges are bundled. However, edges that are strongly bundled generally comprise bundles that are easy to follow since these bundles display less curvature-variation (clearly visible in Figures 8b and 8d). Furthermore, the use of an inverse-quadratic instead of inverse-linear model results in graphs with more localized bundling and less webbing at the expense of increased curvature (see Figures 7d and 8d).

GBEB is faster in generating its bundled graphs than FDEB: 2.5 seconds (GBEB) compared to 18.8 seconds (FDEB) for the airlines graph and 12.9 seconds (GBEB) compared to 79.6 seconds (FDEB) for the migration graph. Quickly varying the global spring constant K to view its effect on the bundling becomes difficult in case of calculation times in excess of a minute. However, these computational-complexity issues can be addressed using various approaches as described in Section 6.

Finally, the algorithmic pipeline of GBEB is more complex than the force-directed approach used by FDEB. The GBEB phases – graph analysis, control-mesh generation, and edge bundling and smoothing – rely on various additional concepts and techniques such as kernel density estimation, Poisson sampling, constrained Delaunay triangulation, K-means clustering, and a local-smoothing algorithm based on dynamic programming, making GBEB more complex to implement. The core algorithm of FDEB can be implemented in a few lines of code and extending it with edge compatibility measures is straightforward as well.

4.3. Bundle Straightening

A continuous way to interactively change the bundling strength without changing the global spring constant K and recalculating is realized as follows. A bundled edge P can be straightened into its initial, linear shape by straightening each of its subdivision points p_i. A straightened subdivision point p_i' is calculated using

$$p_i' = (1-s)p_i + s(P_0 + \tfrac{i+1}{N+1}(P_1 - P_0)),$$

with

N : number of subdivision points,
s : amount of straightening, $s \in [0,1]$.

Figure 9 illustrates how low values of s give an indication of the number of edges comprising a bundle by widening the bundle. If s is 0, gradient-based rendering provides an indication instead by using color to show the number of edges comprising a bundle. By providing a user with the option to interactively vary s, it is possible to quickly generate bundled graphs with wider bundles, such as those created by GBEB.

GBEB uses wide bundles exclusively to give an indication of the number of edges comprising a bundle, since color is already used by the GBEB transfer-functions to encode edge orientation and edge length.

5. Conclusions

We have presented an edge bundling method for general graphs that does not require the generation of a control mesh or a hierarchy. Our FDEB method uses an intuitive, self-organizing approach to bundling by modeling edges as flexible springs that can attract each other. A single parameter K is used to control the amount of bundling. We have also introduced edge compatibility measures that are used to prevent incompatible edges to be bundled together. The force-directed algorithm used to calculate the bundling is straightforward to implement and provides good results when used in conjunction with the provided calculation scheme.

The resulting bundled graphs show significant clutter reduction and clearly visible high-level edge patterns. Variation in curvature along the overall bundle direction is minimized as well. In conjunction with the smoothing that can be applied, this results in bundles that are easy to follow.

Furthermore, an inverse-linear and inverse-quadratic calculation model have been presented. Use of the latter model results in graphs with more localized bundling and less webbing at the expense of increased curvature. We have also provided a rendering technique to further emphasize the bundling as well as a way to interactively change the bundling strength by means of straightening.

Finally, we have provided example visualizations and compared our approach to GBEB with respect to bundling, performance, rendering, and implementational complexity.

As far as limitations are concerned, we currently consider the calculational complexity in case of graphs with a large number of edges to be the biggest problem of FDEB.

6. Future Work

We consider the improvement of the performance and computational complexity of FDEB, which is $O(N \cdot M^2 \cdot K)$, with N = iterations, M = edges, and K = subdivision points per edge, to be the most important direction for future work. A straightforward way to directly address this is to use a subdivision-based approach such as Barnes-Hut simulation to attain $O(N \cdot M \log M \cdot K)$ complexity [BH86]. To further speed up calculation, a stress-majorization method [GKN04] could be used instead of the current iterative refinement scheme for faster convergence. Finally, a GPU-based implementation can make the computation of FDEB highly parallel, since pairwise edge interactions can easily be calculated

independently [FT07]. Although it may be possible to implement GBEB on the GPU as well, computational complexity is not a primary issue in case of GBEB. A GPU-based implementation is therefore especially advantageous for FDEB.

After experimentation, we fixed certain parameters, e.g., P_0, S_0, C, and I_0, of the calculation scheme presented in Section 3.3 to specific values. These values provide satisfactory results for the graphs that we tested. An important direction for future work is the verification of these values to confirm the suitability of FDEB as a visualization method for general graphs. Additional graphs and graphs of various sizes need to be tested to see if parameters need to be adjusted.

Apart from performing controlled user-experiments in which subjects perform different tasks on a collection of graphs to provide a formal evaluation of FDEB, it might also be worthwhile to directly quantify the quality of the visualizations. Measuring the amount of pixel-based overdraw and empty space for FDEB, GBEB, and straight-line graphs could be used to quantify webbing as well as bundling. Furthermore, edge curvature and length after bundling could be used to quantify the amount of deviation from the original as well as how smooth (and easy to follow) bundles are.

Finally, it might be possible to generate a hierarchy for a general graph by using the edge compatibility measures as criteria for a hierarchical-clustering scheme. This could result in bundles that faithfully reflect high-level edge patterns when such a hierarchy is used in conjunction with HEB.

7. Acknowledgements

We would like to thank Weiwei Cui and his coauthors [CZQ*08] for providing us with high-resolution versions of their GBEB images for comparison purposes (see Figures 7c and 8c). This project is funded by the Netherlands Organization for Scientific Research (NWO) Jacquard program under research grant no. 638.001.408 (Reconstructor Project).

References

[BETT99] BATTISTA G. D., EADES P., TAMASSIA R., TOLLIS I. G.: *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[BEW95] BECKER R. A., EICK S. G., WILKS A. R.: Visualizing Network Data. *IEEE Transactions on Visualization and Computer Graphics 1*, 1 (1995), 16–28.

[BH86] BARNES J., HUT P.: A Hierarchical O(N log N) Force-Calculation Algorithm. *Nature 324*, 4 (1986), 446–449.

[CZQ*08] CUI W., ZHOU H., QU H., WONG P. C., LI X.: Geometry-Based Edge Clustering for Graph Visualization. *IEEE Transactions on Visualization and Computer Graphics (Proc. of INFOVIS'08) 14*, 6 (2008), 1277–1284.

[DEGM03] DICKERSON M. T., EPPSTEIN D., GOODRICH M. T., MENG J. Y.: Confluent Drawings: Visualizing Non-Planar Diagrams in a Planar Way. In *Proc. of the 11th Int. Symposium on Graph Drawing* (2003), pp. 1–12.

[DMW07] DWYER T., MARRIOTT K., WYBROW M.: Integrating Edge Routing into Force-Directed Layout. In *Proc. of the 14th Int. Symposium on Graph Drawing* (2007), pp. 8–19.

[ED07] ELLIS G., DIX A.: A Taxonomy of Clutter Reduction for Information Visualisation. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1216–1223.

[EFL96] EADES P., FENG Q.-W., LIN X.: Straight-Line Drawing Algorithms for Hierarchical Graphs and Clustered Graphs. In *Proc. of the 4th Int. Symposium on Graph Drawing* (1996), pp. 113–128.

[Fen97] FENG Q.-W.: *Algorithms for Drawing Clustered Graphs*. PhD thesis, University of Newcastle, 1997.

[FR91] FRUCHTERMAN T. M. J., REINGOLD E. M.: Graph Drawing by Force-Directed Placement. *Software - Practice and Experience 21*, 11 (1991), 1129–1164.

[FT07] FRISHMAN Y., TAL A.: Multi-Level Graph Layout on the GPU. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1310–1319.

[GFC04] GHONIEM M., FEKETE J.-D., CASTAGLIOLA P.: A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations. In *Proc. of the 2004 IEEE Symposium on Information Visualization* (2004), pp. 17–24.

[GK06] GANSNER E. R., KOREN Y.: Improved Circular Layouts. In *Proc. of the 14th Int. Symposium on Graph Drawing* (2006), pp. 386–398.

[GKN04] GANSNER E. R., KOREN Y., NORTH S. C.: Graph Drawing by Stress Majorization. In *Graph Drawing* (2004), pp. 239–250.

[HK02] HAREL D., KOREN Y.: A Fast Multi-Scale Method for Drawing Large Graphs. *Journal of Graph Algorithms and Applications 6*, 3 (2002), 179–202.

[HMM00] HERMAN I., MELANÇON G., MARSHALL M. S.: Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (2000), 24–43.

[Hol06] HOLTEN D.: Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE Transactions on Visualization and Computer Graphics (Proc. of INFOVIS'06) 12*, 5 (2006), 741–748.

[KCH03] KOREN Y., CARMEL L., HAREL D.: Drawing Huge Graphs by Algebraic Multigrid Optimization. *Multiscale Modeling and Simulation 1*, 4 (2003), 645–673.

[KK89] KAMADA T., KAWAI S.: An Algorithm for Drawing General Undirected Graphs. *Information Processing Letters 31*, 1 (1989), 7–15.

[KW01] KAUFMANN M., WAGNER D.: *Drawing Graphs: Methods and Models*. Springer, 2001.

[PXY*05] PHAN D., XIAO L., YEH R., HANRAHAN P., WINOGRAD T.: Flow Map Layout. In *Proc. of the 2005 IEEE Symposium on Information Visualization* (2005), pp. 219–224.

[QZW06] QU H., ZHOU H., WU Y.: Controllable and Progressive Edge Clustering for Large Networks. In *Proc. of the 14th Int. Symposium on Graph Drawing* (2006), pp. 399–404.

[vH03] VAN HAM F.: Using Multilevel Call Matrices in Large Software Projects. In *Proc. of the 2003 IEEE Symposium on Information Visualization* (2003), pp. 227–232.

[WC05] WONG N., CARPENDALE S.: Using Edge Plucking for Interactive Graph Exploration. In *Proc. of the 2005 IEEE Symposium on Information Visualization, Poster Compendium* (2005), pp. 51–52.

[WCG03] WONG N., CARPENDALE S., GREENBERG S.: EdgeLens: An Interactive Method for Managing Edge Congestion in Graphs. In *Proc. of the 2003 IEEE Symposium on Information Visualization* (2003), pp. 51–58.

[ZYC*08] ZHOU H., YUAN X., CUI W., QU H., CHEN B.: Energy-Based Hierarchical Edge Clustering of Graphs. In *Proc. of the 2008 IEEE Pacific Visualization Symposium* (2008), pp. 55–62.