Edge Bundling

Philipp Möller

January 15, 2010

Abstract

Graphs represented as node-link diagrams are widely used to visualize relationships between entities. For large graphs with many relationships visual clutter increases rapidly and makes it hard to recognize high-level patterns. Two different, general methods to bundle edges are represented and compared. Geometry-Based Edge Clustering uses a control-mesh to guide the edges in the node-link diagram. Force-Directed Edge Bundling is a self-organizing approach where the edges are modeled as flexible springs.

1 Introduction

Graphs play an important role in visualization with typical applications being the representation of social networks, software systems and traffic networks [4]. As such there are generally represented as node-link diagrams where dots or circles depict nodes. The nodes are joined by straight lines or curves. As the scale of problems that are tackled by graph visualization keeps increasing the data is generally comprised of many thousand edges and nodes. Thus reducing the resulting visual clutter is an important research problem.

Visual clutter can be addressed by changing the position of the nodes or edges and improved rendering. The position of the nodes often has some syntactical meaning attached changing the node position might not always be desirable. This is less often the case for the direction and form of the edges. Improved rendering can include the use of alpha blending, anti-aliasing and high-resolutions to reduce visual clutter.

The selection of Force-Directed Edge Bundling [4] and Geometry-Based Edge Clustering [1] for comparison is their diversity in technique but their similarity in preconditions. While FDEB uses a physical, self-organizing model GBEB uses a complex processing pipeline. Both approaches work on general straight line graphs, focus on the edges of graphs and leave the position of the nodes unaltered.

2 Related Work

In addition to the methods presented here many further approaches have been proposed. While the approaches presented here only consider the bundling of edges of a graph many more general techniques for clutter reduction exist. An overview over methods closely related to Geometry-Based Edge Clustering and Force-Directed Edge Bundling is given here while the general approaches according to [3] are mentioned but not explored in greater detail.

Node layout The layout of the nodes comprising the graph can altered to reduce visual cluttering of the edges. The most common approach is very similar to Force-Directed Edge Bundling. A physical model is established between the nodes where edges are represented as flexible springs and nodes as rigid bodies. Those approaches provide good results for small graphs but tend to not scale very well. As the technique is similar to FDEB similar optimizations apply to both which shall be discussed later in this work.

Edge dispersion Dispersing the edges of a graph away from a local center is an approach that can be used to temporarily reduce the edge cluttering in a certain area of the node-link diagram. The locality of this method requires makes an interactive approach necessary and the method cannot be used to show global patterns and high-level structure in a dense graph. Examples include EdgeLens [5] and EdgePlucking [6].

Edge clustering Confluent drawings [2] is a technique for visualizing non-planar graphs in a planar way. Edges are grouped together and are drawn as tracks. The main drawback is the complexity of showing that a graph can be confluently drawn.

Sampling Through selection of a random subset of the available data that is to be represented the appearing of visual clutter is reduced as the amount of data is reduced.

Filtering By filtering the data with a certain criteria the effect of filtering is achieved and the result is a less cluttered visualization with the subset of data that is of interest to the user.

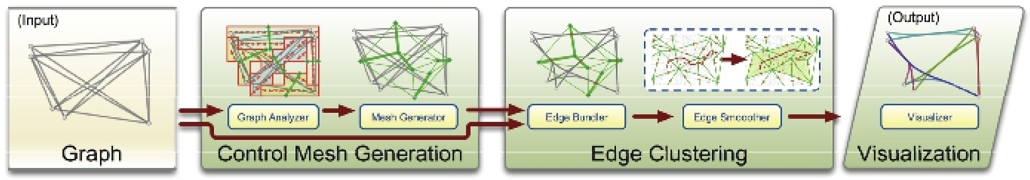

Figure 1: The three stage processing pipeline of Geometry-Based Edge Clustering

3 Geometry-Based Edge Clustering

Geometry-Based Edge Clustering introduced by Cui et al. in [1] aims to transform general straight line graphs into road-map-style graphs. A three stage processing pipeline (Figure 1) is used:

- generation of a control mesh
- edge clustering
- visualization of the resulting graph

The general idea is to analyze the node and edge information to find an underlying distribution pattern and use this pattern to generate a control mesh. The intersections between the control mesh and the original graph are used to generate control points that edges have to pass through. Finally the edge smoother is used to eliminate zigzag curves.

3.1 Control Mesh Generation

The control mesh is the essential part of GBEB as it should reveal the patterns in the graph and thus reduce visual clutter. To maximize the quality of the triangulation it is necessary to consider not only the distribution of the nodes but also the distribution of the edges, their length and, if available, the direction of the edges. Clustering edges with similar length and direction can lead to effective reduction of visual clutter. To generate a mesh that represents those features well three different approaches to control mesh generation are proposed by Cui et al.

Manual Mesh Generation By allowing users to create vertices and edges to mark obvious clusters in a graph a control mesh can be generated through Constrained Delaunay triangulation. While this approach drastically simplifies the generation process it quickly becomes difficult and time consuming with increasing size or density of the graph and not so obvious patterns. It can still be used as a supplementary method for regions where the automatic approaches are unable to generate a satisfactory mesh.

Automatic Mesh Generation To automatically generate a control mesh the bounding box of the graph is computed and divided into a regular grid with a variable resolution. For each grid cell the number of nodes inside the cell and the number of edges passing through the cell are computed. Feature vectors that represent the direction of each passing link are generated and statistical methods are used to analyze if any clustering of feature vector is present in the cell. Cells with clustering of similar feature vectors are grouped together and the feature vectors are merged. Mesh edges are selected perpendicular to the feature vectors. The resulting mesh is post-processed to eliminate unnecessary nodes or generate more nodes and finally triangulated using Constrained Delaunay Triangulation.

Hierarchical Mesh Generation The last approach to control mesh generation changes the way smaller cells are merged with cells with a similar primary direction. A discrete angular threshold between directions can be used to identify cells with merging and thus a discrete level-of-detail can be achieved. A continuous level-of-detail is achieved by selecting the most fitting cells for merging one by one. The resolution of the grid provides an alternative method to create level of detail.

3.2 Edge Clustering

The second step in the pipeline is edge clustering. It consists of two separate stages: the actual edge clustering and the edge smoothing.

Edge Clustering Edge Clustering is straight forward: The K-means clusters of the intersections between the graph edges and control mesh edges are calculated and the edges are forced to pass through the mean of its cluster in the control mesh edge.

Edge Smoothing After the edge clustering process some edges may show a distinctive zigzag pattern. While zigzag edges are not visually pleasing they also tend to indicate the wrong direction of the original edge and thus do not represent the data correctly. Edge smoothing remedies this by introducing a quality metric for polylines that measures how well a polyline represents the original straight line. The metric can then be used to identify lines that do not meet a certain user-defined threshold of quality. The metric is defined as follows:

$$Q(e) = \alpha Q_{angle}(e) + \beta Q_{distance}(e)$$

with

$$Q_{angle}(e) = -\sum_{i=3}^{n} \gamma_i \cdot |\Delta_i|$$

It is assumed that e consists of n segments and $(n-1)$ control points. Δ_i is the angular difference between the surrounding points. γ indicates whether there is a zigzag or direction change at it.

$$Q_{distance}(e) = -\sum -i = 1^{n-1} D_l$$

which describes the distance variation between the curved edge e and the straight line $e\prime$. α and β represent the corresponding weight for each term. The more strict the metric should be, the larger the values of α or β should be. The general idea of the smoothing algorithm is to reroute the line through other control points until it overcomes the threshold set by the metric.

3.3 Visualization

Cui et al. try to further enhance the quality of the resulting graph by using various methods of visualization such as color and opacity enhancement, mesh adjustment and animation.

Color and Opacity Enhancement After edge clustering different patterns of the graph may still not be visible in the resulting graph but are available as data. Through mapping certain information that is known (the number of original lines represented by each polyline segment, the distance of a polyline segment to the original lines) to color and opacity values the visible difference between polylines with different properties can be enhanced.

Mesh Adjustment The manual method for control mesh generation presented in 3.1 can be adapted to be used to refine a resulting graph by adjusting the position of mesh vertices, splitting mesh edges or subdividing triangles. If this step requires the re-invocation of the whole processing pipeline for the complete graph or just a subset is not explored.

Animation Different schemes for animation are proposed. The transformation of the straight lines to polylines and the subsequent merging of those polylines can be shown at discrete points in the calculation. This outlines the mechanisms of the algorithm and gives cues to the user where a different control mesh might produce better results.

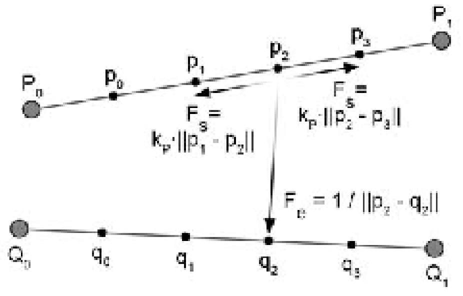

Figure 2: Two interacting edges P and Q. The spring forces F_s and the electrostatic force F_e that are exerted on subdivision point p_2 by p_1, p_3 and q_2 are shown.

4 Force-Directed Edge Bundling

The method used by Holten et al. in [4] is based on modeling a physical system on top of node-link diagrams and using this relationship to let the diagrams organize themselves according to the physical system.

In addition to the simple physical properties various metrics are introduced to measure the compatibility between edges. This additional information is then used to improve the resulting graph layout.

4.1 Main Technique

Each edge is subdivided in into line segments and for each pair of consecutive subdivision points a linear, attracting spring force $F_s = k_p * ||p_1 - p_2||$ where $k_p = K/|P|$ is the local spring constant with K as the global spring constant, where a higher K represents higher stiffness of the edges, and P as the length of each line segment. An electrostatic force F_e is used for each pair of corresponding subdivision points. The choice to only calculate F_e for corresponding points is purely practical. Calculating the electrostatic force for every permutation of subdivision points would increase the computational complexity further and Holten et al. claim that there is no additional clutter reduction gained through the second approach. (Figure 2)

The actual simulation of the physical system is iterative. For every subdivision point the exerted force is calculated and the point is moved by a

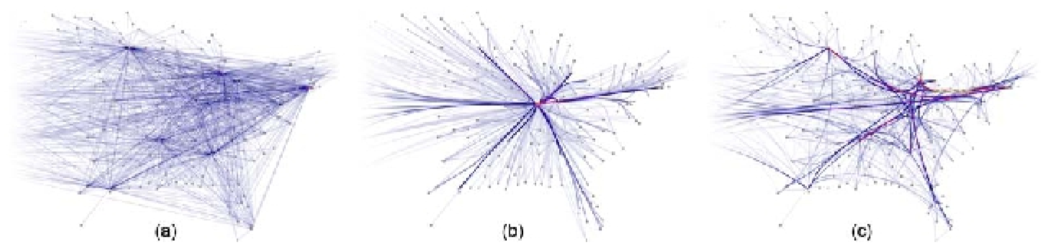

Figure 3: (a) original graph (b) without and (c) with edge compatibility measures.

small value. The exerted force is defined as

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

with
F_{pi}: combined exerted force on subdivision point p_i on the edge P,
E: set of all interacting edges except edge P,
p_{i-1} and p_{i+1}: neighboring subdivision points to p_i on P.

4.2 Edge Compatibility Measures

The iterative approach presented in the last paragraph of section 4.1 results in very strongly bundled edges as illustrated in Figure 3b. The authors propose find edges that are compatible for merging by classifying them with various metrics relative to each other. Those measures are:

Angle Compatibility Edges that are nearly perpendicular should not be bundled. This is calculated through

$$C_a(P,Q) = |\cos(\arccos(\frac{P \cdot Q}{|P| \cdot |Q|}))|$$

Scale Compatibility Edges of significantly different lengths should not be bundled together. This is calculated through

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

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

Position Compatibility Edges that are very far apart from each other should not be bundled together. This is calculated through

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

with

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

Visibility Compatibility There are cases where edges that are parallel, equal in length and close together should not be bundled. An example is a skewed parallelogram. The visibility of edges can be calculated through

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

with

$$V(P, Q)\colon \max(1 - \frac{2||P_m - I_m||}{I_0 - I_1||}, 0),$$
$$I_m\text{: midpoint of } I_o \text{ and } I_1.$$

This defines a "band of sight" between the two edges and the larger the band of sight the larger the value of C_v.

The four presented compatibility measures yield values in the range of $[0, 1]$ where larger values indicate larger compatibility. Total edge compatibility $C_e(P, Q)$ is thus the product of all measures and again a value ranging from 0 to 1. With edge compatibility defined the calculation of the combined edge force is altered F_{pi} to reflect those changes.

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

The effect of this adaptation can be seen in Figure 3c.

4.3 Calculation

To simulate the physical system iterative refinement is used. An initial number of subdivision points P_0 , an initial step size S_0, which determines the displacement of an edge in each step, and a fixed number of simulation cycles

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	8	7

Figure 4: The change of the initial values with each cycle.

C are chosen. Figure 4 represents the change of the values during each cycle. The start values were determined by Holten et al. through experimentation and might be different for other data sets. I is reduced by $\frac{2}{3}$ with each cycle.

A smoothing scheme is also presented: convolving the positions of the subdivision points using a Gaussian kernel.

4.4 Rendering

Holten et al. improve the rendering of the resulting image by measuring the overdraw in each pixel and assigning a either logarithmic or linear low-to-high color gradient to individual line bundles and thus improve visibility of dense edge bundles.

5 Comparison

The Figures 5 and 6 provide representative examples of each presented algorithm for a large data set.

It seems that the bundling of Geometry-Based Edge Bundling is stronger and results in less "webbing". Strong bundles that include almost every edge in the area were generated. To achieve this GBEB has to bend edges significantly in some cases. Also the additional visualization methods that are employed effectively highlight strong bundles. Additionally there is a significant loss of edge properties neither the original direction nor the original length of the edges is preserved.

Force-Directed Edge Bundling tends to generate are more webbed graph. There is a lot less curvature in the bundled graph in comparison to the GBEB approach which can be attributed to the edge compatibility measures. Especially in 6 c and d a huge difference between both approaches is visible. FDEB generates better visible bundles in the right part of the graph while there is no clear structure visible in GBEB result.

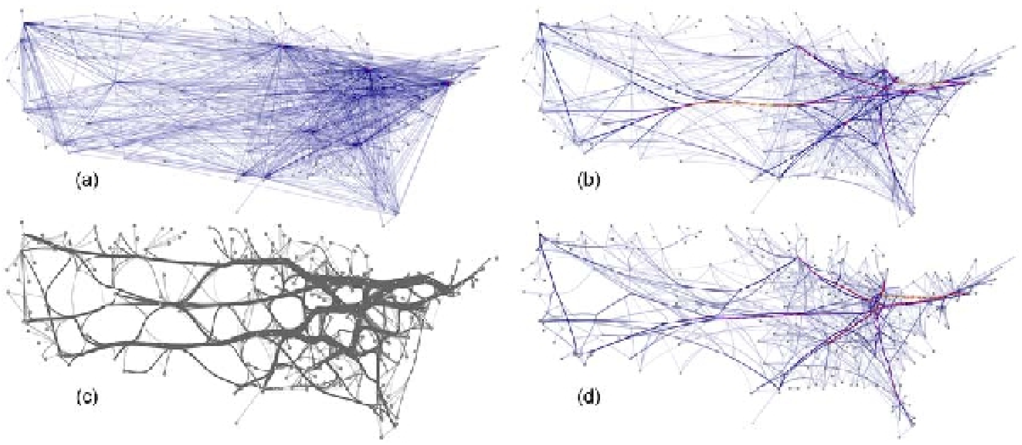

Figure 5: US airlines graph, (a) unbundled (b) bundled with FDEB inverse-linear model (c) GBEB (d) FDEB with inverse-quadratic model.

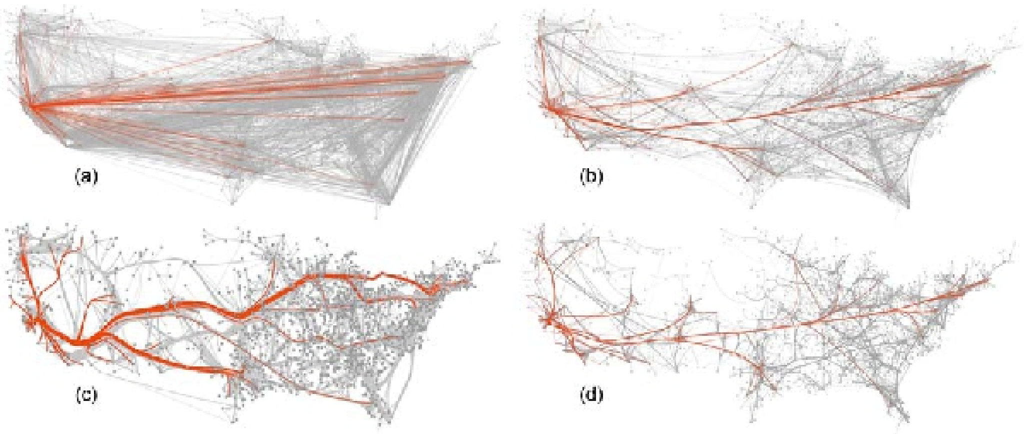

Figure 6: US migration graph, (a) unbundled (b) bundled with FDEB inverse-linear model (c) GBEB (d) FDEB with inverse-quadratic model.

Performance wise GBEB outperforms FDEB. The approach by Cui et al. is, according to benchmarks from Holten, 6 to 7.5 times faster. This could come as a surprise to some considering the different complexity of the methods. GBEB utilizes a complex 3 stage process with various possible methods for each stage and a huge amount of additional concepts while FDEB introduces 4 quality measures and a simple physical model with the only variable being K.

Tuning the result of each algorithm is easier with GBEB. Means to change the control mesh generation process by manual intervention or different variables are provided and it is possible to heavily influence the resulting graph. FDEB on the other hand is only controlled through the global spring constant K which significantly limits the possibilities of adapting the result.

Animation techniques are only provided with GBEB but a similar system could be used with FDEB to show the resulting graph after each iteration.

6 Conclusion and Future Work

Both presented algorithms produce comparable results through completely different means and differ. There is no test case where the results of either are significantly worse than the other. Still there is a room for improvement.

Both Cui et al. and Holten et al. consider GPU implementation of their algorithms possible and expect a considerable improvement. While speed seems to be primarily a concern for the FDEB approach it could possibly benefit much more from a parallel implementation as the algorithm itself has concurrent properties and is simpler to implement. Different parts of GBEB are fit for parallelization. This includes the K-means clustering, line-triangle intersections and Delaunay triangulation.

Cui et al. suggest that improving the transfer function could further enhance the quality of the rendering. Improving the method of control mesh generation could also result in much better results and the results of non-triangular control meshes are currently explored.

While individual parts of GBEB could benefit from a parallel implementation computational complexity is less of a concern for this approach. On the other hand Holten et al. consider reducing the computational complexity of FDEB through various techniques a critical point for further research.

References

[1] Weiwei Cui, Hong Zhou, Huamin Qu, Pak Chung Wong, and Xiaoming Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14:1277–1284, 2008.

[2] Matthew Dickerson, David Eppstein, Michael T. Goodrich, and Jeremy Yu Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. *CoRR*, cs.CG/0212046, 2002.

[3] Geoffrey Ellis and Alan Dix. A taxonomy of clutter reduction for information visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 13:1216–1223, 2007.

[4] Danny Holten and Jarke J. van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, June 2009.

[5] Nelson Wong, M. Sheelagh T. Carpendale, and Saul Greenberg. Edgelens: An interactive method for managing edge congestion in graphs. In *INFOVIS*, 2003.

[6] Nelson Wong and Sheelagh Carpendale. Interactive poster: Using edge plucking for interactive graph exploration. In *IEEE Infovis Poster Compendium*, pages 51–52. IEEE Press, 2005.