# Edge bundling

Group 4

David Fröhlich, Markus Kammerhofer, Samuel Kogler and Stephan Stiboller

706.057 Information Visualisation SS 2017
Graz University of Technology

17 May 2017

## Abstract

Graphs with many nodes and many edges are often cluttered when visualized. Edge bundling is one technique to cope with this problem. In this survey, an overview of different edge bundling techniques is provided. In the first part, cost-based, geometry-based and image-based edge bundling variants are presented and discussed.

The second part of this work is about edge bundling software tools which implement the techniques shown in the first part. In addition, a comprehensive investigation of performance, quality and applications of these tools is conducted.

In the end, some concluding remarks are provided to summarize best practices and the overall value of edge bundling.

# Contents

# List of Figures

# Chapter 1

# Introduction

An very basic problem with visualizations of dense data with a very high number of relations, edges, is that this mass of edges can produces immense visual clutter as illustrated in Figure 1.1 (a) and Figure 3.5. This visual cluttering can make a graph nearly impossible to analyze. Patterns within the graph structure can be obscure or completely hidden and important connections can be easily overlooked. There are various proposed approaches to address this problem of visual edge cluttering which is prevalent in dense graphs. The approach this work focuses on is called edge bundling.

Edge bundling techniques are designed to visually bundle together similar edges to reduce the visual clutter within a graph. This bundling can help to highlight important edge patterns and often makes it easier to spot interesting connections or important data. Figure 1.1 (b), (c) and (d) illustrate the visual improvement when applying a few similar edge bundling techniques to an unbundled graph.

Of course there are also downsides to edge bundling. For example edge bundling can make it considerably harder or depending on the technique to follow edges from their start towards their destination. This happens because depending on the edge bundling technique used multiple edges might be partially merge which makes it impossible to distinguish them.

This survey paper is divided into two parts. The first part is the theoretical part and serves as a taxonomy of different edge bundling techniques, including a detailed description of each of these techniques and algorithms. There are three main approaches introduced, cost-based, geometry-based and image-based variants. Cost-based variants can additionally split up by the variable, which should be minimized, either ink or energy. Geometry-based variant are constructing a grid where the edges are routed through and then are smoothed. Image-based variants take advantage of the parallelism of todays graphics processing units and have therefore a very good performance.

In the second part, the practical aspect of this seminar paper is covered. Different software tools which implement methods described in the first part are presented. For the evaluation, a dataset with many nodes and edges, including geographical information was chosen. The geographical information about the nodes is important to get comparable results from the different implementations. The dataset has more than 1300 nodes and over 21000 edges. The dataset provides information about US airports and their worldwide connections. It has to be pointed out, that the different tools and implementations have most likely totally different applications. As an example, Tulip is designed to analyze data by applying different filters and bundling techniques in a graphical user interface. Whereas the JavaScript implementations are designed for being embedded in web pages, whereby this kind of usage is only possible with small data sets according to performance issues. In addition to that, also a command line tool is presented in this part. Six different tools in total are described in the tool section.

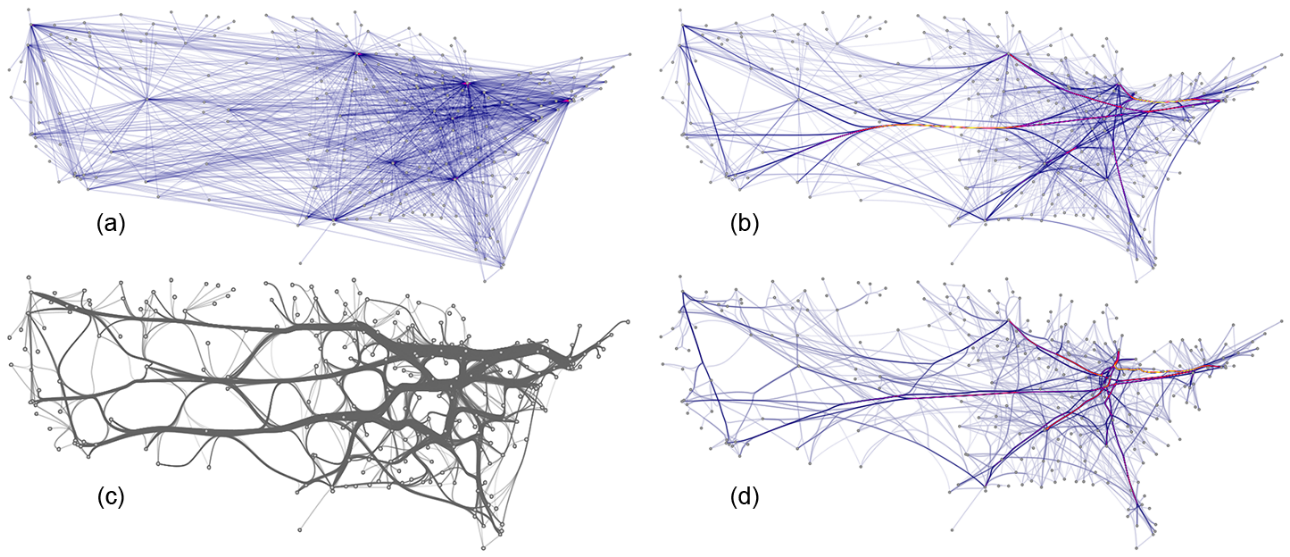


**Figure 1.1:** (a) unbundled graph, (b) FDEB linear, (c) GDEB, (d) FDEB quadratic.
[Images extracted from Holten and van Wijk [2009].]

# Chapter 2

# Variants

## 2.1 Cost-based

Cost-based edge bundling approaches can be divided into two basic categories. The first category is ink minimization which focuses on minimizing the amount of ink used and the other category is energy minimization, which minimizes the energy function of a spring model to determine the shapes/curvature of the edges. Most cost-based edge bundling techniques segment edges to generate control points. The positions of those control points are then optimized in a way that bundles similar edges. Edge similarity is often defined based on proximity, visibility, angle and scale. Often curvature constraints are used to improve bundling quality in addition the edge similarity.

### 2.1.1 Ink Minimization

Edge bundling with ink minimization tries to reduce the amount of ink used to draw the curved edges of a graph.

One technique for edge bundling in circular graph layouts based on ink minimization was proposed by Gansner and Koren [2007]. In this approach the edges are segmented to generate two control points which control the shape(curvature) of the edge, in addition to the start and end points of every edge. During the edge bundling, the positions of generated control points are optimized using ink minimization, such that similar edges are drawn closer to each other. Figure 2.1 shows an example of a circular graph layout before and after edge bundling.

Another edge bundling technique which utilizes ink minimization was proposed by Pupyrev et al. [2011]. It focuses on layered graph layouts and minimizes their global ink usage. Similar to the above mentioned approach by Gansner and Koren [2007], the edges are segmented and two or more control points are generated. Control points of similar edges are then grouped together and their positions are optimized to achieve global ink minimization. The proposed method works well for layered graph layouts but might be significantly slower for general graph layouts. This results in the need for a balancing between bundling quality and time investment.

Gansner et al. [2011] introduced a multilevel agglomerative edge bundling method for general graph layouts that utilizes ink minimization combined with curvature constraints and a proximity graph to achieve better performance than other ink minimization based approaches. The algorithm is described in more detail in the section below.

#### Multilevel Agglomerative Edge Bundling

The intuition of this algorithm is to have a better performance than other edge bundling algorithms, like geometry-based algorithms and algorithms, which are using all-to-all compatibility calculations, which are not scalable. The basic idea described by Gansner et al. [2011] is:

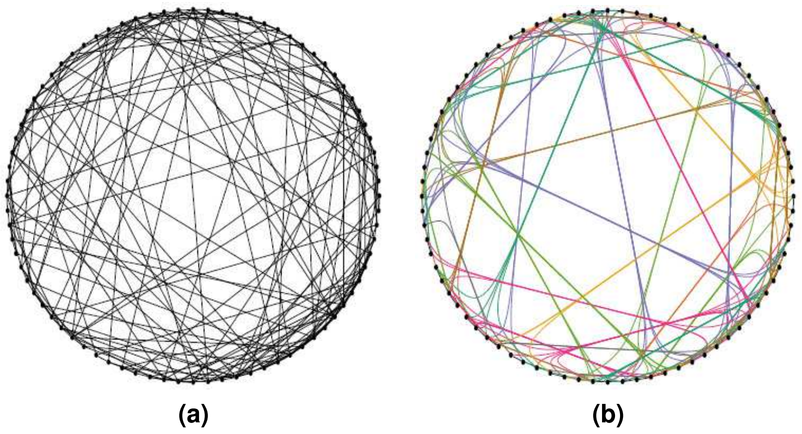


**Figure 2.1:** Circular graph layout before (a) and after (b) edge bundling.
[Image extracted from Zhou et al. [2013].]

> [. . . ] to mimic what a human operator would do when faced with the task of bundling a mass of electrical wires: identify wires that have similar start and end points; merge them; check if additional wires can join existing bundles, or need to start their own; and repeat this process.

To reach that goal, multilevel agglomerative edge bundling uses a simple compatibility measure, where each edge is treated as a vector in 4-dimensional space ($v_{1x}$, $v_{1y}$, $v_{2x}$, $v_{2y}$). The measure identifies two edges to be close, if their Euclidean distance in the 4-dimensional space is small. With a kd-tree it is possible to find the k nearest neighbors in time $k\log(|E|)$ per edge. This leads to an overall time complexity of $k|E|\log(|E|)$ for all edges. The graph, which is created in the 4-dimensional space is called the edge proximity graph. In this graph each edge in the original graph is represented as a node in the edge proximity graph. This configuration is pointed out in figure 2.2a and figure 2.2b.

The next step is the agglomerative bundling. The constructed edge proximity graph guides the bundling decisions. The ink saving for each edge bundled with each of its neighbors is calculated and the best is chosen by the following formula: ink(e(u)) + ink(e(v)) - ink(e(u) ∪ e(v)). The result of this step is shown in figure 2.2c.

In the next step the nodes of the edge proximity graph are merged (see figure 2.2d), so that each node represents bundled edges instead of single edges. The bundling process described in the previous section is now repeated until no further ink saving can be achieved. This leads to the graph in figure 2.2e.

After one step of this bundling process, each edge is represented by a chained line with at most three lines. At this point recursion is used. Bundled straight segments are now used as new edges. On these new edges the bundling process is applied again and again. To avoid excessive bending of strongly bundled edges, a weight measure is added to each edge, according to the number of bundled edges. And the ink saving is calculated in proportion to this weight. The graph after one recursion step is shown in figure 2.2f.

There are two additional remarks to this algorithm, which are not discussed in detail:

- Finding optimal meeting points
- Defining a maximum turning angle to avoid large curvatures for edges that are far away. Consequences of different angle settings are shown in section 3.6 and in figure 2.2h.

### 2.1.2 Energy Minimization

Energy minimization in edge bundling is based on the minimization of energy in spring models and is also referred to as force based edge bundling. Different variants of energy/force based edge bundling have been

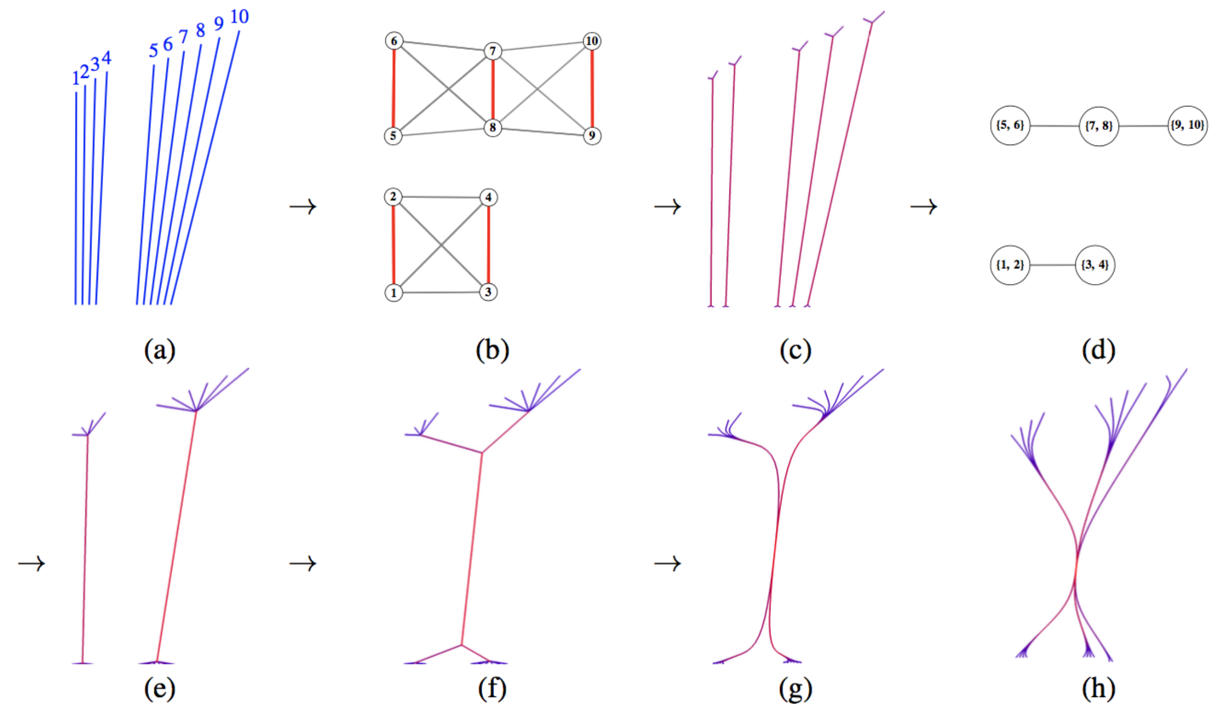


**Figure 2.2:** (a) Original edges. (b) Edge proximity graph (k = 3). (c) Graph after one level of agglomerative bundling. (d) Merged nodes of edge proximity graph. (e) Graph after multilevel agglomerative bundling. (f) Graph after one recursion step. (g) Rendered using splines. (h) With limited turning angle and spline rendering.
[Image extracted from Gansner et al. [2011].]

proposed for different types of visualizations. In the section below a few well known force based edge bundling methods, dealing with different visualization types, will be described.

### Parallel Coordinates

For edge bundling in parallel coordinates visualization, Zhou, Yuan, Qu et al. [2008] proposed a method which replaces the edges in parallel coordinates visualizations with flexible springs and applies an attractive force between the springs/edges. This attractive force causes edges that are close to each other, based on Euclidean distance, or edges that are very parallel, based on angle, to bundle together as shown in Figure 2.3. Additionally, a curvature energy term prevents extreme bending of edges. The spring model is described by an energy function and the global optimization result of the energy function is achieved by utilizing linear programming. Figure 2.4 shows an example result of the described method.

### Graphs

Similar to the energy model described in subsubsection 2.1.2 a model for general graphs was described by Zhou, Yuan, Cui et al. [2008]. This method is needed because general graphs do not have fixed edge patterns like they occur in parallel coordinates. The algorithm by Zhou, Yuan, Cui et al. [2008] deals with this increased complexity in a four step process which is depicted in Figure 2.5 and described below in greater detail.

The first step is to triangulate the graph using the well know Delaunay triangulation, this will result in triangles with minimal internal angles and will avoid very sharp angles. Figure 2.5b shows the resulting Delaunay triangulation.

In the second step, the edges of the graph are sampled by utilizing the generated Delaunay triangulation, as shown in Figure 2.5c, which supports the generation of control points that represent the midpoints of the

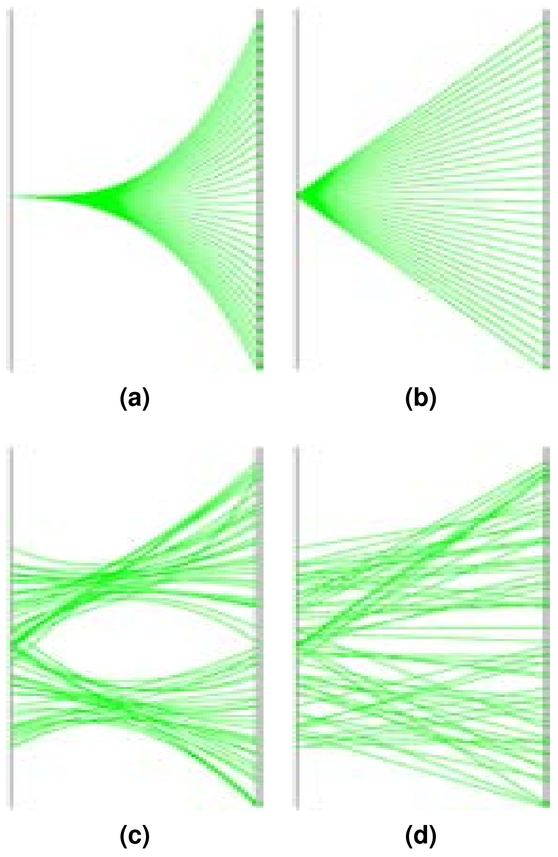


**Figure 2.3:** Before (a,c) and after (b,d) edge bundling using attractive force and flexible springs.
[Image extracted from Zhou, Yuan, Qu et al. [2008]].

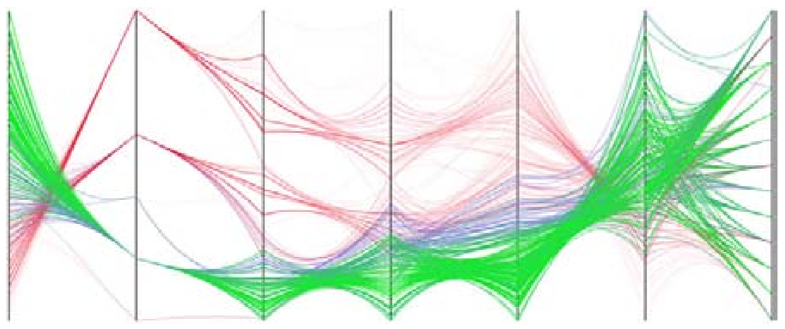


**Figure 2.4:** Edge bundling in parallel coordinates using the approach described by Zhou, Yuan, Qu et al. [2008], including color and opacity enhancements.
[Images extracted from Zhou, Yuan, Qu et al. [2008].]

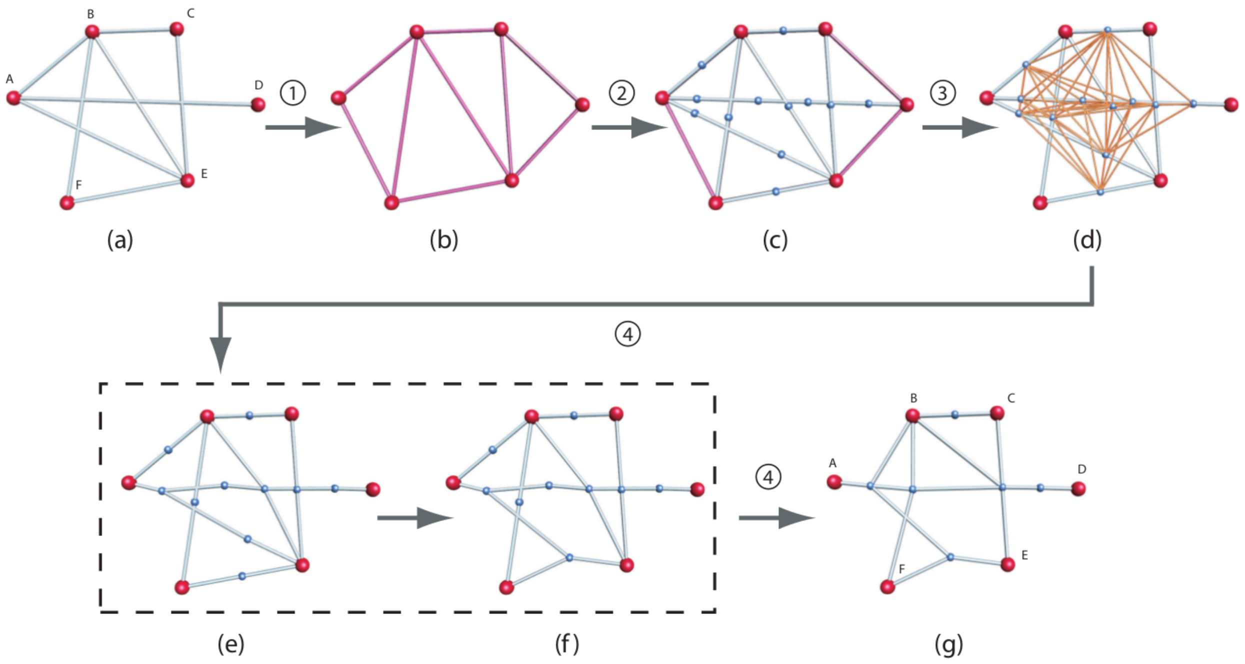
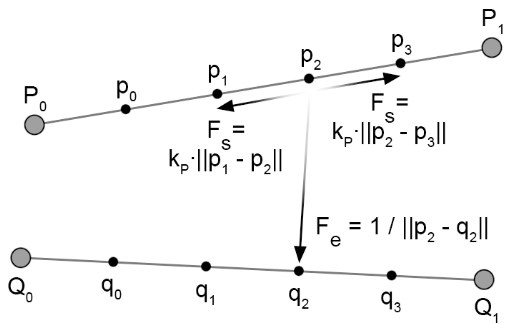


**Figure 2.5:** Algorithm flow of the method proposed by Zhou, Yuan, Cui et al. [2008].
[Images extracted from Zhou, Yuan, Cui et al. [2008].]



$P_0$
$p_0$
$p_1$
$p_2$
$p_3$
$P_1$
$F_s = k_P \cdot ||p_1 - p_2||$
$F_s = k_P \cdot ||p_2 - p_3||$
$F_e = 1 / ||p_2 - q_2||$
$Q_0$
$q_0$
$q_1$
$q_2$
$q_3$
$Q_1$

**Figure 2.6:** Spring forces (Fs) and electrostatic force (Fe) acting in the algorithm proposed by Holten and van Wijk [2009].
[Images extracted from Holten and van Wijk [2009].]

sampled segments. This sampling is non-uniform and preserves the topology of the graph.

Using the control points generated in the previous step, a control map is created. The edges of the control map are modeled as springs which attract edges that are close or parallel to each other. An example of a generated control map is shown in of Figure 2.5d.

In the last step, the hierarchical clustering of the nodes is performed to create the edge hierarchy. The nodes within the control map are then merged pairwise until the energy of the graph is optimized. After each pairwise merge the edges in the original graph are bundled according to the merge result.

Another force directed approach for general graphs was proposed by Holten and van Wijk [2009]. Their force directed approach is self organizing and aims to be easy to use and understand since it is strongly based on physics. As with probably all edge bundling techniques, the first step of this method, segments the edges to generate control points. Zero-length springs—springs which output no force when their length is zero—are used to connect the control points. Additionally, an electrostatic force that attracts the created segments to each other is added. Figure 2.6 shows a depiction of the physical models acting forces.

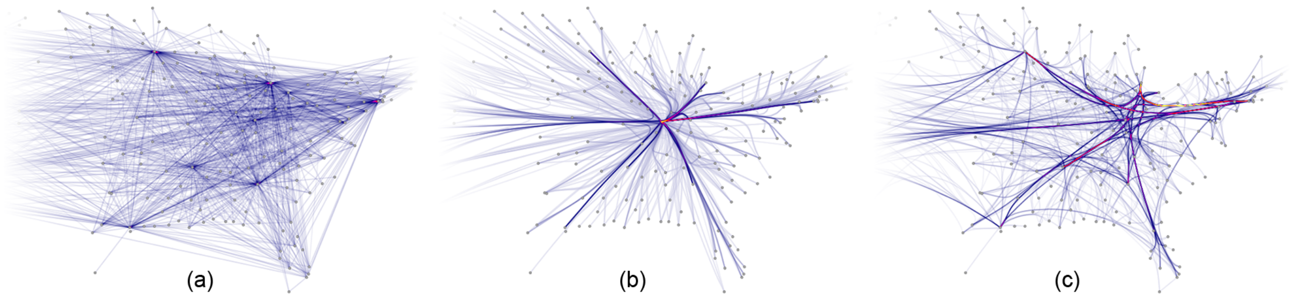


**Figure 2.7:** (a) unbundled graph, (b) simple force based edge bungling, (c) active edge compatibility measures.
[Images extracted from Holten and van Wijk [2009].]

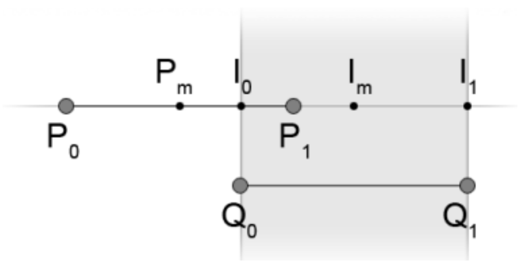


**Figure 2.8:** Force-directed edge bundling visibility compatibility.
[Images extracted from Holten and van Wijk [2009].]

This inverse-linear physical model is then iteratively evaluated and results in the optimal positioning of the generated control points. Figure 2.7b shows an example result of this basic evaluation.

To improve the bundling quality, Holten and van Wijk [2009] use various edge compatibility measures:

1. Angle compatibility
2. Scale compatibility
3. Position compatibility
4. Visibility compatibility

Angle compatibility avoid the bundling of edges that are almost perpendicular while scale compatibility reduces the bundling of edges with considerable differences in length. Position compatibility makes sure that edges which are far apart are not bundled and visibility compatibility avoids the special cases where edges should not be bundled too much despite fulfilling all other edge compatibility measures. An example for visibility compatibility is shown in Figure 2.8.

As shown in Figure 2.7c, the application of this edge compatibility measure improves the resulting visual bundling quality greatly.

## 2.2 Geometry-based

The approaches presented in this section subdivide space into geometrical structures and use these as guidance for edge bundling.

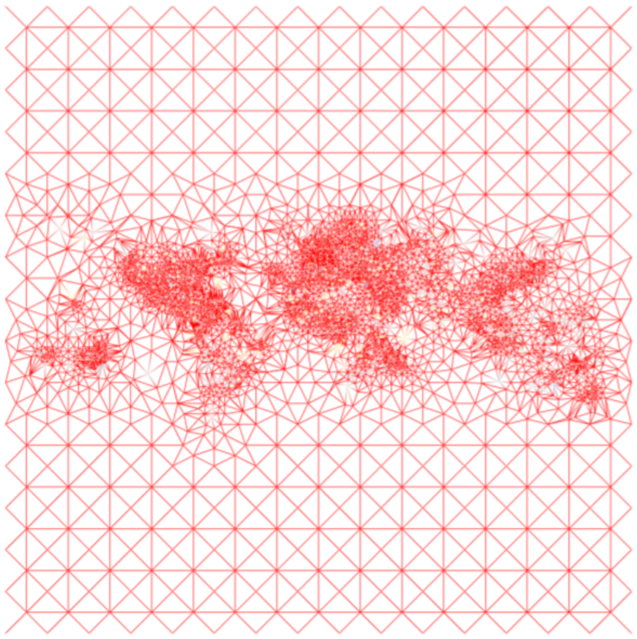


**Figure 2.9:** An example of the resulting quadtree and Voronoi diagram grid. Regions with higher density have a smaller grid size.
[Image extracted from Lambert et al. [2010].]

### 2.2.1 Grid-based

**Grid Construction** These variants are based on a multi-resolution grid that separates the nodes. While there are different approaches to create this grid, the end result is always a grid with one or zero graph nodes per grid cell. Even for regions with very sparse nodes, a maximum grid-size is recommended, otherwise edges connecting these nodes might be distorted disproportionally.

Lambert et al. [2010] compare different algorithms to construct these grids: Quadtree Separation, Voronoi diagrams and a combination of both. The conclusion of their study was that the best result is achieved with a combination of quadtrees and Voronoi diagrams. An example of such a grid can be seen in Figure 2.9.

**Edge Routing** In the next step of the algorithm, all original edges are replaced by paths along the grid. To achieve this, Dijkstra's algorithm is used in multiple iterations [Knuth, 1977].

This algorithm finds the shortest path between two nodes in any weighted network. For the first iteration, each grid-edge has the same weight. Before the next iteration, the weights are updated, and grid-edges which are used by many paths get a lower weight. This ultimately leads to a bundling effect, since grid-edges that were already used by many paths will be used by even more paths after each iteration. See Figure 2.10 for an example of the result of this process.

The resulting poly-lines are then typically replaced with Bezier curves [Knuth, 1986] using the grid-points as control-points. In comparison to the poly-lines, the Bezier curves actually resemble the shape of physically bundled connections. Another positive side-effect of these curves is that similar curves have slightly different curvatures, which causes the bundles to appear thicker if they are composed of more edges. This can be observed in Figure 2.11.

**De-cluttering** The advantage of using Dijkstra's algorithm is that it is possible to assign higher weights to grid-edges that should be avoided. This can be used to route edges away from areas with a high node-density which can de-clutter the resulting graph and prevent confusion.

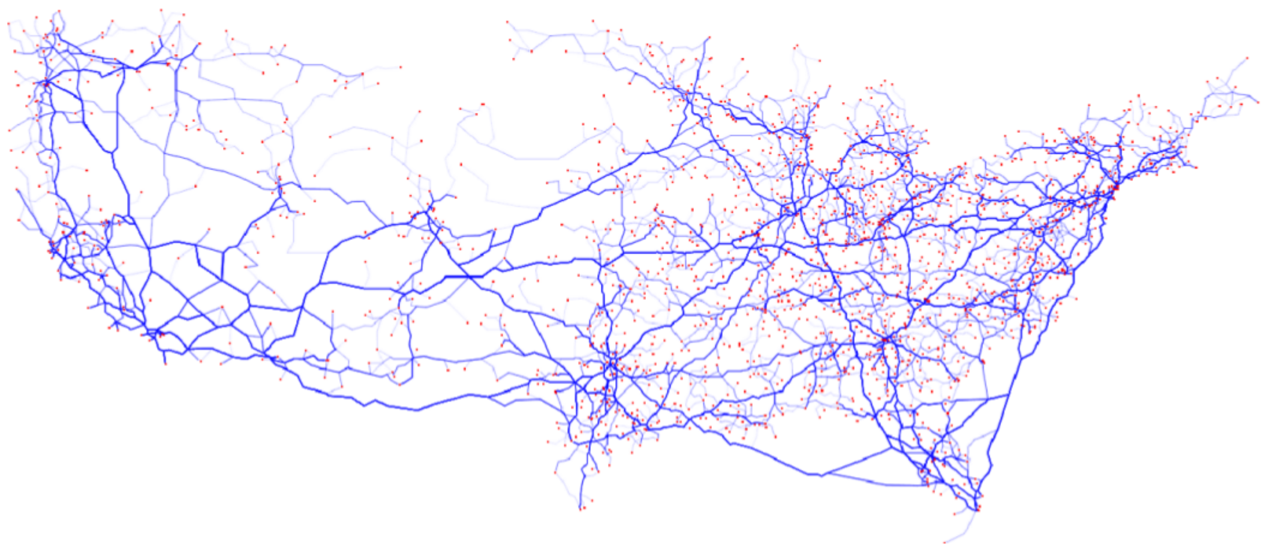


**Figure 2.10:** An example of edges routed along the generated grid.
[Screenshot from Tulip, taken by the authors.]

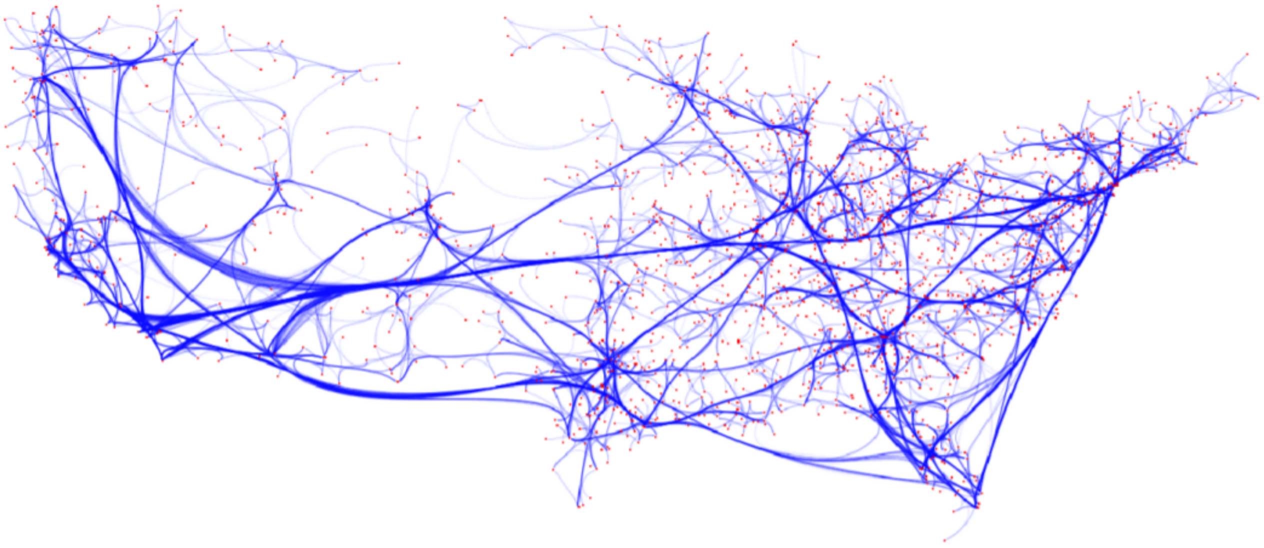


**Figure 2.11:** Bezier curves with grid points as control points.
[Screenshot from Tulip, taken by the authors.]

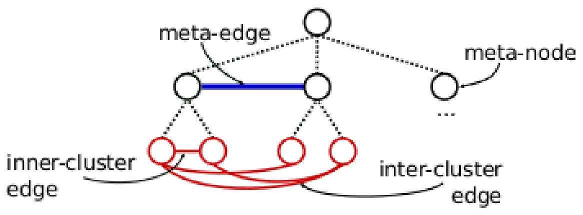


**Figure 2.12:** The hierarchy constructed by inserting the meta-nodes. Inter-cluster edges are replaced by meta-edges.
[Image extracted from Kienreich et al., 2012.]

### 2.2.2 Hierarchical Voronoi-based

This technique, as described in Kienreich et al. [2012], uses hierarchical information between nodes to generate edge-bundling information in different levels of detail. This means that the resulting graph can be zoomed into to display additional information for a specific region of interest.

Unlike the Grid-/Voronoi-based algorithm described in subsection 2.2.1 the original position of the nodes is changed during the bundling process as shown in [Kienreich et al., 2012]. Thus the algorithm is not feasible for networks with information in the node-position, i.e. geographical networks.

#### Pre-processing

This algorithm requires hierarchy information between the nodes. However, the nodes of most networks are not structured hierarchically, but this structure can be inferred by clustering the nodes according to their interconnectivity and introducing so-called meta-notes (or parent-nodes) for nodes in the same cluster. These meta-nodes can subsequently be clustered again to generate several hierarchy-levels. An example of a hierarchy constructed in this manner can be seen in Figure 2.12.

#### Algorithm

**Edge-Aggregation** Any inter-cluster edges — i.e. edges, that connect two nodes, which do not share the same meta-node — are removed. Instead, the corresponding meta-nodes are connected with an edge. The weight of this edge corresponds to the number of edges replaced between the two clusters.

**Node Layout and Grid Construction** Nodes and meta-nodes are positioned layer-wise starting with the highest-level meta-nodes in the hierarchy. The nodes of the top layer are positioned using the LinLog layout algorithm from Noack [2007] which considers the weight of the edge between two nodes, causing more interconnected nodes to be positioned closer to each other.

In the next step a Voronoi area subdivision, as described in Klein [1988], is done with the positioned nodes of that layer resulting in a polygon for each top-level node. The area of each polygon is then used for the next iteration of the layout algorithm, which is recursively processed, until the layer of the original nodes is reached, which are once again placed in the Voronoi-polygon of their lowest level meta-node. An example of the resulting recursive Voronoi-diagram can be seen in Figure 2.13.

The hierarchical Voronoi supports multiple levels of detail in the visualization, which allows zooming into specific regions of interest to reveal the next level of meta-nodes or nodes. See Figure 2.14 for an example of this feature.

**Edge Routing and Bundling** Similarly to the technique described in subsection 2.2.2, all edges are routed along the edges of the Voronoi grid, once again using Dijkstra's algorithm as described in [Knuth, 1977]

However, the weights of the edges are not changed and only one iteration of Dijkstra's algorithm is done, after which the newly routed edges are bundled using a force-directed edge bundling, described in section 2.1.

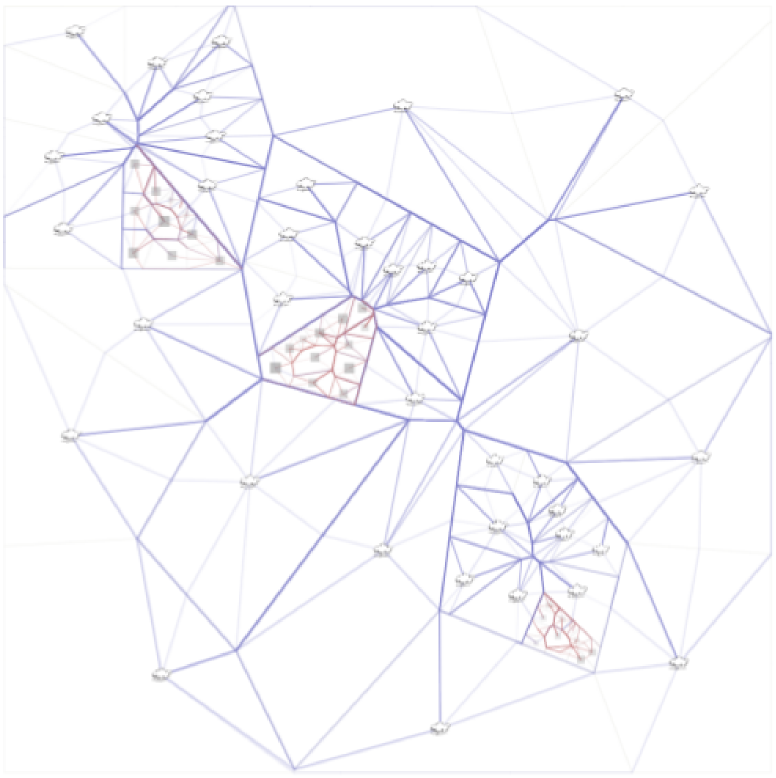


**Figure 2.13:** A recursively generated Voronoi-diagram. The nodes in the central region have more layers in their hierarchy, so there are additional Voronoi-layers.
[Image extracted from Kienreich et al., 2012.]

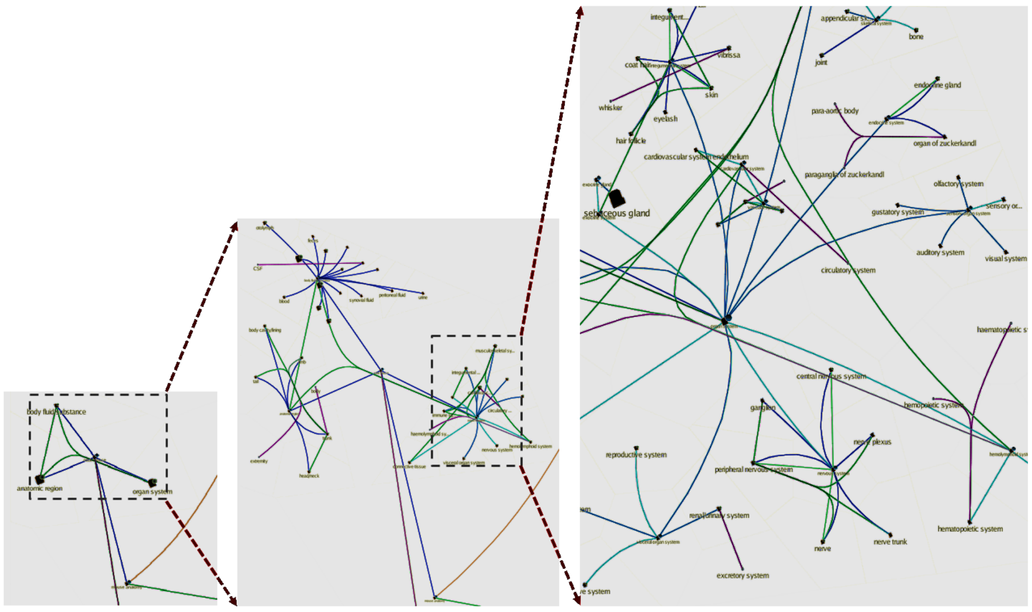
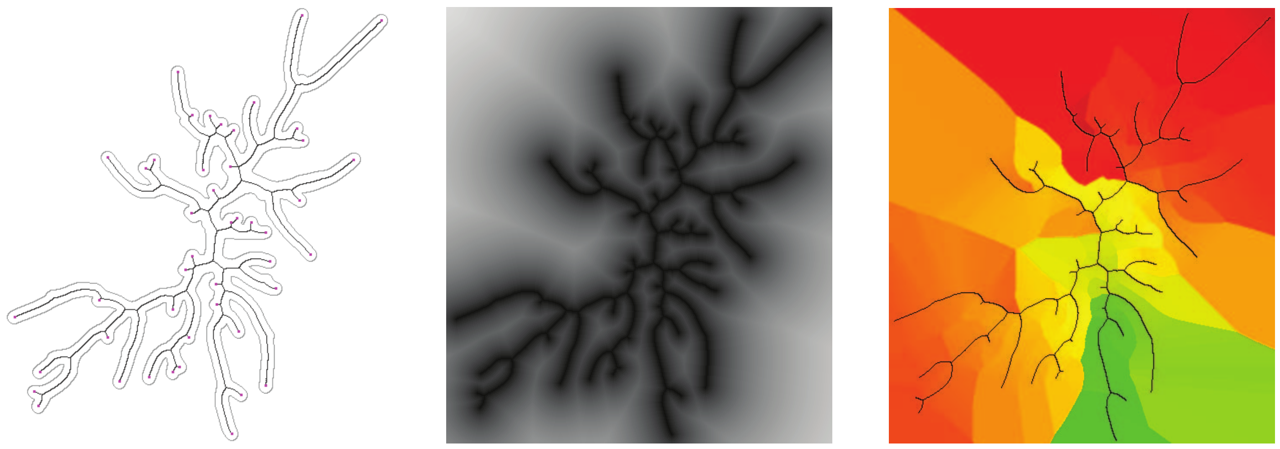


**Figure 2.14:** Result of the Hierarchical Voronoi-algorithm with three different levels of detail.
[Image provided by Vedran Sabol.]



**(a)** Skeleton within its 2D shape. **(b)** Distance transform. **(c)** Feature transform.

**Figure 2.15:** Illustrations of the skeleton-based edge bundling process.
[Images extracted from Ersoy et al. [2011].]

## 2.3 Image-based

Image-based edge bundling techniques combine computation and rendering of bundles into a single process. Due to this, existing techniques in computer graphics and computational geometry can be applied. In addition, the discretization that is obtained by working in image space can be useful to reduce the algorithmic complexity of the computations. Another benefit is that they can take advantage of the high parallelism of modern graphics processing units (GPUs) to achieve considerable performance improvements [van Liere and de Leeuw, 2003; McDonnell and Mueller, 2008; Telea and Ersoy, 2010; Ersoy et al., 2011].

Most image-based techniques such as Ersoy et al. [2011] and McDonnell and Mueller [2008] operate on sets of clustered edges obtained by applying a clustering algorithm (e.g. K-means) either as a separate preprocessing step or iteratively during the bundling process.

In the following sections, a choice of relevant image-based edge bundling techniques will be presented.

### 2.3.1 Skeleton-based

This technique which was proposed by Ersoy et al. [2011] uses a concept called a *Skeleton*. This can be defined for any 2D shape and is basically a line that is locally centered within the shape, i.e. each point on the line has the same distance to at least two points on the boundary of the shape. Figure 2.15a gives an intuitive overview of the structure of such a skeleton.

Skeleton-based edge bundling is applied to clusters of edges computed using a standard clustering method. It then draws the edges of each cluster to a temporary image and finds a bounding shape surrounding the drawing of the edges. The border of this shape is defined by a fixed minimum distance from each drawn pixel. Then, the skeleton of this shape is computed in image space by finding points within the shape that have equal distance to at least two points on the shape.

Subsequently, the edges are attracted towards their respective *feature points*, i.e. the point on the skeleton that is nearest to them. To achieve this efficiently, a distance transform and a feature transform are computed in image space. The distance transform defines the distance to the nearest skeleton point for each position in the image. This can be seen in Figure 2.15b. Similarly, the feature transform, illustrated in Figure 2.15c is used to find out which point on the skeleton is the nearest point.

These steps are repeated iteratively, until the desired result is reached. It can be configured whether to repeat the clustering step, since this part is the only non-image-based process, and can be rather slow. Figure 2.16 provides an overview of the whole skeleton-based edge bundling pipeline.

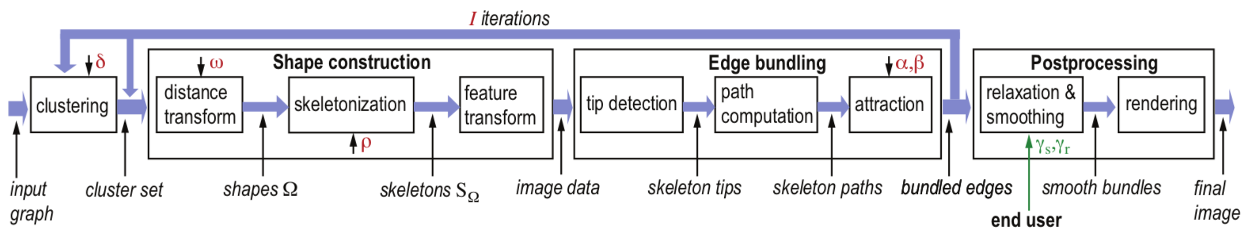
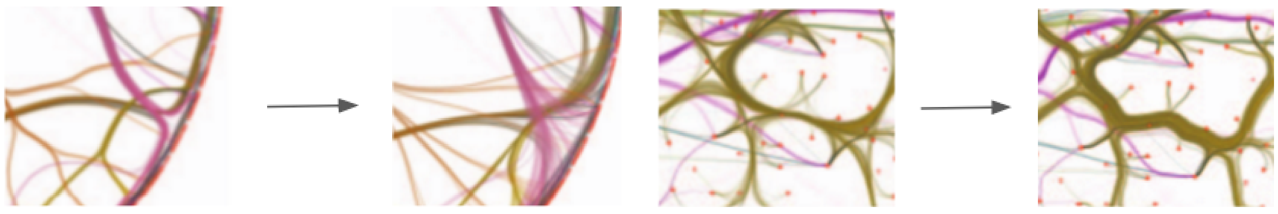


**Figure 2.16:** Illustration of the skeleton-based edge bundling process.
[Image extracted from Ersoy et al. [2011].]



**(a)** Smoothing. **(b)** Relaxation.

**Figure 2.17:** The effects of smoothing and relaxation on bundled edges.
[Images extracted from Ersoy et al. [2011].]

In addition, to achieve a better visual result, some post-processing techniques are usually applied. Ersoy et al. [2011] use *smoothing* and *relaxation* to improve the appearance of the bundled graph.

For smoothing, simple Laplacian line smoothing is applied on the edges. Conceptually, a Laplacian line filter iterates over a curve and moves each point towards the straight line between its neighbours. As can be seen in Figure 2.17a, this mitigates sharp turns in edges. This can be especially useful for intersections of multiple edge bundles, in order to improve the ability to follow their paths.

Relaxation is the application of linear interpolation between the result of the edge bundling and the original position of the edges. This visually highlights the amount of edges within a bundle by increasing its thickness if the edge count is higher. See Figure 2.17b for the visual effect of this transformation.

Finally, some image-based techniques can be applied to improve the rendering of the edge bundles. For example, Ersoy et al. [2011] use the skeleton structure to shade edges depending on their distance to the skeleton, which results in *cushion shading* for bundles. This can be seen in the final result shown in Figure 2.18. Another rendering technique that can be seen there is translucency, which makes it easier to follow the paths of occluded bundles. Depending on the input data, different techniques and parameters can be applied. Figure 2.19 shows a different result from Ersoy et al. [2011] without cushion shading applied.

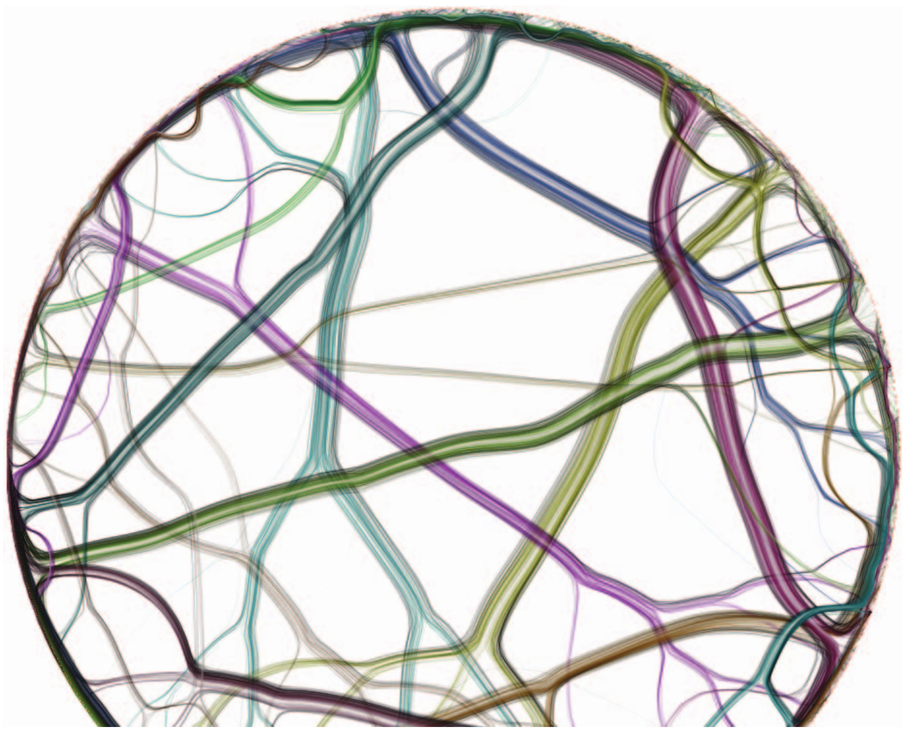


**Figure 2.18:** Skeleton-based edge bundling applied on a hierarchical dataset.
[Image extracted from Ersoy et al. [2011].]

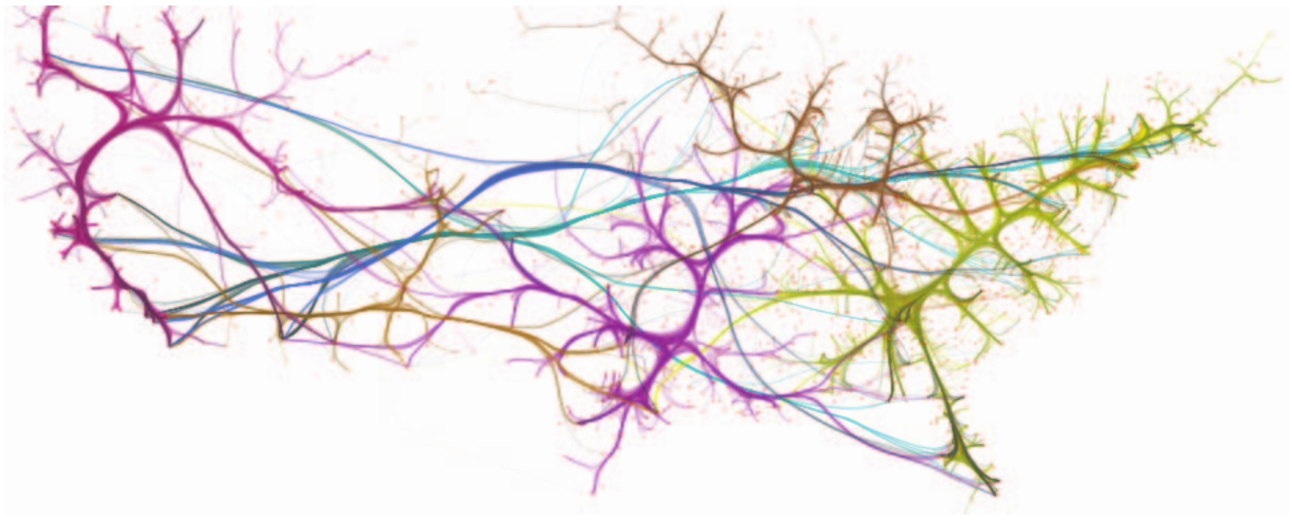


**Figure 2.19:** Skeleton-based edge bundling applied on the US migrations dataset.
[Image extracted from Ersoy et al. [2011].]

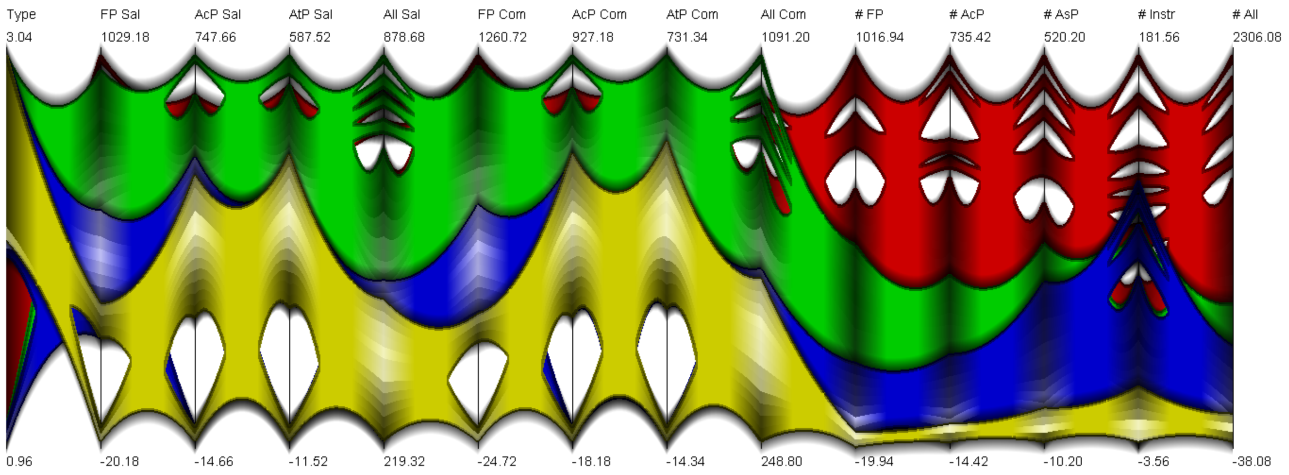


**Figure 2.20:** The result of applying illustrative parallel coordinates.
[Image extracted from McDonnell and Mueller [2008].]

### 2.3.2 Illustrative Parallel Coordinates

Image-based techniques can also be useful for visualizing parallel coordinates. McDonnell and Mueller [2008] propose a method that aims to highlight the difference between clustered sets of parallel coordinates data. They also visualize the density distribution of edges within each cluster that crosses a parallel coordinate axis. Figure 2.20 shows their resulting illustration.

To achieve this look, McDonnell and Mueller [2008] first convert the edges to splines. Then, the splines are curved towards their per-cluster midpoints between the parallel coordinates axes. This bundling aids in distinguishing different clusters. To further increase this effect, the actual edges are then removed. Instead, the area between the top and bottom edge of each cluster on each section are used to define a 2D surface. This surface is then filled with a single colour and the edges are highlighted even further by a drop-shadow effect.

In order to indicate the distribution of edges on a per-cluster basis, small quads are overlaid on each axis which are shaded depending on the local edge density. The quads are also distorted to indicate the edge directions on the axis, since the continuous shading of clusters hides that information.

# Chapter 3

# Software

In this chapter, a choice of software packages that implement edge bundling is evaluated. The aim is to provide a comparable overview of the currently available options. Therefore, very similar coloring and drawing styles for all the results is used. In addition, whenever possible, the visualization software is applied on the same reference dataset.

This reference dataset is *US airports (opsahl-usairport)* from the KONECT network dataset collection [Kunegis, 2017], licensed under a Creative Commons Attribution-ShareAlike 2.0 Germany License. This network was originally published by Opsahl [2011].

The association of the airports in the reference dataset with geographical data was done using data in the public domain [1]. The geographical data is used to produce the same node layout for the different software packages.

The dataset is relatively large, with 21607 undirected edges and 1338 nodes. As will be shown in section 3.5 and section 3.6, a network of this size can already be problematic for some visualization software.

## 3.1 Graphviz

**Website** `http://www.graphviz.org/`

**Technology** C

**Platforms** Windows, Mac, Linux

Graphviz is an open-source graph visualization tool [Graphviz, 2017; Gansner and North, 2000]. It is command-line based and composed of multiple executables that read files which specify the structure of the graph in the DOT graph description language. These programs can be chained together using pipes in UNIX fashion and can output the final graph visualization in multiple formats such as PNG images. Some of these programs just apply transformations upon the DOT-based graph description. These programs are called *filters*.

Edge bundling in Graphviz is provided by a filter called *mingle*. As the name implies, this filter implements the MINGLE algorithm proposed by Gansner et al. [2011], which is described in more detail in subsubsection 2.1.1. Unfortunately, this filter requires the external ANN (Approximate Nearest Neighbour) library [Mount and Arya, 2010]. Therefore, the Graphviz packages in the author's Linux distribution did not support MINGLE and manual compilation of Graphviz was required [Vinet and Griffin, 2017].

The reference dataset was converted to the DOT format using a simple Python script and used as input for Graphviz. In comparison to importing the data into the applications with a graphical user interface, this was relatively convenient and fast. Figure 3.1 shows the result of drawing the reference dataset using Graphviz without edge bundling applied.

[1] `http://ourairports.com/data/`

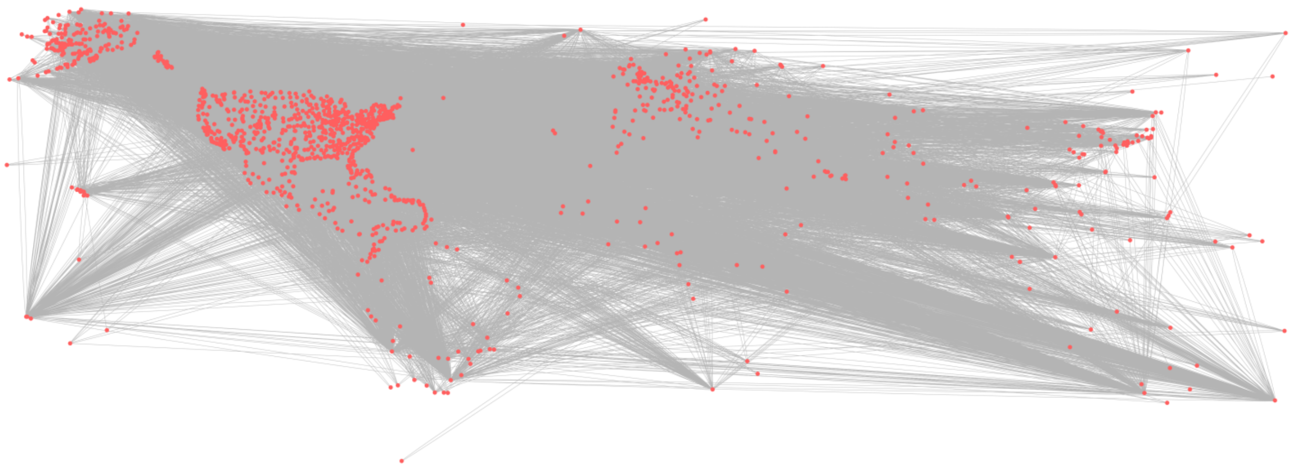


**Figure 3.1:** Graphviz result without edge bundling.
[Produced with Graphviz by the authors of this paper.]

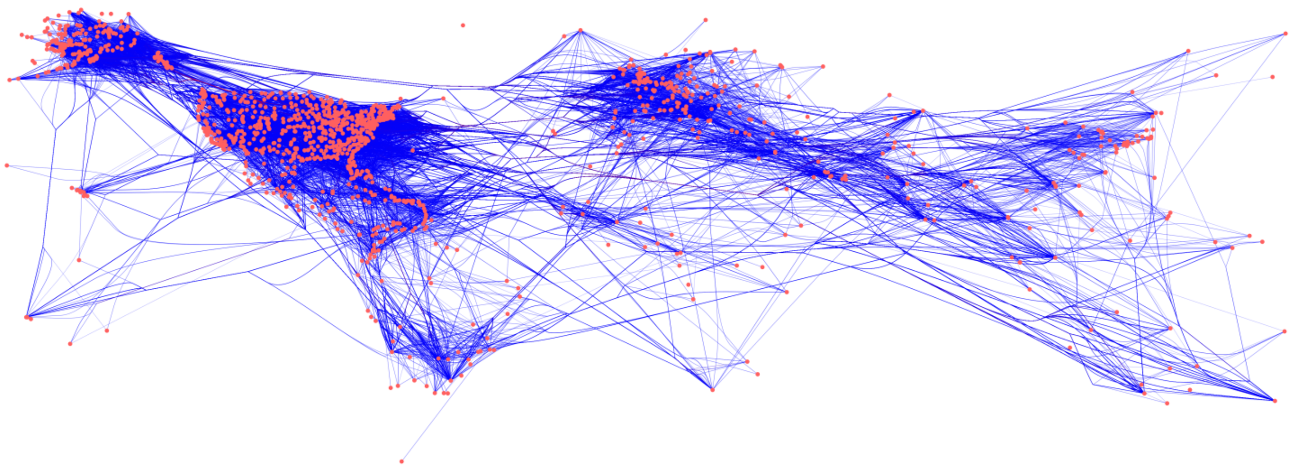


**Figure 3.2:** Graphviz edge bundling result from the mingle filter.
[Produced with Graphviz by the authors of this paper.]

After manually compiling Graphviz, applying the mingle filter was quite easy. The performance was also very good, with the edge bundling taking less than 10 seconds on an average consumer notebook. As can be seen in the final result in Figure 3.2, unfortunately, edge bundling destroyed the edge colorings.

## 3.2 Graph-tool

**Website** `https://graph-tool.skewed.de/`

**Technology** Python, C++

**Platforms** Mac, Linux, (Windows only with Docker or WSL)

Graph-tool is a Python library that is designed to manipulate and analyze graphs. Most parts of the software are implemented in C++, based on the Boost Graph Library [Peixoto, 2014; Jeremy Siek and Lumsdaine, 2017]. This combination is intended to provide an easy user interface to an efficient implementation. Unlike Graphviz, this library cannot be used from the command-line. Instead, the user is required to write a Python script that uses the library.

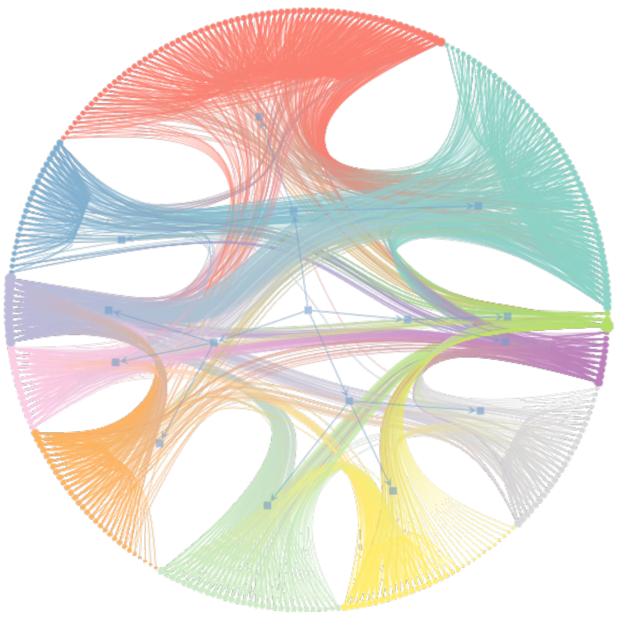


**Figure 3.3:** Hierarchical edge bundling example created using graph-tool.
[Image extracted from Peixoto [2014]. Licensed under CC-BY 4.0.]

In addition to analysis features, the library also supplies a variety of visualization tools. Amongst those are multiple vertex layout algorithms and efficient drawing routines with different back-ends. However, only hierarchical edge bundling as introduced by Holten [2006] is supported. An example of this can be seen in Figure 3.3.

Since edge bundling in graph-tool is limited to this technique, it could not be applied to the reference dataset for comparison because it is not hierarchically structured.

## 3.3 Tulip

**Website** `http://tulip.labri.fr`

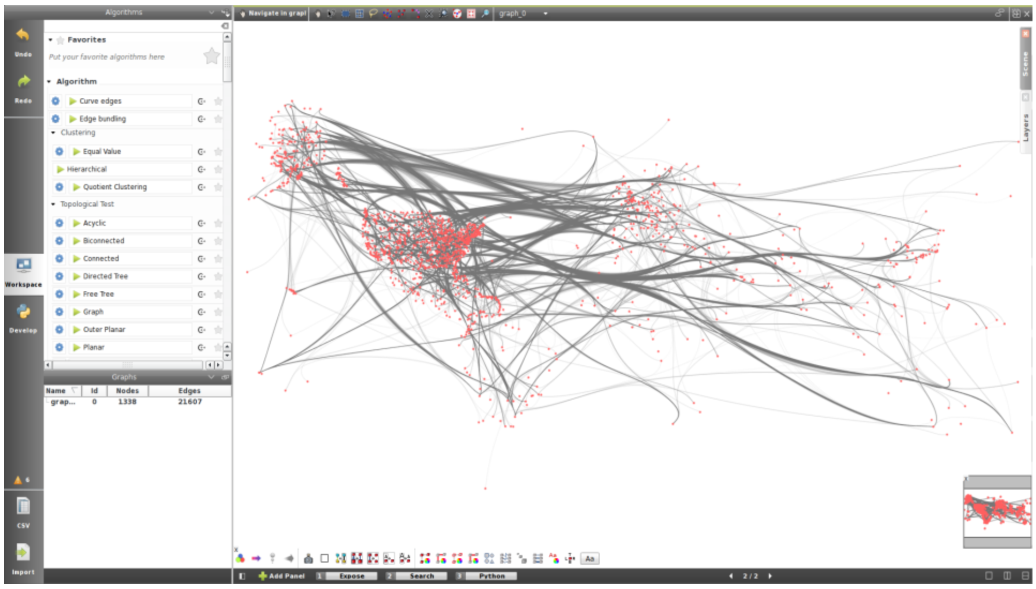
**Technology** C++, Qt

**Platforms** Windows, Mac, Linux

Tulip is another open-source graph visualization and analysis tool [Auber and Mary, 2017]. In contrast to Graphviz and graph-tool, Tulip provides a graphical interface and numerous plugins that implement algorithms which compute graph metrics and statistics as well as different visualizations and layouts. This allows for interactive exploration and manipulation of the dataset. For example, the user can select and move nodes, highlight edges connected to a node or even modify the appearance of single edges and nodes.

Figure 3.4 provides an overview of the graphical user interface of Tulip. On the left-hand side, the plugins that can be run are listed. Around the graph in the center, one can also see a selection of tools that can be used to interactively modify some properties of the displayed graph. There are also some options to import or export data on the far left side. Some more advanced interface panels like the geographical view can be added on demand via the *Add Panel* button on the bottom. One can also view and edit some properties of the current view by expanding the panels on the right-hand side. For example, one could expand the *Layers* panel and hide edges in the current view.

Edge bundling in Tulip is provided by a plugin that implements grid-based edge routing [Lambert et al., 2010]. As described before in subsection 2.2.1, this does not actually bundle edges; rather it aligns them on a



| Name | Id | Nodes | Edges |
|---|---|---|---|
| grap... | 0 | 1338 | 21607 |

**Figure 3.4:** Graphical user interface of Tulip.
[Screenshot taken by the authors of this paper.]

grid.

Since Tulip supports importing CSV files, it was relatively straightforward to convert the reference dataset so it could be loaded. However, on import it was still necessary to specify how properties should be associated to nodes via the graphical interface. In addition, some manual work was required to use the Geodata for node positioning. Figure 3.5 shows the result of visualizing the reference dataset in Tulip without edge bundling. Applying the edge bundling plugin results in Figure 3.6. In this visualization, it is impossible to identify the number of edges following a path. It is quite simple to use the tools at the bottom of Figure 3.4 to apply simple alpha blending to achieve darker shades for paths with higher edge count, as can be seen in Figure 3.7. In addition, Tulip supports converting edges to curves with control points along the grid cells resulting from the edge routing algorithm, similar as the approach discussed in subsection 2.2.1. This facilitates a better sense of how many edges are in a bundle, as can be seen in Figure 3.8. Combining this with alpha blending results in Figure 3.9. This, in the author's opinion, is the visually best result with Tulip.

Unlike with the JavaScript-based tools described in section 3.5 and section 3.6, performance was not an issue for visualizing the reference dataset. Applying edge routing took less than 30 seconds on an average consumer notebook computer.

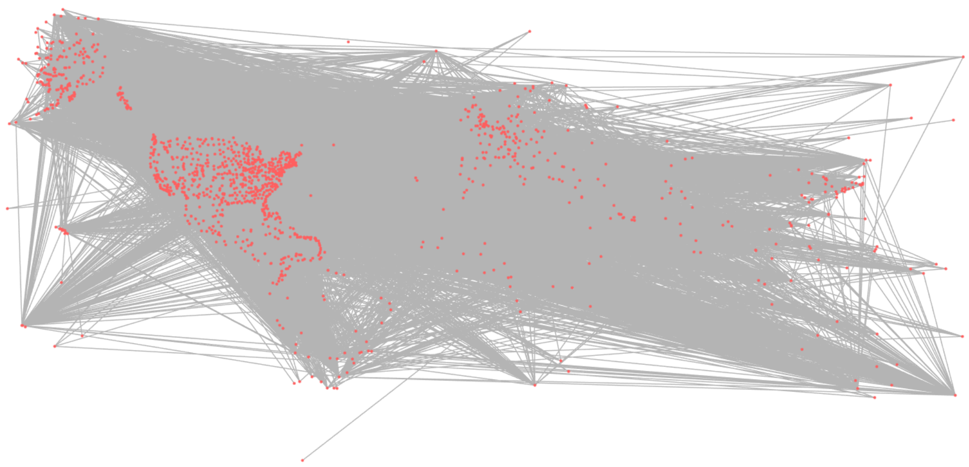


**Figure 3.5:** Tulip result without edge bundling.
[Screenshot taken by the authors of this paper.]

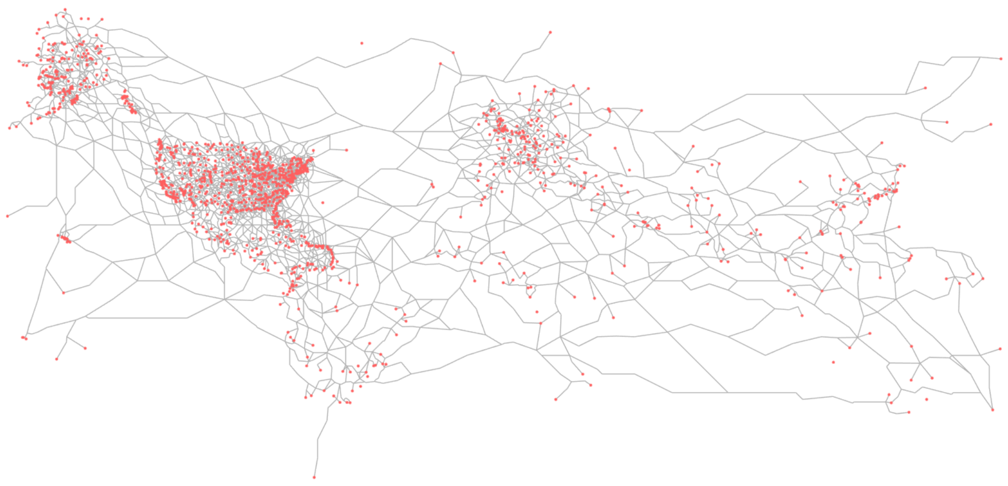


**Figure 3.6:** Tulip edge bundling, initial result.
[Screenshot taken by the authors of this paper.]

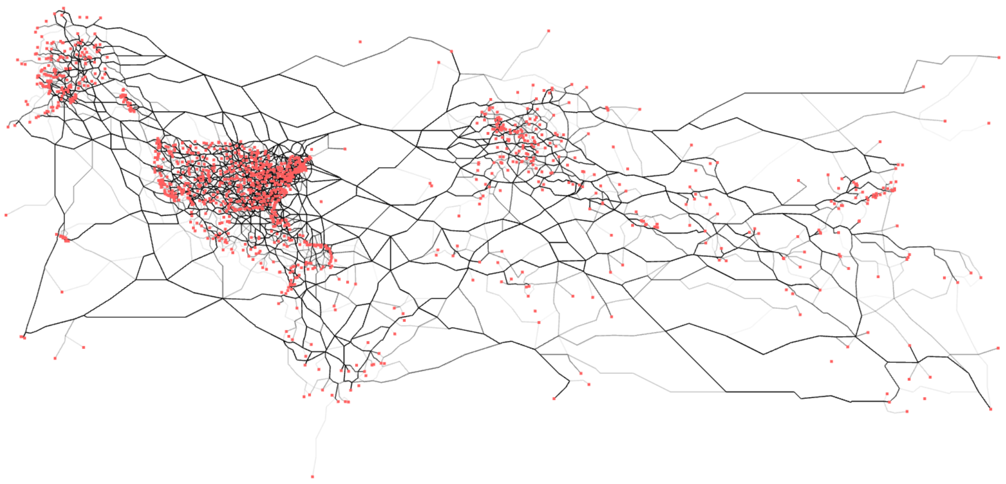


**Figure 3.7:** Tulip edge bundling with alpha blending.
[Screenshot taken by the authors of this paper.]

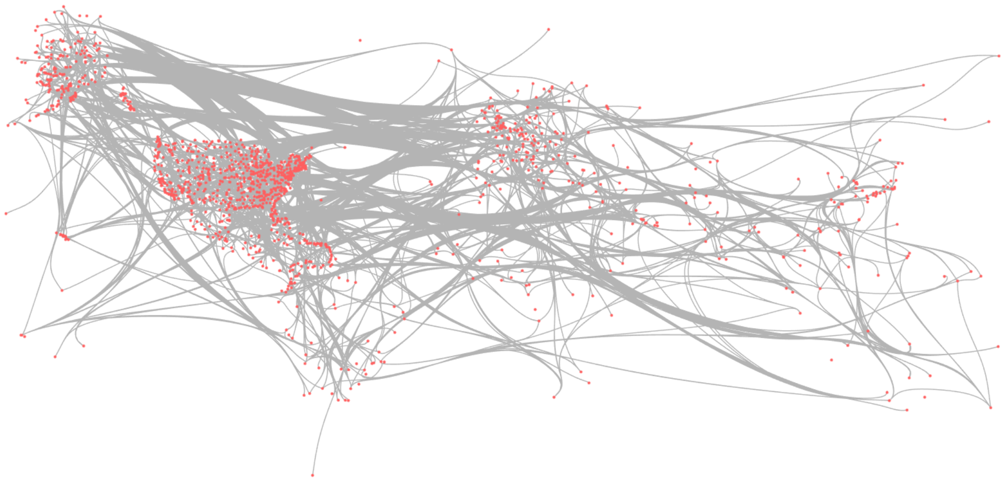


**Figure 3.8:** Tulip edge bundling with bezier curves.
[Screenshot taken by the authors of this paper.]

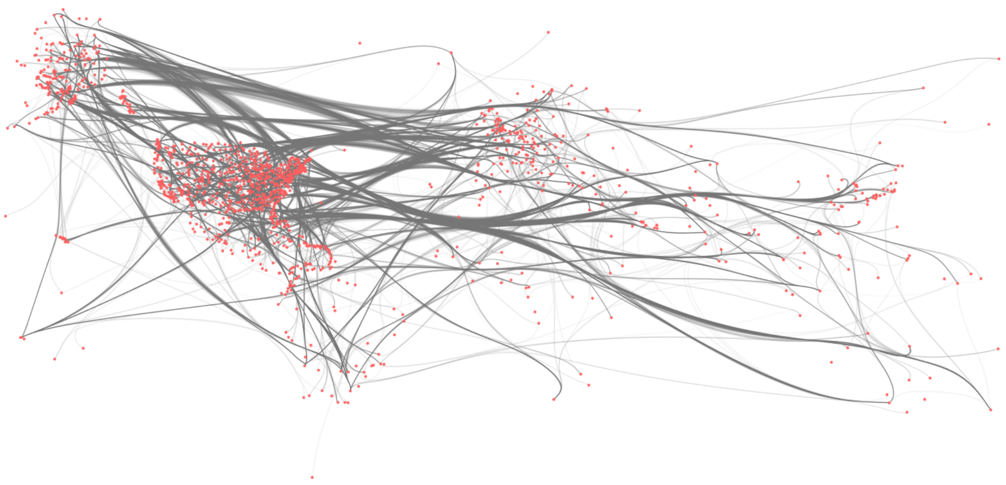


**Figure 3.9:** Tulip edge bundling with bezier curves and alpha blending.
[Screenshot taken by the authors of this paper.]

## 3.4 Gephi

**Website** `https://gephi.org/`

**Technology** Java

**Platforms** Windows, Mac, Linux

Gephi is an open-source graph analysis and visualization program similar to Tulip. It is written in Java and provides numerous plugins to extend the base functionality. The focus appears to be on interactive exploration and visualization of data via the graphical interface. For example, one can easily color and resize nodes or edges by associating imported data to them. Dynamic filters and queries provide the ability to further drill down. The visualized graph can be manipulated interactively with the mouse by selecting and dragging nodes. There are many more mouse-based tools to interact with the graph.

The current version (0.9.1) of Gephi does not support edge bundling. However, force-directed edge bundling as introduced by Holten and van Wijk [2009], was implemented as part of a Google Summer of Code project [Heymann, 2012]. Unfortunately, this work was not integrated into the main project, and has not been maintained since 2012. For evaluation purposes, we tried to run the old version, but in all our attempts, the software crashed on startup. For a screenshot of the edge bundling, refer to Heymann [2012].

## 3.5 D3 - Force Edge Bundling

**Website** `https://d3js.org/` `https://github.com/upphiminn/d3.ForceBundle`

**Technology** JavaScript

**Platforms** All modern browsers

The force edge bundling algorithm applied in this edge bundling implementation converts a given graph with edges and nodes into a graph with a significantly higher number of edges, because it converts each given edge

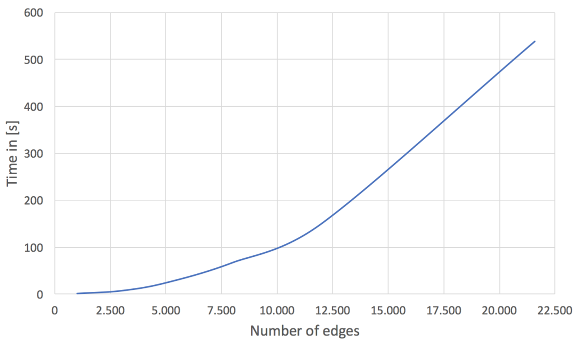


**Figure 3.10:** Performance of d3-forceEdgeBundling in relation to the number of edges.
[Diagram created by the authors of this paper.]

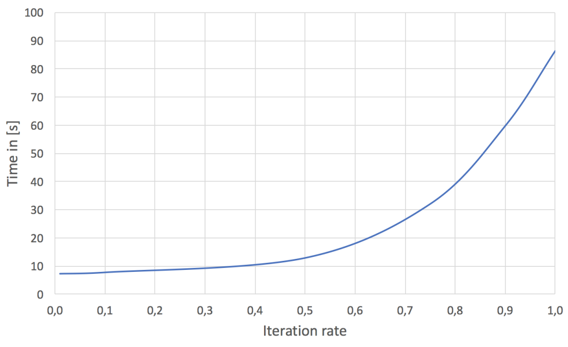


**Figure 3.11:** Performance of d3-forceEdgeBundling in relation to the iteration rate.
[Diagram created by the authors of this paper.]

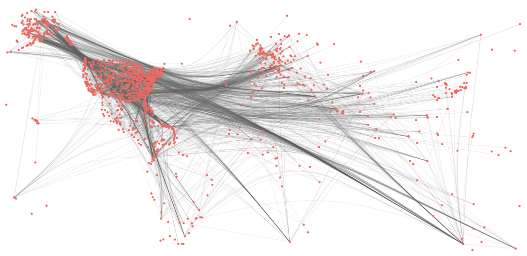


**Figure 3.12:** 3000 edges with iteration rate of 0.1 takes 7.7 seconds.
[Screenshot taken by the authors of this paper.]

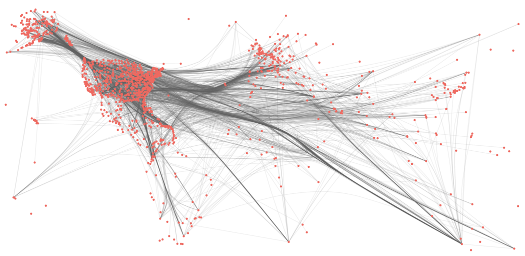


**Figure 3.13:** 3000 edges with iteration rate of 0.5 takes 12.85 seconds.
[Screenshot taken by the authors of this paper.]

into a polyline with many different line segments. Drawing the calculated graph is pretending the feeling of curved edges. The advantage of this approach is, that you could calculate all edges in advance and serve them with a web service. But this is also the only possibility to work with this implementation interactively, because the algorithm is really slow and has a polynomial time complexity as shown in figure 3.10. Increasing the quality by a rising iteration rate makes the algorithm soon useless in the sense of fast results (see figure 3.11). In figures 3.12, 3.13 and 3.14 you can see the quality of the edge bundling in relation to the chosen iteration rate. It also shows the time, which was needed to calculate the graph.

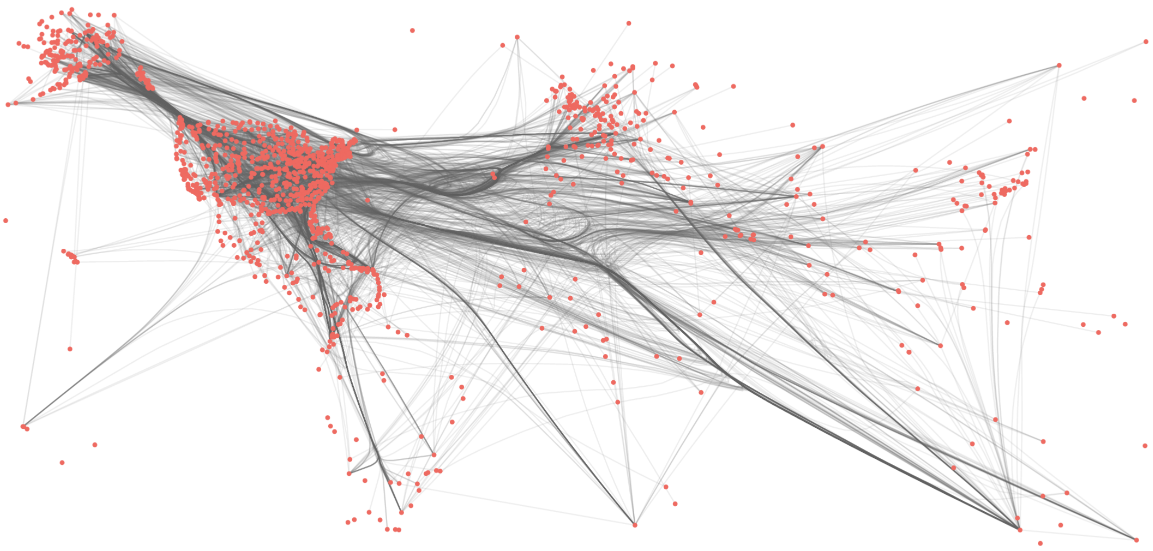


**Figure 3.14:** 3000 edges with iteration rate of 1.0 takes 86 seconds.
[Screenshot taken by the authors of this paper.]

## 3.6 MingleJS

**Website** `https://github.com/philogb/mingle`

**Technology** JavaScript

**Platforms** All browsers that support WebGL

MingleJS is a JavaScript implementation of Multilevel Agglomerative Edge Bundling for Visualizing Large Graphs by Gansner et al. [2011], already described in subsection 2.1.1. As stated there, the algorithm has a time complexity of $k|E|\log(|E|)$. Where $k$ is the number of nearest neighbors and $|E|$ is the number of edges. As this implementation is very easy to run with customized parameters, the behavior of performance and quality in relation to the number of nearest neighbors, number of edges and the maximum allowed angle will be investigated.

As can be seen in Figure 3.15, the relation of needed computation time to a different number of nearest neighbors is linear, as expected. Surprisingly, the computation time in relation to the number of edges in this implementation is also linear (see Figure 3.16). From the time complexity given in the paper, a complexity of $|E|\log(|E|)$ was expected.

By looking at the numbers stated in figures 3.15 and 3.16 in more detail, it has to be said, that you are not able to use this algorithm interactively for a large number of edges with more than 20 nearest neighbors, which is needed for sufficient quality. The time measured depending on to the number of edges is performed with only 5 nearest neighbors. For our work, the the result of the edge bundling algorithm starts to make sense by using at least 20 nearest neighbors, which would lead to a time of approximately eight seconds for 5000 edges. How the number of nearest neighbors affects the quality of this edge bundling algorithm is shown in figures 3.17, 3.18, 3.19, 3.20 and 3.21.

In this last part, different angle settings are compared. As described in subsection 2.1.1, you can choose different maximum turning angles, which avoids edges that are far away, leading to splines with large curvature. In Figure 3.22, different results by ascending maximum turning angles are shown.

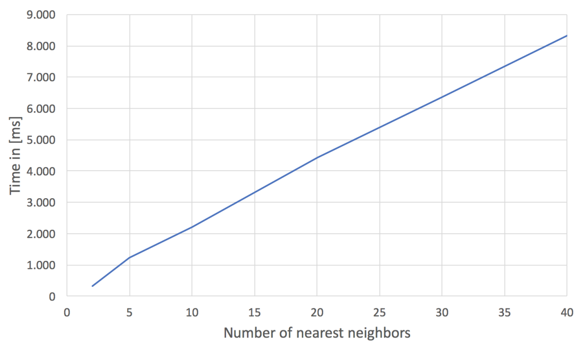


**Figure 3.15:** Performance in relation to number of nearest neighbors.
[Diagram created by the authors of this paper.]

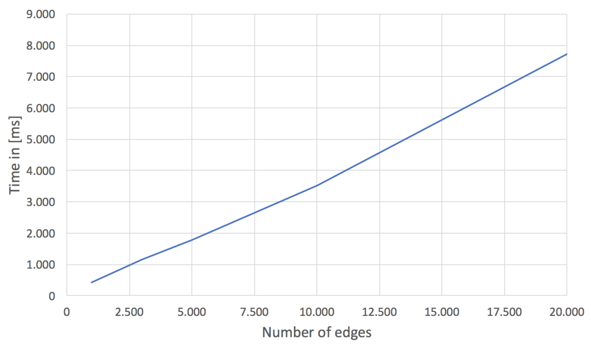


**Figure 3.16:** Performance in relation to number of edges.
[Diagram created by the authors of this paper.]

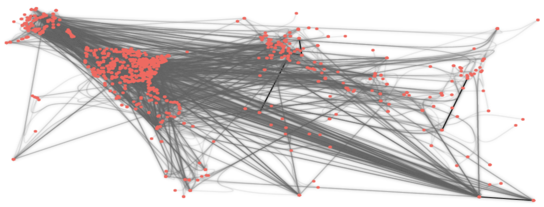


**Figure 3.17:** Mingle using 5 nearest neighbors.
[Screenshot taken by the authors of this paper.]

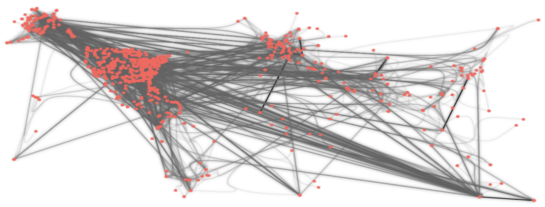


**Figure 3.18:** Mingle using 10 nearest neighbors.
[Screenshot taken by the authors of this paper.]

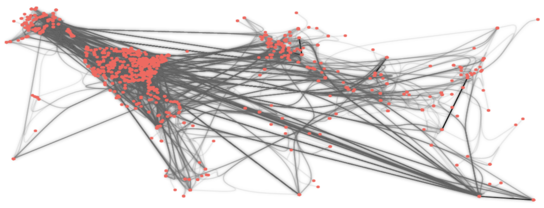


**Figure 3.19:** Mingle using 20 nearest neighbors.
[Screenshot taken by the authors of this paper.]

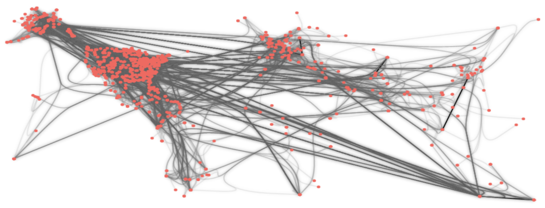


**Figure 3.20:** Mingle using 30 nearest neighbors.
[Screenshot taken by the authors of this paper.]

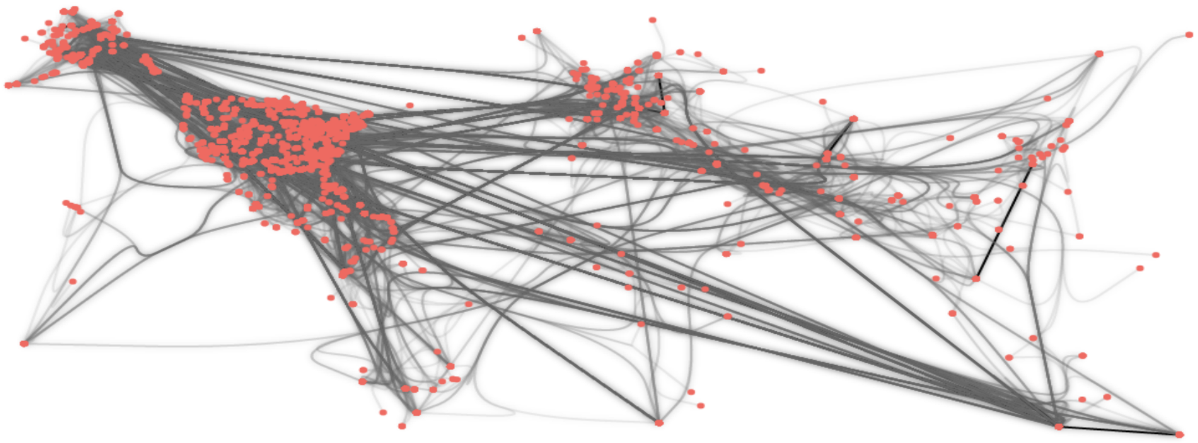


**Figure 3.21:** Mingle using 40 nearest neighbors.
[Screenshot taken by the authors of this paper.]

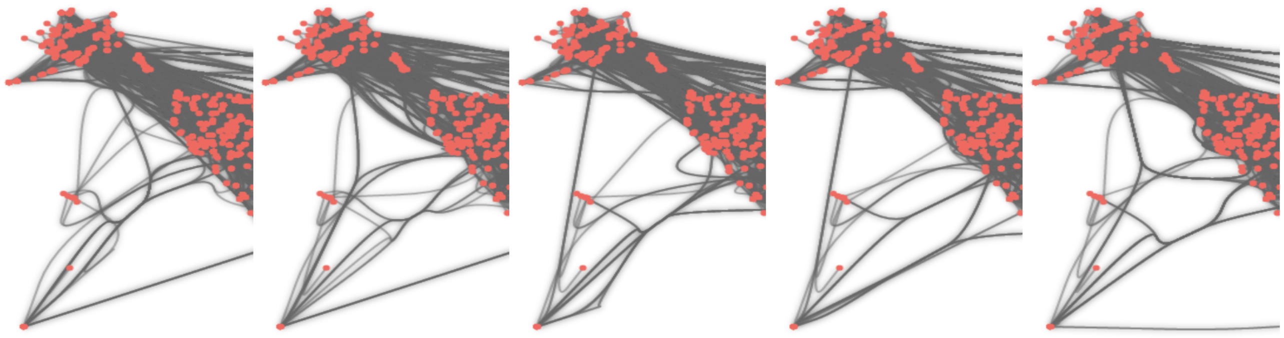


**Figure 3.22:** Shape of edges with increasing limit of turning angles.
[Screenshot taken by the authors of this paper.]

# Chapter 4

# Concluding Remarks

**Algorithms** There are several different techniques, each with their advantages and different fields of applications. If GPUs are available, parallelizable algorithms as described in section section 2.3 can produce high quality results. Some algorithms, like the Hierarchical Voronoi, described in subsection 2.2.2 fulfill specific purposes, since it is the only described algorithm that supports multiresolution results with different levels of detail. However, it is also the only algorithm described here, that changes the position of the nodes in the graph, which can be a problem for specific graphs, if, for instance geographical information is encoded in the position of the nodes.

**Software** With regards to software the recommendation depends on the use case. For standalone software the author's recommendation is Tulip, described in section 3.3, since it is very easy to use, has a lot of additional functionality, and produces great results. For graphs included in websites, the recommendation depends on the size of the network and whether or not the graph should be interactive. Large interactive graphs can be computed much faster with MingleJS since the runtime scales linearly, while the D3 framework has polynomial computation time increase. However, since the required computation can be performed ahead of time for D3, it is possible to simply store the computation result on the server. This of course requires the data to be static, preventing interaction by the user. In that case one could also use any other tool like Tulip.

# Bibliography

Auber, David and Patrick Mary [2017]. *Tulip*. 15th May 2017. `http://tulip.labri.fr/TulipDrupal/` (cited on page 19).

Ersoy, O., C. Hurter, F. Paulovich, G. Cantareiro and A. Telea [2011]. “Skeleton-Based Edge Bundling for Graph Visualization”. *IEEE Transactions on Visualization and Computer Graphics* 17.12 [Dec 2011], pages 2364–2373. ISSN 1077-2626. doi:10.1109/TVCG.2011.233 (cited on pages 13–15).

Gansner, E. R., Y. Hu, S. North and C. Scheidegger [2011]. “Multilevel agglomerative edge bundling for visualizing large graphs”. In: *2011 IEEE Pacific Visualization Symposium*. Mar 2011, pages 187–194. doi:10.1109/PACIFICVIS.2011.5742389 (cited on pages 3, 5, 17, 25).

Gansner, Emden R. and Yehuda Koren [2007]. “Improved Circular Layouts”. In: *Graph Drawing: 14th International Symposium, GD 2006, Karlsruhe, Germany, September 18-20, 2006. Revised Papers*. Edited by Michael Kaufmann and Dorothea Wagner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pages 386–398. ISBN 3540709045. doi:10.1007/978-3-540-70904-6_37 (cited on page 3).

Gansner, Emden R. and Stephen C. North [2000]. “An open graph visualization system and its applications to software engineering”. *SOFTWARE - PRACTICE AND EXPERIENCE* 30.11 [Sep 2000], pages 1203–1233. ISSN 0038-0644. doi:10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.3.CO;2-E (cited on page 17).

Graphviz [2017]. *Graphviz*. 15th May 2017. `http://graphviz.org/` (cited on page 17).

Heymann, Sébastien [2012]. *GSoC: Force Directed Edge Bundling*. 2012. `https://gephi.wordpress.com/2012/09/21/gsoc-force-directed-edge-bundling/` (cited on page 23).

Holten, D. [2006]. “Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data”. *IEEE Transactions on Visualization and Computer Graphics* 12.5 [Sep 2006], pages 741–748. ISSN 1077-2626. doi:10.1109/TVCG.2006.147 (cited on page 19).

Holten, Danny and Jarke J. van Wijk [2009]. “Force-directed Edge Bundling for Graph Visualization”. In: *Proceedings of the 11th Eurographics / IEEE - VGTC Conference on Visualization*. EuroVis’09. Berlin, Germany: The Eurographics Association & John Wiley & Sons, Ltd., 2009, pages 983–998. doi:10.1111/j.1467-8659.2009.01450.x (cited on pages 2, 7–8, 23).

Jeremy Siek, Lie-Quan Lee and Andrew Lumsdaine [2017]. *BGL: The Boost Graph Library*. 15th May 2017. `http://www.boost.org/doc/libs/1_64_0/libs/graph/doc/index.html` (cited on page 18).

Kienreich, Wolfgang, Ralph Wozelka, Vedran Sabol and Christin Seifert [2012]. “Graph Visualization Using Hierarchical Edge Routing and Bundling”. In: *EuroVA 2012: International Workshop on Visual Analytics*. Edited by Kresimir Matkovic and Giuseppe Santucci. The Eurographics Association, 2012. ISBN 3905673894. doi:10.2312/PE/EuroVAST/EuroVA12/097-101 (cited on pages 11–12).

Klein, Rolf [1988]. “Abstract Voronoi Diagrams and Their Applications”. In: *Proceedings of the International Workshop on Computational Geometry on Computational Geometry and Its Applications*. CG ’88. London, UK, UK: Springer-Verlag, 1988, pages 148–157. ISBN 3540503358. http://dl.acm.org/citation.cfm?id=647779.734920 (cited on page 11).

Knuth, Donald E. [1977]. “A generalization of Dijkstras algorithm”. *Information Processing Letters* 6.1 [1977], pages 1–5. ISSN 0020-0190. doi:10.1016/0020-0190(77)90002-3. http://www.sciencedirect.com/science/article/pii/0020019077900023 (cited on pages 9, 11).

Knuth, Donald Ervin [1986]. *METAFONT: the program*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1986. ISBN 0201134381 (cited on page 9).

Kunegis, Jérôme [2017]. *KONECT – The Koblenz Network Collection*. 15th May 2017. http://konect.uni-koblenz.de/networks/ (cited on page 17).

Lambert, A., R. Bourqui and D. Auber [2010]. “Winding Roads: Routing Edges into Bundles”. In: *Proceedings of the 12th Eurographics / IEEE - VGTC Conference on Visualization*. EuroVis. Bordeaux, France: The Eurographs Association & John Wiley & Sons, Ltd., 2010, pages 853–862. doi:10.1111/j.1467-8659.2009.01700.x (cited on pages 9, 19).

McDonnell, K. T. and K. Mueller [2008]. “Illustrative Parallel Coordinates”. *Computer Graphics Forum* 27.3 [2008], pages 1031–1038. ISSN 1467-8659. doi:10.1111/j.1467-8659.2008.01239.x (cited on pages 13, 16).

Mount, David M. and Sunil Arya [2010]. *ANN: A Library for Approximate Nearest Neighbor Searching*. 27th Jan 2010. https://www.cs.umd.edu/~mount/ANN/ (cited on page 17).

Noack, Andreas [2007]. “Energy Models for Graph Clustering.” *J. Graph Algorithms Appl.* 11.2 [2007], pages 453–480. http://www.emis.ams.org/journals/JGAA/accepted/2007/Noack2007.11.2.pdf (cited on page 11).

Opsahl, T. [2011]. *Why Anchorage is not (that) important: Binary ties and Sample selection.* 2011. https://toreopsahl.com/2011/08/12/why-anchorage-is-not-that-important-binary-ties-and-sample-selection/ (cited on page 17).

Peixoto, Tiago P. [2014]. “The graph-tool Python library”. *figshare* [2014]. doi:10.6084/m9.figshare.1164194. http://figshare.com/articles/graph_tool/1164194 (cited on pages 18–19).

Pupyrev, Sergey, Lev Nachmanson and Michael Kaufmann [2011]. “Improving Layered Graph Layouts with Edge Bundling”. In: *Graph Drawing: 18th International Symposium, GD 2010, Konstanz, Germany, September 21-24, 2010. Revised Selected Papers*. Edited by Ulrik Brandes and Sabine Cornelsen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pages 329–340. ISBN 3642184693. doi:10.1007/978-3-642-18469-7_30 (cited on page 3).

Telea, A. and O. Ersoy [2010]. “Image-Based Edge Bundles: Simplified Visualization of Large Graphs”. *Computer Graphics Forum* 29.3 [2010], pages 843–852. ISSN 1467-8659. doi:10.1111/j.1467-8659.2009.01680.x (cited on page 13).

Van Liere, R. and W. de Leeuw [2003]. “GraphSplatting: visualizing graphs as continuous fields”. *IEEE Transactions on Visualization and Computer Graphics* 9.2 [Apr 2003], pages 206–212. ISSN 1077-2626. doi:10.1109/TVCG.2003.1196007 (cited on page 13).

Vinet, Judd and Aaron Griffin [2017]. *Arch Linux*. 15th May 2017. https://www.archlinux.org/ (cited on page 17).

Zhou, H., Panpan Xu, X. Yuan and H. Qu [2013]. “Edge bundling in information visualization”. *Tsinghua Science and Technology* 18.2 [Apr 2013], pages 145–156. doi:10.1109/TST.2013.6509098 (cited on page 4).

Zhou, H., X. Yuan, W. Cui, H. Qu and B. Chen [2008]. “Energy-Based Hierarchical Edge Clustering of Graphs”. In: *2008 IEEE Pacific Visualization Symposium*. Mar 2008, pages 55–61. doi:10.1109/PACIFICVIS.2008.4475459 (cited on pages 5, 7).

Zhou, Hong, Xiaoru Yuan, Huamin Qu, Weiwei Cui and Baoquan Chen [2008]. “Visual Clustering in Parallel Coordinates”. In: *Proceedings of the 10th Joint Eurographics / IEEE - VGTC Conference on Visualization*. EuroVis’08. Eindhoven, The Netherlands: The Eurographics Association & John Wiley & Sons, Ltd., 2008, pages 1047–1054. doi:10.1111/j.1467-8659.2008.01241.x (cited on pages 5–6).