# Divided Edge Bundling for Directional Network Data

David Selassie, Brandon Heller and Jeffrey Heer

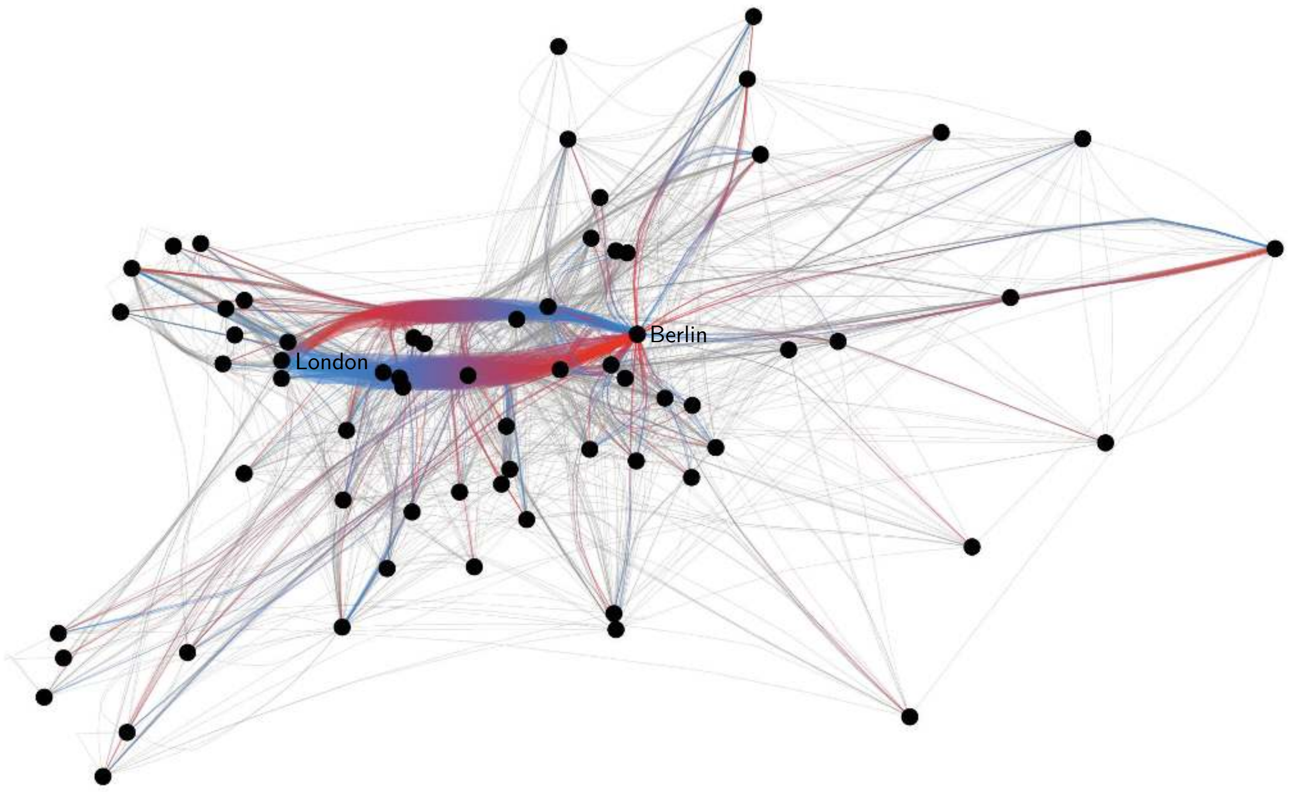


Fig. 1. European follower graph for GitHub, a hosted source code repository, highlighting connections to and from Berlin in color. Highlighted edges fade from blue (source) to red (target) to indicate direction. Divided edge bundling separates antiparallel edges into emergent "traffic lanes", enabling inspection of network asymmetries, such as the connections between Berlin and London. (235 nodes, 2101 edges; 18.1 seconds to bundle)

**Abstract**—The node-link diagram is an intuitive and venerable way to depict a graph. To reduce clutter and improve the readability of node-link views, Holten & van Wijk's force-directed edge bundling employs a physical simulation to spatially group graph edges. While both useful and aesthetic, this technique has shortcomings: it bundles spatially proximal edges regardless of direction, weight, or graph connectivity. As a result, high-level directional edge patterns are obscured. We present divided edge bundling to tackle these shortcomings. By modifying the forces in the physical simulation, directional lanes appear as an emergent property of edge direction. By considering graph topology, we only bundle edges related by graph structure. Finally, we aggregate edge weights in bundles to enable more accurate visualization of total bundle weights. We compare visualizations created using our technique to standard force-directed edge bundling, matrix diagrams, and clustered graphs; we find that divided edge bundling leads to visualizations that are easier to interpret and reveal both familiar and previously obscured patterns.

**Index Terms**—Graph visualization, aggregation, node-link diagrams, edge bundling, physical simulation.

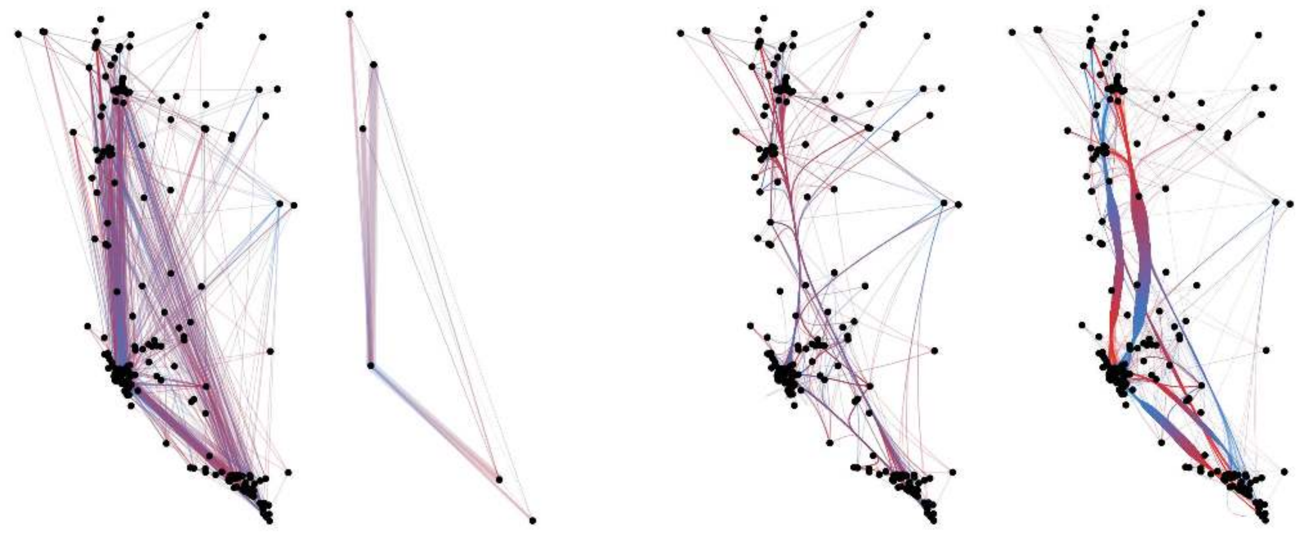
## 1 Introduction

Dense directed and weighted graphs present a common and difficult challenge in graph visualization. Tasks such as managing computer network traffic, shipping logistics, and communication records require interpretation of both geographic and directional patterns to gain insight. Social networks constitute another arena where connections among (often geo-located) entities are at the heart of any study.

Straight-edge node-link diagrams are an intuitive way to communicate graph structure for geo-located data, but they quickly suffer from occlusion issues with larger datasets. Figure 2(a) presents an unsuccessful visualization of open source software collaborations on the west coast of the United States based on data from GitHub [8]. The directionality of edges is encoded by a color gradient pointing from the source (blue) to the target (red); edge weight is encoded in the width of the edge. We could reduce the resulting visual clutter by optimizing node placement, but then the spatial dimensions would no longer faithfully communicate geographic patterns.

• *David Selassie, Brandon Heller and Jeffrey Heer are with the Computer Science Department of Stanford University, Stanford, CA 94305. E-mails: {selassid, brandonh}@stanford.edu, jheer@cs.stanford.edu*

*Manuscript received 31 March 2011; accepted 1 August 2011; posted online 23 October 2011; mailed on 14 October 2011.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org.*



(a) Unbundled (b) Clustered to metro areas (c) Force-directed edge bundling (d) Divided edge bundling

Fig. 2. Views of a subset of GitHub follower data on the United States west coast. As a reference, from bottom to top, the node clusters in these maps are: San Diego and Los Angeles, San Francisco, Portland, Seattle and Vancouver. Edges fade from blue to red along their length to indicate direction. Divided edge bundling reveals connections to the San Francisco area as having an asymmetry: more blue bundles leave the area than red bundles leave. (Unclustered 238 nodes, 1495 edges; 11.6 seconds to bundle)

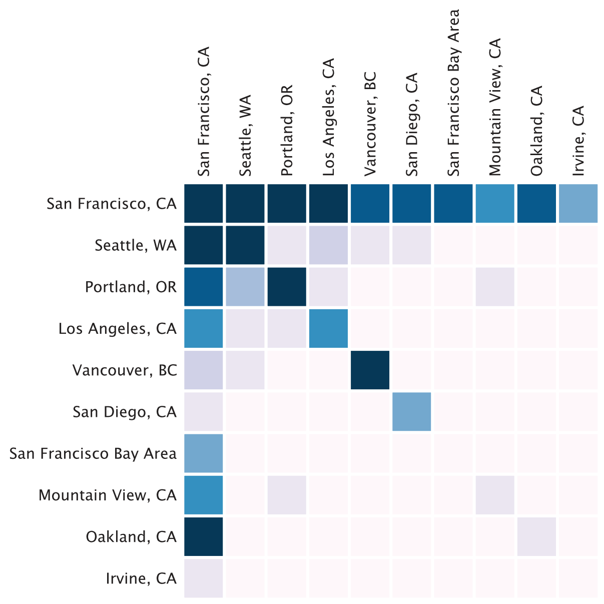


Fig. 3. Matrix diagram of clustered GitHub data. Darker cells indicate more connections from row to column clusters. The asymmetry in connections to and from San Francisco found in Figure 2(d) can be seen since San Francisco's row is darker than its column.

Another approach to clutter reduction is clustering. Many graph simplification techniques exist [14], but each makes assumptions about the data that may be inappropriate for a specific data set. Simplification can also obscure fine-grained or exceptional patterns that may be present. Figure 2(b) shows the GitHub data clustered to major metropolitan areas; although the graph is much simpler, higher-level directional trends and asymmetries still are not visible.

Matrix diagrams present an uncluttered edge-centric visualization of the data [6] at the cost of abstracting away familiar spatial correlations, such as geographic patterns. To make a large and sparse matrix diagram of geographically distinct nodes more digestable, one could apply graph simplification or aggregation methods before generating the diagram. Figure 3 presents a simplified version of the GitHub data using a matrix diagram. Although the patterns of connectivity are visible, one requires intimate knowledge of the data being represented (in this case the proximity of the major cities on the west coast of the U.S.) to draw geographic conclusions from this visualization. Moreover, comparisons must be made across the diagonal to assess asymmetries, requiring a potentially error-prone visual search when viewing large matrices.

As a result, we return to the node-link diagram. To reduce clutter without resorting to graph simplification, we can spatially bundle edges that traverse similar paths. For example, Holten & van Wijk's force-directed edge bundling [12] uses a physical simulation to cluster edges together. Figure 2(c) applies force-directed edge bundling to this GitHub network, revealing higher-level connectivity patterns not visible in the preceding graphs. However, current edge bundling approaches suffer from some shortcomings. Figure 2(c) provides little insight into directional patterns and does not effectively show the magnitude of weights in the bundled edges. In addition, spatially proximate edges are bundled regardless of graph topology; edges from disjoint subgraphs would be grouped. While researchers have proposed a variety of edge bundling methods [1, 5, 10, 12, 15], each ignores edge direction, graph connectivity, and edge weights when calculating edge layouts. To address these issues, we leverage the observation that spatially pairing edges can improve our ability to compare their attributes, and set out to design an algorithm that performs this pairing.

Thus, we introduce *divided edge bundling*, an extension of the force-directed edge bundling method [12]. By modifying the forces in the physical simulation, directional lanes appear as an emergent property of edge direction. By considering graph topology, we only bundle edges related by graph structure. Finally, we aggregate edge weights in bundles to enable more accurate visualization of total bundle weights. Bringing forward directional lanes and edge weights enables analysts to more easily spot asymmetries, while incorporating graph connectivity helps prevent spurious inferences. As a result, patterns previously visible only in a matrix view can now be seen in a node-link diagram with the added benefit of spatial or geographic context.

Figure 2(d) demonstrates the application of divided edge bundling to the same GitHub data. We now see asymmetries in the flow into and out of San Francisco, as indicated by the thicker outgoing (blue) bundles connected to the area. The magnitude of flows between cities, asymmetries in these flows, and the directions of edges may now be inspected at a high level.

The rest of the paper is organized as follows. After surveying related work on graph visualization, we present the details of divided edge bundling. We describe both our physical model and visual encoding choices, and we discuss salient implementation and performance details. We then present case studies applying divided edge bundling to a collection of real-world data sets.

## 2 Related Work

The visualization community has devised a variety of methods to improve the effectiveness of graph visualizations. Techniques for node-link diagrams include clutter reduction [3], node clustering [14], interactive focus+context techniques [18, 19], and edge bundling [1, 5, 10, 12, 15]. However, these methods often fail to accurately convey directional information, particularly for dense graphs with attribute-driven (e.g., geographic) node placement.

Matrix diagrams [6] solve some of these shortcomings, but can obscure nodes' spatial or geographic patterns. Matrix diagrams often encode edge weight using color or hue. Perceptual experiments [17] of standard data graphics (e.g., bar charts) have shown that length encoding is strongly preferable to color encoding of quantitative values; this suggests that edge width comparisons may be more accurate than color or hue comparisons of matrix adjacency cells. Attribute-based aggregation, as applied in Honeycomb [16], can improve scalability but retains these shortcomings. Hybrid approaches such as NodeTrix [9] reduce clutter by converting selected portions of the dataset to matrix diagrams; however, one must determine a clustering scheme and fine-grained geographic data cannot be depicted within the clusters.

**Force-Directed Edge Bundling**. The prior work most closely related to our own research is the growing literature on edge bundling methods. Researchers have proposed a variety of techniques, including hierarchical edge bundling [10], force-directed edge bundling [12], geometry-based edge clustering [1], multi-level agglomerative edge bundling [5], and grid-based methods [15]. While these methods have their advantages, they uniformly ignore edge direction, connectivity, and weight and therefore fall short in communicating patterns involving those characteristics.

We developed divided edge bundling as a set of extensions to Holten & van Wijk's force-directed edge bundling [12]. As our contributions build upon this prior work, we now describe their technique in detail. *Force-directed edge bundling* is a discrete time physical simulation that models each edge as a set of control points. Each control point interacts with adjacent ones via ideal Hooke's law springs and interacts with control points on other edges via a Coulombic interaction, as depicted in Figure 4.

The *spring force* between two adjacent control points $p_i$ and $p_j$ on edge $P$ comes from Hooke's law using a spring constant equal to a global spring constant $k_s$ times the number of control points $C$. The force acts along the vector between the points.

$$F_s(p_i, p_j) = k_s C |p_i - p_j| \tag{1}$$

Proportionally scaling the effective spring constant by $C$ gives each edge a constant stiffness independent of $C$.

The *Coulombic force* between edge control points $p_i$ and $q_j$ on distinct edges $P$ and $Q$ is an inverse radius force using a global Coulombic constant $k_C$. The force acts along the vector between the points.

$$F_C(p_i, q_j) = \frac{k_C}{|p_i - q_j|^2} \tag{2}$$

Holten & van Wijk reduce the computational complexity from $O(E^2C^2)$ to $O(E^2C)$ (where $E$ is the number of edges and $C$ is the number of control points) by having each control point attract only control points of the same index on other edges. They assert that this trick does not change the qualitative result of the edge bundling algorithm [12]; this has been our experience as well.

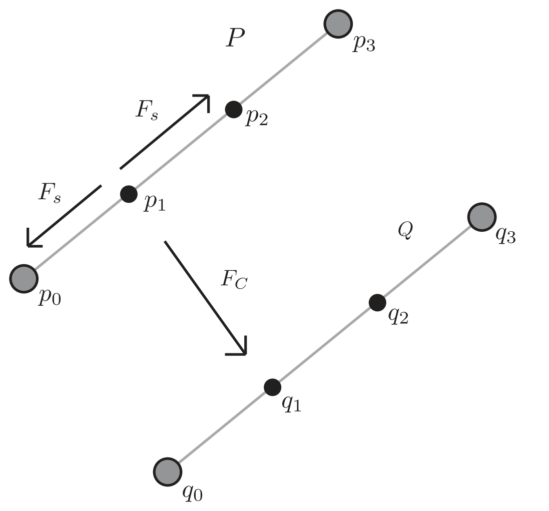


Fig. 4. Force-Directed Edge Bundling. Each control point of an edge is forced towards adjacent points with a spring force $F_s$ and toward a point on every other edge with a Coulombic force $F_C$. (Adapted from [12])

Holten & van Wijk also give a method for smoothing edges where the position of all control points in an edge are convolved with a Gaussian kernel. Over-bundled edges are spread apart by this convolution, allowing the viewer to roughly see the number of edges in each bundle.

*Compatibility Measures*. In order to prevent over-bundling — and thus avoid a visually inscrutable image — prior work introduced a set of *compatibility measures* that reduce inter-edge attraction [12]. As two edges $P$ and $Q$ diverge in length, position, angle, or projection overlap (called "visibility"), the force between their control points is multiplied by the product of these measures, $C_e(P,Q) \in [0,1]$ to reduce it. These measures are defined with respect to the fixed positions of the graph nodes, not to the positions of the movable control points.

## 3 Divided Edge Bundling

Building directly upon Holten & van Wijk's approach, we developed *divided edge bundling* to enable the perception of connectivity features previously discernible only in matrix views, while retaining the intuitive encoding and spatial placement of a node-link diagram. In its original form, force-directed edge bundling does not incorporate edge directions, connectivity patterns, or edge weights; our technique addresses these three omissions. First we describe our modifications to the Holten & van Wijk's physical simulation for edge bundling layout; then we describe our visual encoding decisions.

### 3.1 Directional Lanes

We employ a partial spatial (position) encoding for edge direction: edges that travel in antiparallel directions are not bundled directly on top of each other, but are separated to form two directional lanes like a divided highway. Although the efficacy of this spatial encoding has not been studied, it has some precedent in successful similar designs, such as Fekete et al.'s [4] approach of using edge curvature to indicate direction, and it corresponds with the common experience of highway systems. Given a cultural convention, viewers who encounter a divided highway can quickly discern the directions of a lane of travel by virtue of which side of the road it is on, even without other clues. These directional lanes arise as an emergent behavior of the physical simulation by modifying the potential function of the Coulombic force to depend on edge direction.

This modification to the potential requires a definition for relative edge direction. Consider each edge to be a vector from a source node position to a target node position; two edges $P$ and $Q$ go in the same direction if their dot product is positive and go in the opposite direction if their dot product is negative. For a point $p_i$ in $P$ attracted to a point $q_j$ in $Q$, we refer to the location of the potential minimum as $m_j$. When $P$ and $Q$ are going the same direction, $m_j$ is located at $q_j$; this behavior

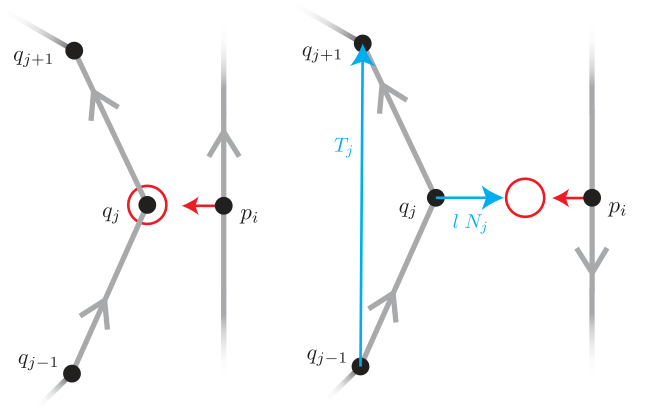


Fig. 5. Directional lanes emerge by attracting control points to different locations (red circles) depending on the relative direction of edges. (a) If edges $P$ and $Q$ travel in the same direction ($P \cdot Q > 0$), control points on $P$ are attracted to the position of control points on $Q$. (b) If edges $P$ and $Q$ travel in opposite directions ($P \cdot Q < 0$), control points on $P$ are attracted to a point a lane width $l$ to the "right" of $Q$.

is identical to standard force-directed edge bundling. However, when $P$ and $Q$ are going opposite directions, $m_j$ moves to a point a distance $l$ to the "right" of $q_j$, where $l$ is a defined lane width, as is depicted in Figure 5. In other words, when edges are antiparallel, we translate the potential function by $l$.

"Right" is defined as the 90° rotation of the vector $T_j$ between $q_{j-1}$ and $q_{j+1}$, which is $N_j$ (the unit normal at $q_j$), times $l$ plus $q_j$.

$$T_j = q_{j+1} - q_{j-1} \tag{3}$$

$$N_j = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \frac{T_j}{|T_j|} \tag{4}$$

Since "right" is defined relative to the local edge direction, edges that are antiparallel constructively force themselves into separate lanes.

We chose an inverted Lorentzian (see Figure 6) with a minimum at $r = 0$, where $r$ is the distance between two interacting points, as our specific potential function $U(r)$. We chose this potential because it does not have any discontinuities or singularities and is amenable to producing stable conformations in a discrete physical simulation. That said, any smooth potential function that has a minimum at $r = 0$ could be used and cause lanes to emerge.

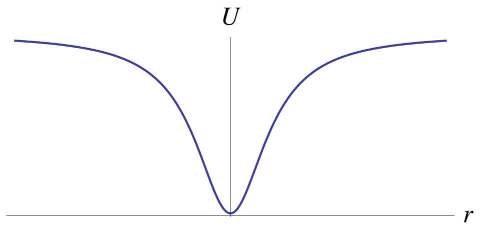


Fig. 6. The potential function we use is an inverted Lorentzian, although any smoothly varying function with a minimum at $r = 0$ can be used.

The force $F(r)$ is the negative gradient of the potential. Parameter $s$ determines the slope into the potential well and the well depth is determined by an effective Coulombic constant equal to a global Coulombic constant $k_C$ divided by the number of control points $C$. The value $r$ is the distance between the current control point $p_i$ and the potential minimum $m_j$ for its corresponding control point $q_j$.

$$U_C(p_i, q_j) = \frac{-sk_C}{\pi C(s^2 + |p_i - m_j|^2)} \tag{5}$$

$$F_C(p_i, q_j) = -\nabla U_C(p_i, q_j) = \frac{-sk_C|p_i - m_j|}{\pi C(s^2 + |p_i - m_j|^2)^2} \tag{6}$$

The potential minimum $m_j$ is defined in the same way as above:

$$m_j = \begin{cases} q_j & \text{if } P \cdot Q > 0 \\ q_j + lN_j & \text{if } P \cdot Q < 0 \end{cases} \tag{7}$$

Inversely scaling the effective Coulombic constant by the $C$ points gives each edge a constant Coulombic charge, causing the character of the bundling to be independent of $C$.

We implemented a modified version of the complexity reduction trick described previously, where each control point $p_i$ only interacts with a single control point on the other edge. Since all edges have the same number of control points, $p_i$ interacts with $q_i$ on the other edge if the edges are going the same direction, or $p_i$ interacts with $q_{C-i}$ on the other edge if the edges are going opposite directions. This index inversion based on edge direction is performed so that each control point interacts with the most spatially appropriate control point on the other edge. This complexity reduction trick was found to have little effect on the final visualization, as in prior work [12].

As a simple demonstration of this extension, Figure 7(a) and Figure 7(b) compare the results of bundling a synthetic graph with and without directional lanes. Emergent lanes physically separate edges of different directions and reveal the simple pattern that standard bundling obscures. Although the unbundled version of this simple graph will also show the directional pattern, edge occlusion often obscures such patterns as networks become more dense.

## 3.2 Connectivity Compatibility

Graph topology is an interesting omission from existing compatibility measures in force-directed edge bundling. Edges that happen to be close in space are bundled, irrespective of their proximity in the graph structure. As edges in one subgraph do not necessarily provide context for edges in the other, disjoint subgraphs should not be bundled.

The goal of incorporating connectivity into the edge bundling algorithm is to ensure that the observed high-level patterns are reflective of the underlying structure of the graph. We now introduce the *connectivity compatibility* $C_c \in [0,1]$ which quantifies how closely edges are related by graph distance. Between edges $P$ and $Q$, $C_c$ is defined as

$$C_c(P,Q) = \frac{1}{1 + D_{\min}(P,Q)} \tag{8}$$

where $D_{\min}(P,Q)$ is the number of edges in the minimum length path connecting either of the nodes of $P$ to either of the nodes of $Q$, disregarding edge direction and weight. If there is no possible route $D_{\min}$ is assigned to $+\infty$. Thus, if the two edges share a node $C_c = 1$, if they are in disjoint subgraphs $C_c = 0$. This $C_c$ is multiplied by the other compatibility coefficients described in the original technique, and the product is used to scale the Coulombic forces between $P$ and $Q$. $D_{\min}$ values for all edges can be calculated once in a preprocessing phase using the Floyd-Warshall algorithm.

Figure 7 compares two graphs bundled with and without connectivity compatibility. Prior to incorporating connectivity, the existence of two disjoint subgraphs is completely obscured, nor is it immediately obvious in the unbundled graph (c.f., Figure 9(a)).

The connectivity coefficient described here will severely limit bundling in data sets with many disjoint subgraphs; the user may need to decide if this extension is appropriate to use with their data.

## 3.3 Edge Weights

To meet our goal of faithfully incorporating weighted edges, a third extension is needed. We assume edge weight corresponds to the importance of an edge in the graph, and that more important edges should exert more influence on the final structure of the graph.

As the simulation proceeds, the potential function moves control points, and those that are in compatible edges begin to overlap. We call a group of edges with overlapping control points a *bundle*. Bundles produce what we perceive to be a single merged edge; these edges and control points are still distinct in the physical simulation, but they just happen to be overlapping.

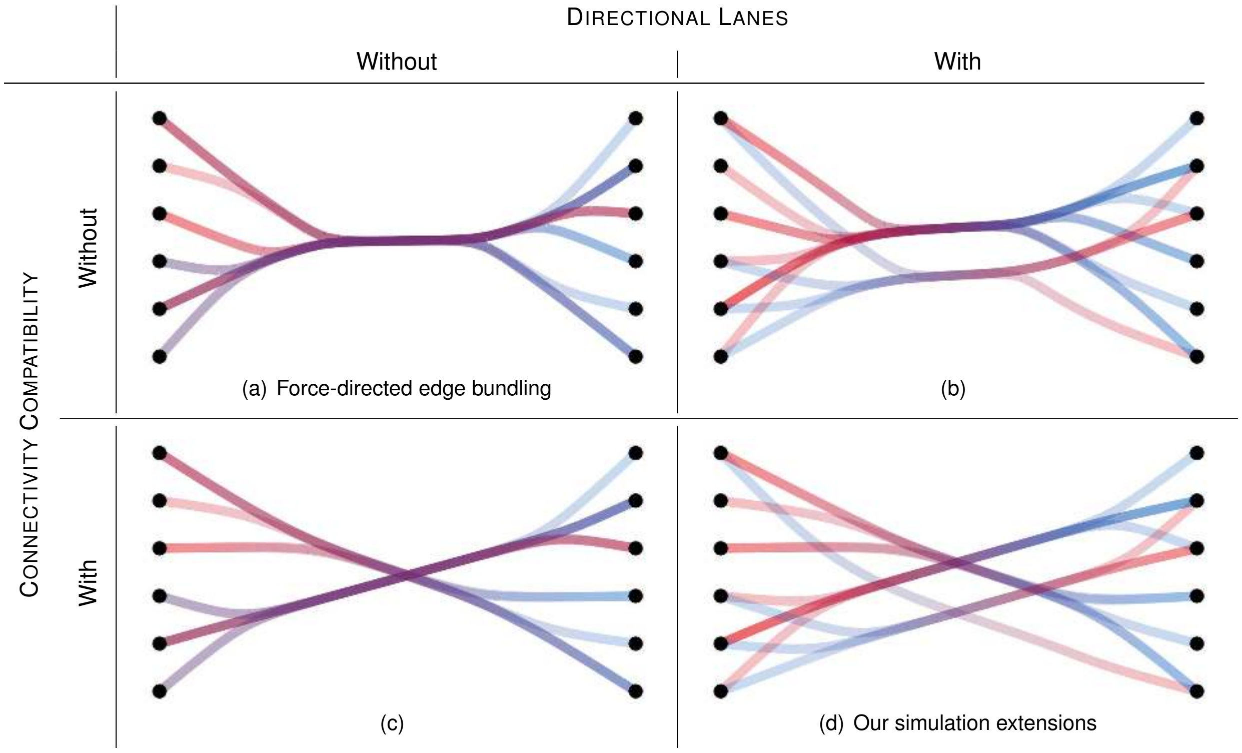

DIRECTIONAL LANES
Without
With
CONNECTIVITY COMPATIBILITY
Without
With
(a) Force-directed edge bundling
(b)
(c)
(d) Our simulation extensions

Fig. 7. Examples of edge bundling simulation extensions. Directed lanes form when antiparallel edges attract each other to a given lane width apart. Taking into account graph topology via a *compatibility coefficient* reveals disjoint subgraphs that were previously obscured.

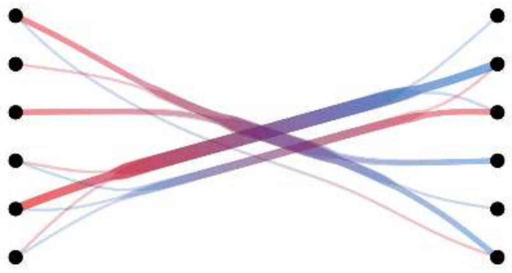


Fig. 8. Divided edge bundling including weight extensions. Edge widths are scaled to encode bundle weight. Asymmetry of flow becomes apparent: more edges point from right to left.

Individual edges that have a weight equal to the sum of the weights in a bundle should behave similarly to that bundle as a whole in the physical simulation. As a bundle gathers edges, it becomes more influential in the simulation: it contains more edges with control points that influence other edges. By scaling the force an edge exerts on other edges by its weight, we can ensure that bundles and single edges with the same total weight act equivalently.

First, individual edge weights are normalized to $\in [0,1]$. Then our physical simulation scales an edge's internal spring force of the original technique and its modified external Coulombic force by this normalized weight. Heavier edges thus exert more influence over the bundled graph structure. The intra-edge spring forces are relatively stronger, resulting in heavier edges being less flexible, and the inter-edge Coulombic forces are relatively stronger, resulting in heavier edges attracting other edges more. In the case of unweighted graphs, all the edge weights are equal and the technique reduces gracefully.

## 3.4 Visual Encodings

Our goal of highlighting overall patterns in graph structure must be met by our choice of visual encoding. A number of visual encodings are available to convey individual edge weights and directions, and in this section we describe our choices and the rationale behind them.

### 3.4.1 Edge Direction Encoding

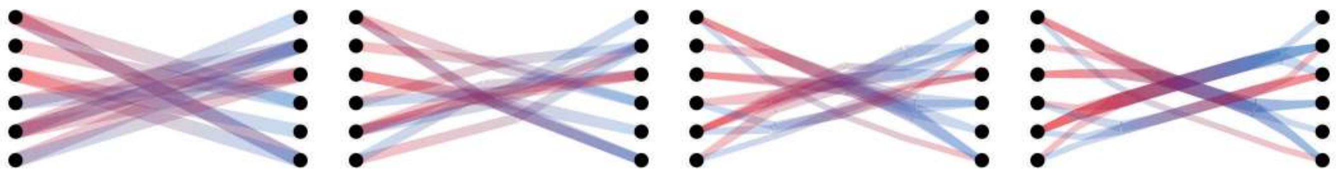
Visualizations of directed graphs often appear more cluttered than their undirected counterparts due to extra symbols (e.g., arrow heads) or other decorations indicating edge direction. Within divided edge bundling, the relative spatial position of lanes takes on some of the burden of encoding edge direction. In addition, prior empirical work has shown that color gradients are an effective way to communicate direction to viewers [13]. Accordingly, we use a blue (source) to red (target) gradient to signify direction. As a result, a viewer inspecting any edge in isolation can determine its absolute direction. It is also relatively easy to see if a region of the graph is predominantly source or target nodes by comparing the average color.

The blue to red gradient encoding does have a shortcoming. The midpoint of an edge will be purple regardless of direction, so comparing the direction of two edges at their midpoints will require scanning along the length of an edge until a color change is detected. This difficulty is highlighted in the center of Figure 7(c).

Prior work [11] has explored the efficacy of other directional encodings for straight line node-link diagrams, with tapered edges performing best in a user study. A tapered edge encoding may be less desirable when used in combination with edge bundling, as overlapping edges of varied width may not produce intuitive and perceptually effective views. As presented, bundling poses a general problem for visual encodings that vary along the length of an edge, as information can be obscured when edges at different stages of traversal are placed on top of each other.

### 3.4.2 Edge Weight Encoding

Since the edge bundling process spatially condenses edges, we need to augment bundling to support the perception of total bundle weight. Holten & van Wijk employ Gaussian smoothing of control point positions to tease apart bundles into component edges, resulting in bundles with more edges appearing larger. Since this approach undoes some of the bundling, it may increase the amount of clutter. Moreover, the effect of the smoothing is quite limited at the middle of an edge, and so does not provide a uniform visual indicator of bundle weight. In addition to smoothing, Holten & van Wijk explored encoding the number



(a) 0 cycles – 0 control points (b) 1 cycle – 1 control point (c) 2 cycles – 3 control points (d) 3 cycles – 7 control points

Fig. 9. Automated, iterative edge bundling. Each cycle (left to right) increases the number of control points and halves the simulation time step.

of edges as a colored heatmap; however, edge color can no longer encode direction. Another possibility is to render semi-transparent edges so that thicker bundles appear more vivid than single edges.

To further improve perception of bundle weights, we have modified the edge bundling routine to recognize when multiple edges overlap and to draw those edges as if they carry the combined weight of all the edges in this bundle. The weight $g$ of a bundle at a given control point $p_i$ in an edge $P$ is the sum of all the weights of the edges $Q$ going the same direction as $P$ $(P \cdot Q > 0)$ and exceeding a threshold compatibility criteria (we use $C \geq 0.05$) with control points $q_j$ in any edge $Q$ within a certain cutoff distance $d$ of $p_i$, where $d$ is the visible edge thickness of $P$ calculated using only the edges own weight.

We globally normalize these $g \in (0,1]$ before computing the visual thickness of an edge at that control point. This weight normalization step makes a single set of visual parameters more applicable to a wider array of input graphs. We define the visible edge thickness $D$ to be

$$D = w g^p \tag{9}$$

where the edge width constant $w$ and edge width exponent $p$ are configurable visual parameters. As $g \leq 1$, the edge width power $p$ parameter controls how quickly the visual thickness of edges falls off with weights less than the maximum. This allows more subtle asymmetries in edge and bundle weight to be seen.

Note that calculating bundle weights has no effect on the physical simulation. Even when the control points of two edges have collapsed upon each other, our implementation renders both edges; since bundle weight is only calculated at mesh control points, as edges join or leave a bundle, the resulting jumps in bundle weight can cause visual artifacts that look like feathers. In order to ease visual interpretation of edge overlap, we render each edge with an alpha value of 0.25.

Figure 8 demonstrates our encoding of bundle weight, which makes apparent the difficult-to-discern edge count asymmetry in Figure 7(d).

## 4 Implementation and Performance

We implemented[1] these techniques using a GPU-accelerated physical simulation with a leapfrog integrator [7] written for MacOS X in Objective-C and OpenCL. Our prototype system also enables interactive parameterization and exploration of bundled graphs. All figures in this paper were produced using our prototype application. All times were measured on a 2010 Apple MacBook Pro with a 2.66 GHz Intel Core i7 CPU and a NVIDIA GeForce GT 330M GPU.

### 4.1 Computational Complexity

The complexity of the described physical simulation is the same as force-directed edge bundling, $O(E^2C)$ per frame, since force computation uses the complexity reduction trick described in §3.1. The bundle weight calculation described above can be performed while calculating the force on a control point, and adds no complexity.

Running the physical simulation requires preprocessing the input graph to generate compatibility coefficients. Generating these values requires an all-pairs shortest path computation, a one time cost of $O(N^3)$ in our implementation, though for sparse graphs like our examples, Johnson's Algorithm with Fibonacci heaps drops this cost to $O(N^2 log(N) + NE)$. Finding the resulting compatibility coefficients for every edge pair incurs a one-time cost of $O(E^2)$.

[1]Our implementation can be downloaded at `http://selassid.github.com/DividedEdgeBundling`.

| Figure | Nodes | Edges | Preprocessing | Simulation |
|---|---|---|---|---|
| 1 | 69 | 1537 | 1.9 sec | 18.1 sec |
| 2(d) | 238 | 1495 | 1.5 sec | 11.6 sec |
| 10(b) | 235 | 2101 | 2.5 sec | 23.7 sec |
| 12 | 44 | 142 | 0.1 sec | 1.0 sec |

Table 1. Size and generation times of images presented in this paper. All times were measured on a 2010 Apple MacBook Pro with a 2.66 GHz Intel Core i7 CPU and a NVIDIA GeForce GT 330M GPU.

It is important to reiterate that the extensions described here enhance the visible patterns but do not increase the computational complexity of the physical simulation; only the complexity of the one-time preprocessing step is increased.

### 4.2 Automated Image Generation

Holten & van Wijk outline an iterative technique to quickly converge on an approximation of the most stable bundled graph; we apply a similar approach. We first set up a simulation with a large initial time step ($dt = 40$) and subdivide each edge into two control segments. After a specified number of simulation steps (30), the time step is halved and the number of control segments in each edge is doubled, completing one cycle. We perform a total of five cycles, over which the edge bundling approaches a stable conformation. The initial cycles with fewer control points but larger time steps allow the coarse structure of the bundled graph to quickly form; later cycles refine these structures. This iterative process is illustrated in Figure 9.

For the graphs presented here generated using the automated iterative technique, most of the computational time is spent in the physical simulation, and not in the preprocessing step, as revealed in Table 1.

### 4.3 Edge Bundling Parameters

The visualizations produced by our edge bundling technique depend on a number of parameters. Thus far we have introduced seven parameters: the spring constant $k_s$, the Coulombic constant $k_C$, the number of control points $C$, the lane width $l$, the edge width $w$, the edge width exponent $p$, and the Lorentz function width $s$. There are also two additional parameters: the simulation time step $dt$ and a velocity damping friction coefficient $f$.

Table 2 provides a brief physical explanation of the parameters and the default values used to generate images in this paper (ranges indicate extent during automated image generation). Table 3 depicts the effects of varying the selected parameters. There is interplay between these parameters: since the spring force straightens edges and the Coulombic force bends edges, only the ratio of the two has an effect on the final layout of control points. A lower spring force to Coulombic force ratio results in more flexible edges that readily form bundles, while a higher ratio results in stiffer edges.

Like all discrete physical simulations, $dt$ needs to be set low enough and $f$ high enough to produce stable graphs without simulation artifacts. Decreasing $dt$ comes at the expense of increasing the number of iterations required to create a final bundled graph, and therefore the computation time to produce an image.

### 4.4 Scalability

Ideally, our technique should reliably scale with the size of the input graph in every dimension: number of edges, number of nodes, node

| Parameter | Physical Analogy | Default |
|---|---|---|
| $k_s$ | Edge Stretchiness | $0.5 \times 10^{-3}$ |
| $k_C$ | Edge Attraction | $2.0 \times 10^{4}$ |
| $C$ | Edge Resolution | $< 35$ |
| $l$ | Directional Lane Width | 25 |
| $w$ | Visual Edge Width | 7.0 |
| $p$ | Edge Width Fall-Off | 1.25 |
| $s$ | Attractive Force Range | 30.0 |
| $dt$ | Simulation Time Step | $40 - 1.25$ |
| $f$ | Friction | 0.2 |

Table 2. Physical simulation parameters for divided edge bundling and their default values. Ranges indicate values spanned during the iterative automatic image generation procedure described in §4.2.

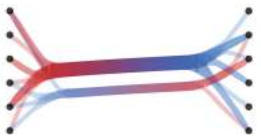
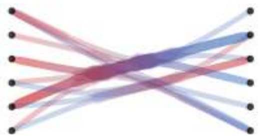
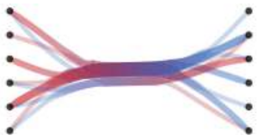
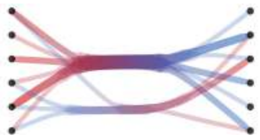
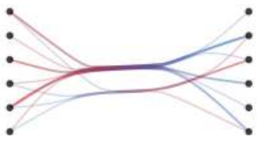
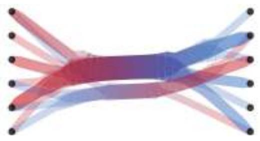
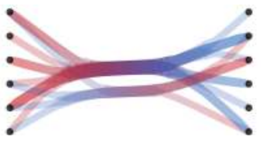
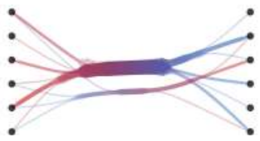
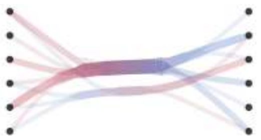
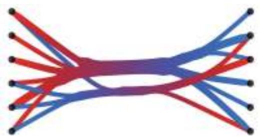
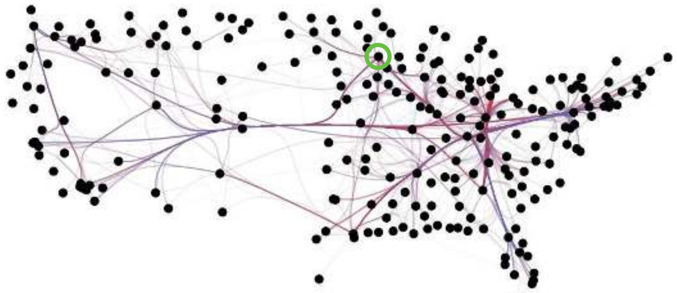
| Param. | Low Value | High Value |
|---|---|---|
| $\frac{k_s}{k_C}$ |  |  |
| $l$ |  |  |
| $w$ |  |  |
| $p$ |  |  |
| *alpha* |  |  |

Table 3. Effects of simulation and visual parameters. Some parameters are complementary, so their ratio determines the final layout.

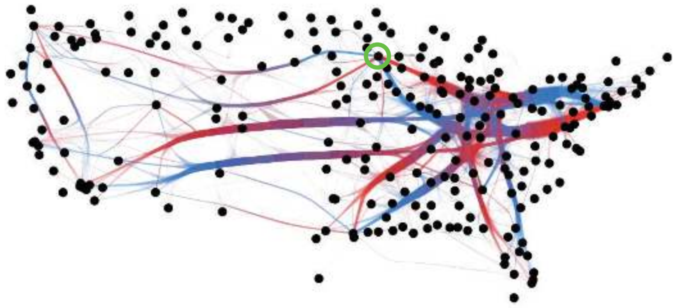
placement, edge weight, and number of control points. Both the spring and Coulombic forces, as previously described, scale with the number of control points $C$. Multiplying the effective spring constant by $C$ and dividing the effective Coulombic constant by $C$ keeps the same physical characteristics for an edge while allowing for variable resolution.

The input node positions are proportionally scaled to fit in a box of constant dimension (we use a square with a side of 1000.0 units) and the edge weights are normalized before the simulation begins. This normalization allows the bundling technique to be more flexible since it depends only on the relative weights and positions of edges.

As the density of edges in a graph grows, the total Coulombic force on any control point also increases. To counter this, the global Coulombic constant is divided by the square root of the number of edges, which physically corresponds to keeping the simulation's average charge density approximately constant. If this step is omitted, the force acting on a control point can exceed the threshold for simulation stability as you add more edges to the graph.



(a) Smoothed force-directed edge bundling



(b) Divided edge bundling

Fig. 10. Domestic flights in the United States. Edges fade from blue to red to indicate direction. The circled node is Minneapolis; the connectivity coefficient prevents the over-bundling of its westbound and eastbound edges. (235 nodes, 2101 edges; 23.8 seconds to bundle)

## 5 Evaluation

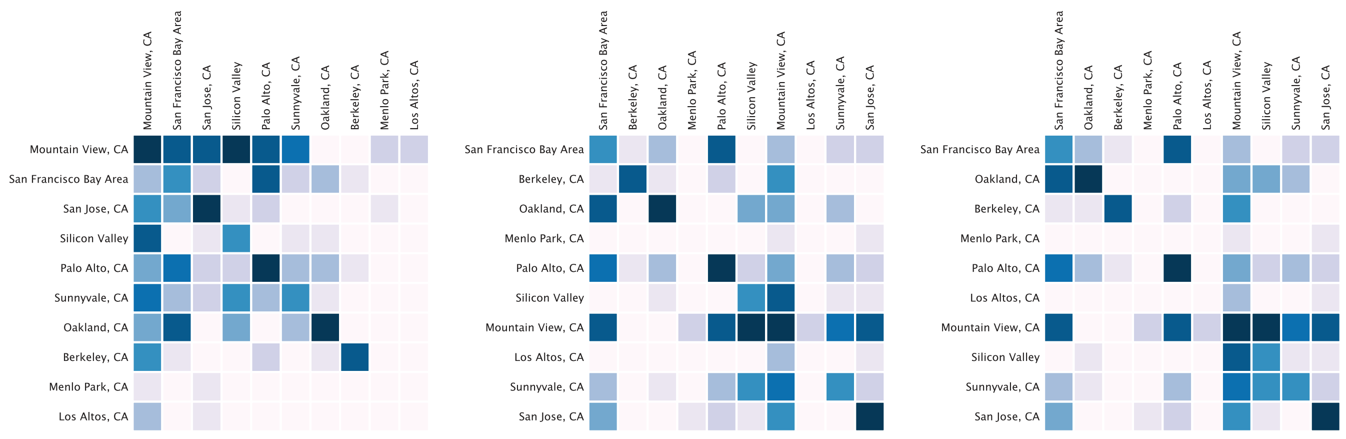
The original force-directed edge bundling technique enables viewers to see patterns that are difficult to see in a standard node-link diagram. While edge bundling may obscure some patterns, even more patterns are likely being obscured in an undifferentiated "hairball" graph. As a first step toward evaluating divided edge bundling, we apply the technique to some real-world data sets. We first compare to the original force-directed edge bundling technique using airline flight data; then, we compare to matrix diagrams using a social graph. These examples show how divided edge bundling enables a viewer to identify high-level directional edge direction, weight, and connectivity patterns.

### 5.1 Airline Flights

Figure 10 shows a network of U.S. domestic airline traffic (235 nodes, 2101 edges), visualized using (a) force-directed edge bundling with smoothing and (b) divided edge bundling. Directional lanes bring order to the formerly over-bundled right side of the graph; it becomes easier to follow edges because of the directional lanes.

Divided edge bundling enables a viewer to conclude that almost all of the links in this graph are symmetrical, i.e., the same number of edges travel in both directions between any two nodes, as demonstrated by the equal thickness lanes. Patterns seen with force-directed edge bundling are still visible: major airports are prominently shown, with thick edges leading to these nodes.

Without directional lanes, a viewer cannot come to this conclusion. Edges more closely track their original direction and connectivity compatibility prevents over-bundling. Major airports are still visible as sources in the graph. As an example of the connectivity coefficient in action, the circled airport, Minneapolis, is revealed to have large numbers of both eastbound and westbound flights in Figure 10(b), while these edges are more difficult to interpret in 10(a). Including bundle weights makes flows in the graph more apparent. These features of the generated visualization support our claim that divided edge bundling better reveals patterns regarding edge direction and edge weight.



(a) Ordered by edge weight (b) Ordered by latitude (c) Ordered by longitude

Fig. 11. GitHub follower data in the San Francisco Bay area as a matrix diagram. Row nodes follow column nodes; the shade of each square indicates the relative number of followers. Axes are organized by (a) the number of total edges, (b) latitude, and (c) longitude. The patterns are similar to those shown in Figure 12, except that spotting geographic trends requires a mental mapping from city name to geographic location. In this case, one-dimensional spatial node orderings are insufficient to show two-dimensional geographic patterns in the data.

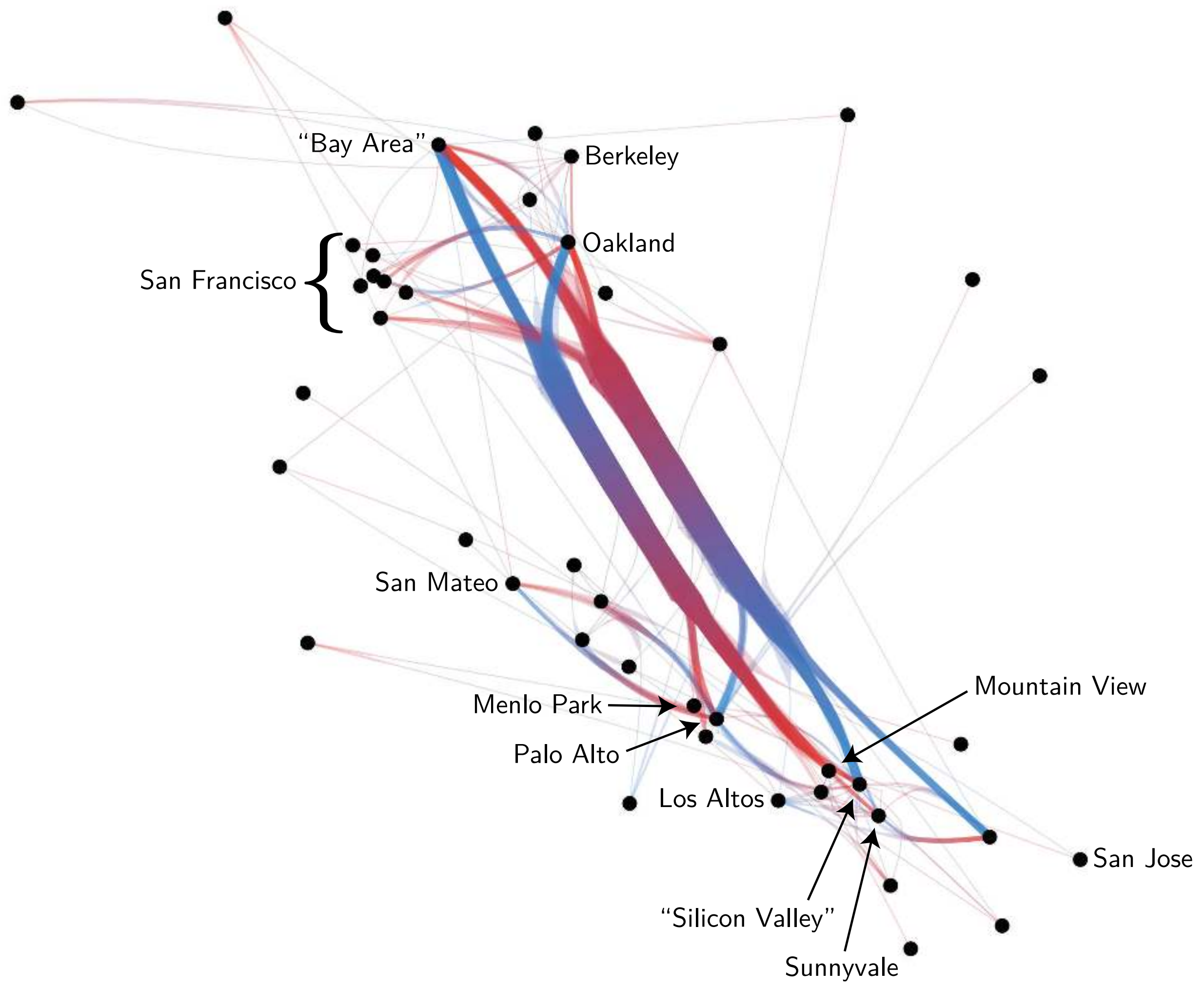


Fig. 12. GitHub follower data in the San Francisco Bay area with divided edge bundling. Edges fade from blue to red to indicate direction. In spite of San Francisco being represented as a cluster, edges connecting all of those points are still bundled. An asymmetry in cross-bay connections can be seen in the thickness of the two major parallel bundles. (44 nodes, 142 edges; 1.0 seconds to bundle)

## 5.2 GitHub Follower Data

We previously introduced GitHub follower data in Figure 2 as a way to motivate our work and briefly showed how divided edge bundling reveals geographic patterns [8]. Given the task of uncovering these types of patterns, we now attempt to compare the efficacy of a matrix diagram to divided edge bundling on a subset of the GitHub data. The matrix diagram in Figure 11 shows the ten cities[2] with the most GitHub followers in the San Francisco Bay area. The relative number of followers in each link is encoded by shades of blue and the nodes in each respective diagram are sorted by followers, latitude, and then longitude. Row nodes follow column nodes; the top row of Figure 11(a) represents links from Mountain View. The square in the Oakland column of that row is white, signifying that there are few links from Mountain View to Oakland. Compare this to the diagonally opposite light blue square representing more weight in the reverse edge, and we can conclude that more Oakland users are following Mountain View users than the reverse. We notice that this graph is fairly balanced across the diagonal, though a few connections have asymmetries. The two spatial orderings in Figure 11(b) and 11(c) help translate node labels to map positions, but do not reveal larger geographic patterns.

We now turn to Figure 12, which shows divided edge bundles for 57 geographic locations with GitHub users in the San Francisco Bay area. This diagram needs less explanation; edges fade from blue to red to indicate a following relationship. Many directional and geographic features become apparent, and we see patterns due to clusters of nearby nodes combining into a meaningful group that does not appear in the matrix diagrams of the top 10 cities. Notice that even though there are many nodes that represent locations within San Francisco[3], it is still possible for edges from the collection of nodes to group together and show patterns. Due to this segmentation, rank ordering nodes by in-flow or out-flow in a list or matrix diagram does not reveal that San Francisco is influential in this dataset.

We notice that the majority of connections are bundled into cross-bay lanes, and that the largest connections are asymmetric; more programmers in the South Bay follow those in the North Bay than vice versa, as seen by the thicker right bundle running up the length of the bay. This pattern is not obvious in any of the matrix diagrams presented. Oakland has more edges leaving it, shown by the predominance of blue around the node, which was previously seen in the asymmetry between Oakland's column and row in the matrix diagrams.

Many of the users on GitHub tag their location as "Bay Area," which is geolocated to a nonexistent place west of Berkeley in the water. Although this quirk can be exposed by any visualization technique that retains spatial encoding, meaningful trends can still be extracted from it using divided edge bundling. The "Bay Area" node primarily connects to points in the south part of the bay, seen by the thicker bundles leaving in that direction, suggesting that the node represents more South Bay programmers. San Mateo also mostly has connections to the South Bay, but links are evenly geographically distributed for most other nodes, showing an unexpected social bias.

There is no absolute winner to the comparison between matrix diagrams and divided edge bundling. Matrices show self edges and clearly separate individual links, but at the cost of extra mental effort to map labels and groups of labels to geographic locations. Divided edge bundling diagrams present a more easily-explained, easily-perceived view of flow in a way that aggregates nearby edges automatically, but the technique does not depict self edges and inherently hides detail in aggregation. If the task is to extract fine pairwise comparisons from a graphic, matrices are still a good choice, but if the task is to uncover high-level patterns, we find divided edge bundling faster and more intuitive. As always, it is critical to consider users' goals and constraints.

## 6 Future Directions

Divided edge bundling might be extended in a number of ways:

*Applicability*. The techniques described here are not limited to Holten & van Wijk's approach; other forms of bundling could be enhanced with the ideas here to account for graph topology and directional information. Further work on the connectivity compatibility is necessary to make it more applicable for graphs with different topological motifs and to properly account for edge weight.

*Joint Node/Edge Layout*. A major question unaddressed in both the prior and current work is the interplay between node layout and edge bundling; Holten & van Wijk's compatibility coefficients are exclusively a function of node position, so layout greatly affects bundling. Automated graph layout algorithms that tend to orthogonalize edges cause force-directed edge bundling techniques to coalesce edges ineffectively. Layout approaches that jointly optimize node placement and edge bundling might enable improved pattern perception.

*Scale Independence*. Normalizing node position brings us only part of the way toward a scale-independent technique — one that works well regardless of zoom level. The current implementation of divided edge bundling effectively supports only one scale, since lane width is fixed. Patterns that are contained entirely within the lane width often suffer reduced fidelity, due to these spatially proximal antiparallel edges bulging apart. Future extensions to divided edge bundling might allow multi-level analysis to occur on large datasets by dynamically varying the effective lane width.

*Visual Encodings*. A systematic exploration of visual encodings of edge direction in combination with bundling might also lead to improved results. Novel encodings might leverage the reduced amount of directional information that each edge has to convey, as the relative spatial position of a divided bundle already conveys some direction information. New directional encodings might free other visual variables (e.g., color) to convey additional edge information.

*Interaction*. Although our prototype does include interaction techniques (e.g., selection and zooming), future research might explore ways to interact with bundled graphs more effectively. A real-time physical simulation could react to selection queries and interactively reveal less prominent patterns in novel ways. For example, the techniques of Wong et al. (EdgeLens [19] and Edge Plucking [18]) might be adapted directly, as force-directed edge bundling techniques already make use of flexible edges.

*Continuous Updates*. Every graph in this paper visualizes a static data set, but many graphs, such as shipments, network traffic, and social graphs, are continually changing. Edge bundling pulls together edges in a way that minimizes potential functions; incremental updates of these edge weight adjustments, as well as the number of edges changing, might yield a different result than starting from scratch.

Finally, an alternative implementation of edge bundling might employ constraint based layout algorithms. The work of Dwyer [2] on constraint based graph layout might be applied to edge control points to bundle edges effectively.

## 7 Conclusion

In this paper, we presented divided edge bundling, an extension to force-directed edge bundling that allows adept visualization of directed graphs. Our technique is particularly well-suited to networks with predefined spatial coordinates. By incorporating directional lanes, graph topology, and edge weights, divided edge bundling reveals patterns of connectivity and symmetry obscured by existing node-link visualizations. The resulting visualizations offer many of the same insights as matrix diagrams, but without sacrificing the benefits of a spatially faithful and intuitive layout.

[2] Beyond ten cities the matrix becomes sparse.

[3] Coders in San Francisco seem to have neighborhood pride and have entered more specific location data in GitHub than just the name of their city.

## References

[1] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization & Computer Graphics*, 14(6):1277–1284, oct 2008.

[2] T. Dwyer. Scalable, versatile and simple constrained graph layout. *Computer Graphics Forum*, 28(3):991–998, 2009.

[3] G. Ellis and A. Dix. A Taxonomy of Clutter Reduction for Information Visualization. *IEEE Transactions on Visualization & Computer Graphics*, 13(6):1216–1223, 2007.

[4] J. Fekete, D. Wang, N. Dang, A. Aris, and C. Plaisant. Overlaying graph links on treemaps. *IEEE Symposium on Information Visualization Conference Compendium (demonstration)*, 2003.

[5] E. Gansner, Y. Hu, S. North, and C. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 187 –194, march 2011.

[6] M. Ghoniem, J. D. Fekete, and P. Castagliola. On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. *Information Visualization*, 4(2):114–135, 2005.

[7] E. Hairer, C. Lubich, and G. Wanner. Geometric numerical integration illustrated by the St ormer-Verlet method. *Acta Numerica*, 12:399–450, May 2003.

[8] B. Heller, E. Marschner, E. Rosenfeld, and J. Heer. Visualizing collaboration and influence in the open-source software community. In *Mining Software Repositories*, 2011.

[9] N. Henry, J.-D. Fekete, and M. J. McGuffin. NodeTrix: A hybrid visualization of social networks. In *IEEE Transactions on Visualization & Computer Graphics*, pages 1302–1309, 2007.

[10] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization & Computer Graphics*, 12(5):741–748, 2006.

[11] D. Holten, P. Isenberg, J. J. van Wijk, and J.-D. Fekete. An Extended Evaluation of the Readability of Tapered, Animated, and Textured Directed-Edge Representations in Node-Link Graphs. In *2011 IEEE Pacific Visualization Symposium (PacificVis)*, pages 195–202. IEEE, 2011.

[12] D. Holten and J. J. van Wijk. Force-Directed Edge Bundling for Graph Visualization. *Computer Graphics Forum (Proc. EuroVis)*, 28(3):983–990, Jan 2009.

[13] D. Holten and J. J. van Wijk. A user study on visualizing directed edges in graphs. In *ACM CHI*, pages 2299–2308, 2009.

[14] M. Kaufmann and D. Wagner, editors. *Drawing Graphs: Methods and Models (Lecture Notes in Computer Science)*. Springer, Berlin, 2001.

[15] A. Lambert, R. Bourqui, and D. Auber. Winding Roads: Routing edges into bundles. *Computer Graphics Forum (Proc. EuroVis)*, 29(3):853–862, Jan 2010.

[16] F. van Ham, H.-J. Schulz, and J. Dimicco. Honeycomb: Visual analysis of large scale social networks. In *INTERACT*, volume 5727 of *Lecture Notes in Computer Science*, pages 429–442. Springer Berlin / Heidelberg, 2009.

[17] C. Ware. *Information Visualization: Perception for Design*. Morgan-Kaufmann, 2004.

[18] N. Wong. Using edge plucking for interactive graph exploration. In *IEEE InfoVis Posters*, 2005.

[19] N. Wong, S. Carpendale, and S. Greenberg. Edgelens: an interactive method for managing edge congestion in graphs. In *Information Visualization, 2003. INFOVIS 2003. IEEE Symposium on*, pages 51 –58, oct. 2003.