Skeleton-Based Edge Bundling for Graph Visualization

Ozan Ersoy, Christophe Hurter, Fernando V. Paulovich, Gabriel Cantareira, and Alexandru Telea

Abstract—In this paper, we present a novel approach for constructing bundled layouts of general graphs. As layout cues for bundles, we use medial axes, or skeletons, of edges which are similar in terms of position information. We combine edge clustering, distance fields, and 2D skeletonization to construct progressively bundled layouts for general graphs by iteratively attracting edges towards the centerlines of level sets of their distance fields. Apart from clustering, our entire pipeline is image-based with an efficient implementation in graphics hardware. Besides speed and implementation simplicity, our method allows explicit control of the emphasis on structure of the bundled layout, *i.e.* the creation of strongly branching (organic-like) or smooth bundles. We demonstrate our method on several large real-world graphs.

Index Terms—Graph layouts, edge bundles, image-based information visualization.

1 Introduction

Graphs are among the most important data structures in information visualization, and are present in many application domains including software comprehension, geovisualization, analysis of traffic networks, and social network exploration. Classical visualization metaphors for general graphs include node-link diagrams [17], matrix plots [34], and graph splatting [35]. For specific types of graphs, such as hierarchies (trees), additional methods exist such as treemaps.

As the number of nodes and edges of a graph increases, node-link graph visualizations become challenged by *clutter*, *i.e.* unorganized groups of nodes and edges onto small screen areas. To reduce clutter, and also address use-cases which focus on simplified depiction of large graphs with an emphasis on graph structure, several methods have emerged. Specifically, *bundling* methods are an interesting alternative for classical node-link metaphors. Bundling typically starts with a given set of node positions, either present in the input data, or computed using a layout algorithm. Edges found to be close in terms of graph structure, geometric position of their endpoints, data attributes, or combinations thereof, are drawn as tightly bundled curves. This trades clutter for overdraw and produces images which are easier to understand and/or better emphasize the graph structure. Edge bundles can be rendered using various effects such as blending or shading [15, 22, 32]. Edge bundling algorithms exist for both compound (hierarchy-and-association) [14] and general graphs [15, 7, 24, 22].

In this paper, we present a novel approach for constructing edge bundles for general graphs. We adapt a recent result which computes centerlines, or skeletons, of groups of edges [32] and use the skeleton for actual edge bundling rather than shading only. In detail, we combine edge clustering, distance fields, and 2D skeletonization to construct bundled layouts by iteratively attracting edges towards the centerlines of level sets of their distance fields. Apart from clustering, our pipeline is image-based, which allows an efficient implementation in graphics hardware. Besides speed, our method allows users to explicitly control the emphasis on bundle structure, *i.e.* create strongly branching (organic-like) or smooth bundles which always have a tree structure. This type of control can be helpful in applications where one is interested to see how several edges 'join' together into, or split from, main structures, for example when exploring the structure of a network. Instances hereof are examining the local hierarchy of traffic connections in a road or airline network, or identifying the number and size of branches (fan in/out patterns) in software structures.

The structure of this paper is as follows. In Section 2, we review related work on edge bundles. Section 3 presents our bundling algorithm. Section 4 details implementation. Section 5 presents applications on large real-world graphs. Section 6 discusses our method. Section 7 concludes the paper and outlines future work directions.

2 Related Work

Related work in reducing clutter in large graph visualizations can be organized as follows.

Graph simplification techniques reduce clutter by simplifying the graph prior to layout *e.g.* by grouping strongly connected nodes and edges into so-called metanodes, followed by using classical node-link layouts for visualization. Several simplification methods exist, *e.g.* [1, 2]. Graph simplification is attractive as it reuses existing node-link layouts out of the box, but can be sensitive to simplification parameters, which further depend on the type of graph being processed. It does not allow a continuous treatment of the graph: the simplification events yield a set of discrete graphs rather than a smooth exploration scale [22]. Also, simplification typically changes node positions (collapse to metanodes), which can be undesirable *e.g.* when positions encode information.

Edge bundling techniques trade clutter for overdraw, by routing geometrically and semantically related edges along similar paths. Further details on clutter causes and reduction strategies in information visualization are given in [11]. Bundling can be seen as condensing the edges' angle distribution along a reduced set of directions and also sharpening the local edge spatial density, by making it high at bundle locations and low elsewhere. This improves readability in terms of finding groups of nodes related to each other by groups of edges (the bundles). Bundling increases the amount of white space between bundles, which makes their visual separation easier.

Dickerson *et al.* merge edges by reducing non-planar graphs to planar ones [9]. Holten pioneered edge bundling under this name for compound (hierarchy-and-association) graphs by routing edges along the hierarchy layout using B-splines [14]. Gansner and Koren bundle edges in a circular node layout similar to [14] using area optimization metrics [13]. Dwyer *et al.* use curved edges in force-directed layouts to minimize crossings, which implicitly creates bundle-like shapes [10]. Force-directed edge bundling (FDEB) creates bundles by attracting control points on edges close to each other [15]. FDEB can be significantly optimized using multilevel clustering techniques such as the MINGLE method [12]. Flow maps produce a binary clustering of nodes in a directed graph representing flows to route curved edges along [24]. Control meshes are used by several authors to route curved edges, *e.g.* [26, 36]; a Delaunay-based extension called geometric-based edge bundling (GBEB) [7]; and 'winding roads' (WR) which use boundaries of Voronoi diagrams for 2D [22] and 3D [21] layouts.

Several techniques exist for rendering bundled layouts, *e.g.* color

• *O. Ersoy and A. Telea are with the University of Groningen, the Netherlands, E.mail: o.ersoy@rug.nl, a.c.telea@rug.nl.*
• *C. Hurter is with DGAC-DTI R& D, ENAC/Univ. Toulouse, France, E.mail: christophe-hurter@aviation-civile.gouv.fr.*
• *F. V. Paulovich and G. Cantareira are with the University of Sao Paulo, Brazil, E-mail: paulovic@icmc.usp.br, cantareira@icmc.usp.br.*

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.

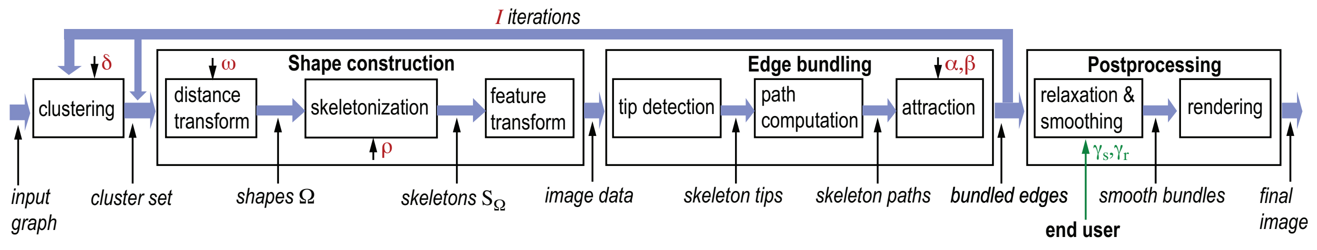

Fig. 1. Skeleton-based edge bundling pipeline. End user parameters are marked in green. System preset parameters are in red.

interpolation along edges for edge directions [14, 7]; transparency or hue for local edge density, *i.e.* the importance of a bundle, or for edge lengths [22]. Whole bundles can be drawn as compact shapes whose structure is emphasized by shaded cushions [32]. Graph splatting visualizes node-link diagrams as continuous scalar fields using color and/or height maps [35, 16].

3 Algorithm

The inspiration behind our method relates to a well-known fact in shape analysis: given a 2D shape, its skeleton is a curve locally centered with respect to the shape's boundary [6]. Skeleton branches capture well the topology of elongated shapes [20, 29]. Hence, if we could create such shapes from sets of edges in a graph, their skeletons could be suitable locations for bundling. To this end, we propose a skeleton-based edge bundling method, as follows (see Fig. 1):

1. we *cluster* edges into groups, or clusters, C_i which have strong geometrical and optionally attribute-based similarity;
2. for each cluster C, we compute a thin shape Ω surrounding its edges using a distance-based method;
3. for each shape Ω, we compute its skeleton S_Ω and feature transform of the skeleton FT_S;
4. for each cluster C, we attract its edges towards S_Ω using FT_S;
5. we repeat the process from step 1 or step 2 until the desired bundling level is reached;
6. we perform a final smoothing and next render the graph using a cushion-like technique to help understanding bundle overlaps.

We start with an unbundled graph $G = (V, E)$ with nodes V and edges E. We assume that we have node positions $v_i \in \mathbf{R}^2$, either from input data, or from laying out G with any existing method *e.g.* spring embedders [17]. Edges $e_i \in E$ are sampled as a set of points connected by linear interpolation; other schemes such as splines work equally well. The start and end points of an edge, denoted e_i^s and e_i^e respectively, are the positions of the nodes the edge connects. Edge points may come from input data, *e.g.* when we bundle a graph which has explicit edge geometry. If no edge positions are available, we initialize the edge points by uniformly sampling the line segments (e_i^s, e_i^e) with some small step. Our bundling algorithm iteratively updates these edge points. Its output is a bundled layout of G which keeps node positions intact and adjusts the edge points to represent bundled edges.

The six steps of our method are explained next.

3.1 Clustering

To obtain elongated 2D shapes, needed for our bundling (described next in Sec. 3.3), we first cluster edges using a similarity metric which groups same-direction, spatially close, edges, using the clustering method described in [32]. We have tested several clustering algorithms: hierarchical bottom-up agglomerative (HBA) clustering using full, centroid, single, and average linkage, and k-means clustering, both with Euclidean and statistical correlation (Pearson, Spearmans rank, Kendalls τ) distances. HBA with full linkage and Euclidean distance given by

$$d(e_i, e_j) = \sqrt{\sum_{k=1}^{N} \|e_{ik} - e_{jk}\|^2} \tag{1}$$

where $e_{ik, k\in\overline{1,N}}$ are uniformly spaced sample points along the edges, with $N \in [50, 100]$, gives the best results, *i.e.* clusters with geometrically close edges which naturally follow the graph structure. Using the same N for all edges removes edge length bias. HBA delivers a dendrogram $D = \{C_i\}$ with the edge set E as leaves and similarity (linkage) values $d(C)$, equal to the full linkage of cluster C based on the distance metric in Eqn. 1, increasing from root to leaves. We select a 'cut' in D, or partition, $P = \{C_i \in D | d(C_i) < \delta\}$ of E based on a similarity value δ, set by our algorithm as explained further in Secs. 3.5 and 4. If desired, d in Eqn. 1 can be easily adapted to incorporate edge data attributes, as outlined in [32].

3.2 Shape construction

Clustering delivers sets of spatially close edges, *i.e.*, the bundling candidates. Given such a cluster $C = \{e_i\}$, we consider its drawing $\Delta(C) \subset \mathbf{R}^2$, *e.g.* the set of polylines corresponding to its edges e_i if we use the default linear edge interpolation. We construct a compact 2D shape $\Omega \subset \mathbf{R}^2$ surrounding $\Delta(C)$, as follows (see also Fig. 2). Given any shape $\Phi \subset \mathbf{R}^2$, we first define its distance transform $DT_\Phi : \mathbf{R}^2 \to \mathbf{R}_+$ as

$$DT_\Phi(\mathbf{x} \in \mathbf{R}^2) = \min_{\mathbf{y}\in\Phi} \|\mathbf{x} - \mathbf{y}\| \tag{2}$$

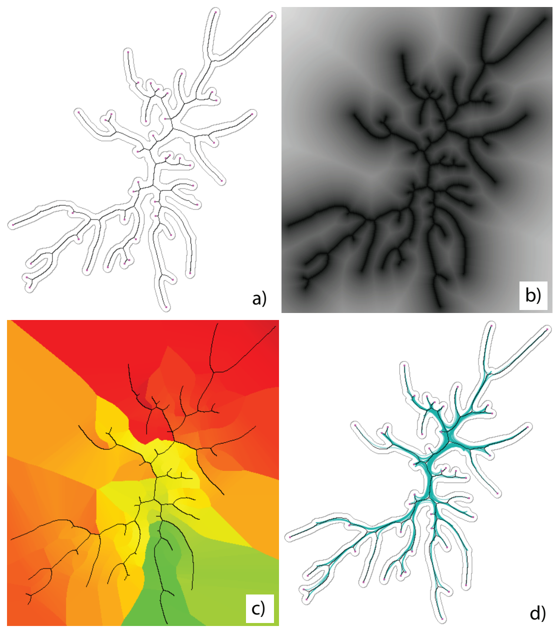

Fig. 2. Shape construction: a) $\partial\Omega$ and S; b) DT_S; c) FT_S; d) bundling result. See Secs. 3.2-3.4 for details.

Given a distance value ω, we next define our shape Ω as

$$\Omega = \{\mathbf{x} \in \mathbf{R}^2 | DT_{\Delta(C)}(\mathbf{x}) \leq \omega\} \tag{3}$$

where $DT_{\Delta(C)}$ is the distance transform of the drawing $\Delta(C)$ of C's edges. The shape's boundary $\partial\Omega$ is the level set of value ω of $DT_{\Delta(C)}$ (see Fig. 2 a). This is equivalent to inflating $\Delta(C)$ with a distance ω in all directions. In practice, we set ω to a small fraction (*e.g.* 0.05) of the bounding box of G. Efficient computation of distance transforms is detailed further in Sec. 4.

3.3 Shape creation

Given a shape Ω computed from an edge cluster drawing as outlined above, we next compute its skeleton S_Ω defined as

$$S_\Omega = \{\mathbf{x} \in \Omega | \exists \mathbf{y}, \mathbf{z} \in \partial\Omega, \mathbf{y} \neq \mathbf{z}, \|\mathbf{x}-\mathbf{y}\| = \|\mathbf{x}-\mathbf{z}\| = DT_{\partial\Omega}(\mathbf{x})\} \tag{4}$$

i.e. the set of points in Ω which admit at least two different so-called feature points on $\partial\Omega$, at distance equal to the distance transform of $\partial\Omega$ (Fig. 2 a).

Given S, we now compute its so-called one-point feature transform $FT_S : \mathbf{R}^2 \to \mathbf{R}^2$, defined as

$$FT_S(\mathbf{x}) = \{\mathbf{y} \in S | DT_S(\mathbf{x}) = \|\mathbf{x}-\mathbf{y}\|\} \tag{5}$$

i.e. one of the feature points of $\mathbf{x}$. Figure 2 b,c show the DT_S and FT_S of a skeleton. Gray values in Fig. 2 b indicate the DT_S value (low=black, high=white). Colors in Fig. 2 c indicate the identity of different feature points: same-color regions correspond roughly to the Voronoi regions of the skeleton branches [33]. The skeleton is the identity set of FT_S, *i.e.* $\forall \mathbf{x} \in S, FT_S(\mathbf{x}) = \mathbf{x}$. Note that, in Eqn. 5, we use the distance transform DT_S of the skeleton S, and not the distance transform $DT_{\partial\Omega}$ of the shape. Also, note that the one-point feature transform is simpler than the so-called full feature transform

$$FT_S^{full}(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{y}\in S} \|\mathbf{x}-\mathbf{y}\| \tag{6}$$

which records *all* feature points of $\mathbf{x}$ [6].

In practice, we compute distance transforms, one-point feature transforms, and skeletons in discrete image (screen) space. This allows efficient implementation (see Sec. 4) and also further processing of the skeleton for edge bundling, as described next.

3.4 Edge attraction

Using the skeleton S and its feature transform FT_S, we now bundle the edges $e_i \in C$ by attracting a discrete representation of each edge towards S. This idea is based on the following observations. First, given the way we combine clustering and edge bundling, a cluster contains only edges having close trajectories; the reasons for this are detailed in Sec. 3.5. By construction, the skeleton S of a cluster is locally centered with respect to the (similar) edges in that cluster, *i.e.* a good candidate for the position to bundle towards. Secondly, $FT_S(\mathbf{x}) - \mathbf{x}$ gives, for each point $\mathbf{x} \in \mathbf{R}^2$, the direction vector from $\mathbf{x}$ to the closest skeleton point to $\mathbf{x}$, *i.e.* the direction to bundle towards. We use these observations to bundle e_i as follows.

First, we compute all branch termination points, or *tips*, $T = \{\mathbf{t}_i\}$ of S. Given that S is represented in image space, we use a simple and efficient 3×3 pixel template-based method [19] to locate t_i. Next, we compute all skeleton paths $\Pi = \{\pi_i \subset S\}$ between any two tips $\mathbf{t}_i$ and $\mathbf{t}_j$. The paths are represented as pixel chains and are found using depth-first search from each $\mathbf{t}_i$ on the skeleton pixel-adjacency-graph. We next use these paths to robustly attract the edges towards the skeleton.

For each $e_i \in C$ with start and end points e_i^s and e_i^e respectively, we select a skeleton path $\pi(e_i) \in \Pi$ so that $\{FT_S(e_i^s), FT_S(e_i^e)\} \subset \pi(e_i)$, *i.e.* a path passing through the feature points of both edge end points. If there are several such paths in Π, we pick any one of them, the particular choice having no influence on the algorithm.

We now use $\pi(e_i)$ to bundle e_i along the skeleton, as follows. Consider a point $\mathbf{x} \in e_i$ located at arc-length distance $\lambda(\mathbf{x})$ from $\mathbf{e}_i^s$. We move $\mathbf{x}$ towards $FT_S(\mathbf{x})$ with a distance which is large if $\mathbf{x}$ is far away from $FT_S(\mathbf{x})$ and/or close to the middle of the edge:

$$\mathbf{x}^{new} = \left[1 - \alpha\phi\left(\frac{\lambda(\mathbf{x})}{\lambda(\mathbf{e}_i^e)}\right)\right]\mathbf{x} + \alpha\phi\left(\frac{\lambda(\mathbf{x})}{\lambda(\mathbf{e}_i^e)}\right)FT_S(\mathbf{x}) \tag{7}$$

Here, $\alpha \in [0,1]$ controls the tightness of bundling: Large values bring the edge closer to the skeleton, whereas small values bundle less. The function $\phi : [0,1] \to [0,1]$ defined as

$$\phi(t) = [2 \min(t, 1-t)]^K \tag{8}$$

modulates the motion amount so that the edge's end points $\mathbf{e}_i^s$ and $\mathbf{e}_i^e$ do not move at all, points close to these end points move less, and points around the middle of the edge move most. This produces the curved edge profile we require for bundling, and also keeps edge end points fixed to their node locations. The parameter K controls how smoothly edges twist, or curve, from their nodes to reach their bundled location. Higher K values produce more twists, and low K values produce smoother twists. Values of $K \in [3,6]$ give very similar results to known bundling methods *e.g.* [14, 15, 22]. Also, for any $\mathbf{x} \in S$, $FT_S(\mathbf{x}) = \mathbf{x}$ (Sec. 3.3), so for such points we have $\mathbf{x}^{new} = \mathbf{x}$ (Eqn. 7), *i.e.* points which have reached the skeleton, the extreme bundling location, do not move any longer.

Equation 7 is equivalent to advecting edge points $\mathbf{x}$ in the gradient field $-\nabla DT_S$. Distance transforms of any shape except a straight line have div $\nabla DT_S \neq 0$ [28]. Hence, our attraction typically shortens and/or lengthens edges, since these get immediately curved after one application of Eqn. 7. We compute the edge points $\mathbf{x}$ used in Eqn. 7 by uniformly sampling edges in arc-length space with a distance equal to a small fixed fraction (0.05) of the layout's bounding box. This removes points where the edge contracts (div $\nabla DT_S < 0$) and inserts points where the edge dilates (div $\nabla DT_S > 0$) as needed, thus ensuring a uniform edge sampling density.

3.4.1 Attraction singularities

As explained, Eqn. 7 is equivalent to advecting $\mathbf{x}$ in the field $-\nabla DT_S$. This field is smooth everywhere in $\mathbf{R}^2$ except on points $\mathbf{x}$ where $\|FT_S^{full}(\mathbf{x})\| > 1$, *i.e.* points located on the skeleton of the skeleton's complement, or Voronoi diagram of S, $\overline{S} = S_{\mathbf{R}^2 \setminus S}$. Intuitively, $\overline{S}$ corresponds in Fig. 2 c to color discontinuities. Although this singularity set is small, *i.e.* a set of curves in 2D, we need special treatment for such situations. If we were to directly advect a curve using Eqn. 7 with no further precaution, singularities would appear where the curve crosses $\overline{S}$, since ∇DT_S has a high absolute divergence, *i.e.* changes direction rapidly, in such areas [28]. Such singularities appear as sharp kinks in the curve, which defeats our purpose of creating smooth bundles. For example, attracting the blue edge e in Fig. 3 a towards the Y-shaped skeleton yields the red line which shows two kinks, where e crosses $\overline{S}$ (dotted line) at points $\mathbf{a}$ and $\mathbf{b}$. The problem is made only more complex by the fact that we use a sampled edge representation, so $\mathbf{x}$ may be close, but not on, $\overline{S}$.

We solve such situations by an implicit *regularization* of the advection field determined by FT_S. First, we enforce the constraint that points $\mathbf{x} \in e$ can only be advected to points on the edge's path $\pi(e)$. This ensures that, during advection, parts of e cannot be attracted towards other skeleton branches than the set of *contiguous* branches which form π. Intuitively, Eqn. 7 should not pull e towards non-connected skeleton branches. We achieve this constraint as follows (see Fig. 3 b). For each $\mathbf{x} \in e$, we evaluate its $FT_S(\mathbf{x})$. If $FT_S(\mathbf{x}) \in \pi(e)$, we attract the 'regular' point $\mathbf{x}$ using Eqn. 7, else we mark $\mathbf{x}$ as special case. Special points along e (yellow in Fig. 3 b) form compact sets σ_i, which are preceded and followed on e by regular points σ_i^{start} and σ_i^{end} respectively, whose feature points belong to $\pi(e)$ by construction. We next map each special point $\mathbf{x}$ to a corresponding point $\mathbf{x}^{map}$ on $\pi(e)$ using arc-length interpolation along both σ_i and their corresponding path fragments $[FT_S(\sigma_i^{start}), FT_S(\sigma_i^{end})] \subset S$ (dark green in

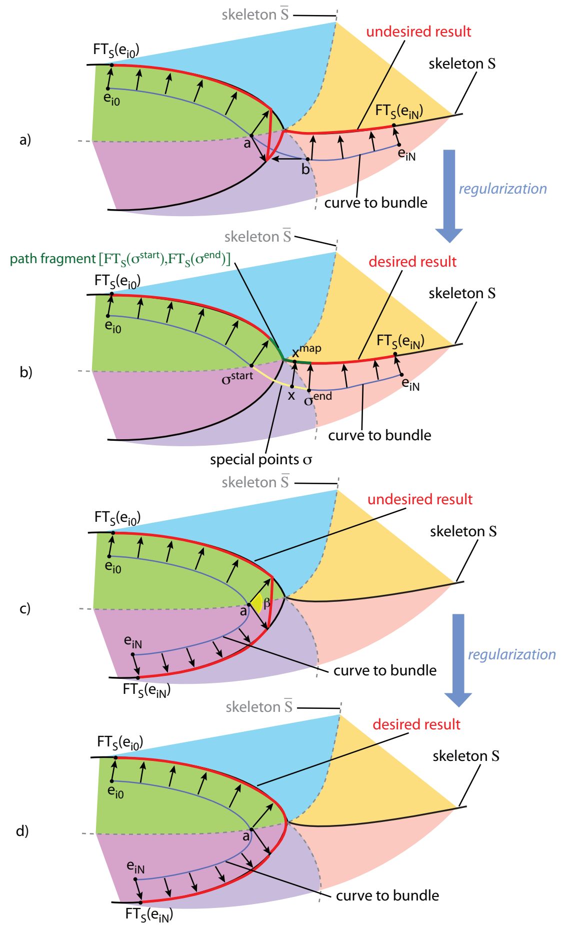

Fig. 3. Attraction singularities. Naive solution (a,c) and corresponding solutions with regularization (b,d). Final bundled curve is shown in red. Voronoi regions of the branches of $\bar{S}$ are shown in different hues.

Fig. 3 b), and use $\mathbf{x}^{map}$ in Eqn. 7 instead of $FT_S(\mathbf{x})$. This ensures that both special and regular points are attracted to the same path $\pi(e)$, and thus, since $\pi(e)$ is a compact curve, that the motion of e is smooth.

However, the above regularization does not eliminate *all* sharp kinks in the advection of an edge: Consecutive points of the edge can 'see' points on the same skeleton path π, and still be separated by a singularity (see point **a** in Fig. 3 c). As explained, advecting such points **a** using Eqn. 7 would produce undesirable bends. Since the feature-point of **a** is located on the same path $\pi(e)$ as those of **a**'s neighbors on the edge, we cannot find **a** using the path-based detection criterion outlined above. We solve this problem by using an angle-based criterion: Given our discrete edge representation $e = \{\mathbf{x}_i\}$, we test if the feature vectors $FT_S(\mathbf{x}_i) - \mathbf{x}_i$ and $FT_S(\mathbf{x}_{i+1}) - \mathbf{x}_{i+1}$ of consecutive edge sample points $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ form a large angle β. If β exceeds a user-defined value β_{max}, we mark $\mathbf{x}_i$ as a special point and treat it as explained earlier for the path-based detection criterion. In practice, $\beta_{max} = \pi/4$ has given good results for all graphs we tested. The overall effect is that sharp edge angles are eliminated and edges are advected smoothly towards the skeleton (Fig. 3 d). As a more complex example of our regularization, Fig. 2 d shows the bundling of a set of edges (green) close to the skeleton in Fig. 2 a.

Our angle criterion is a one-dimensional version of the divergence-based Hamilton-Jacobi skeleton detector of [28]. It subsumes the path-based criterion. In theory, it would be sufficient to use the angle criterion to achieve smooth motion. However, the path-based criterion is more numerically robust as it involves no angle estimation or thresholding. Since its application is equally fast (we need paths anyway to regularize the attraction in both cases), we use it when applicable to reduce any chance for numerical instabilities.

3.5 Iterative algorithm

For a given graph layout, one application of the clustering, shape construction, and edge attraction steps outlined above yields a new layout whose edges are closer to their respective cluster skeletons. To achieve full bundling, we repeat this process iteratively until a user-specified number of iterations I is reached. More iterations yield tighter bundled edges. This process is strictly monotonic, *i.e.* edges can only get closer to their clusters' skeletons (hence to each other) by construction, as explained below (see also Fig. 4).

First, let us explain why clustering needs to be repeated during the iterative process. For the first clustering, we use a high similarity threshold δ in order to guarantee elongated, thin, clusters regardless of the edge spatial distribution in the input graph (Sec. 3.1). This is essential for getting the initial bundling under way. Indeed, if we had weakly coherent clusters, these would contain edges that intersect each other at large angles, hence the shapes surrounding them, and their skeletons, would be meaningless as bundling cues. For subsequent iterations, we decrease δ and recluster the graph each few $(3to5)$ iterations. This produces fewer, increasingly larger, clusters, which allows fine-scale bundles to group into coarse-scale ones. However, these large clusters are *locally* elongated, since they contain already partially bundled edges. Hence, coarsening the clustering will not group unrelated edges. The overall effect is bottom-up bundling: First, the closest edges get bundled, yielding fine-scale local bundles, followed by increasingly coarser-scale bundle merging.

Similarly, we decrease α during the iterative process. Initial large α values yield strongly coherent initial bundles, needed for clustering stability as explained above. Subsequent relaxed α values allow edges in more complex, larger, bundles to adjust themselves. Concrete values for δ and α are given in Sec. 4.2.

3.6 Postprocessing

3.6.1 Relaxation and smoothing

The output of our bundling algorithm has a strong branch-like structure (see *e.g.* Fig. 6 f). This is the inherent effect of using skeletons as bundling cues. Indeed, skeleton branches asymptotically meet at large angles [25]. This visual signature of our bundles may be desirable for use-cases where one is interested to see the branching structure of a graph. However, often the fact that two bundles join at some point in a thicker bundle is irrelevant, and should not be over-emphasized. We offer this possibility by performing a final postprocessing on the bundled layout. Here, two variations are proposed. First, we apply a simple Laplacian smoothing filter along the edges γ_s times, much like [15]. This removes sharp bundle turns, which by construction appear precisely, and only, where skeleton branches meet. Indeed, as known from medial axis theory, a skeleton branch is always a smooth curve; the only curvature discontinuities along a skeleton appear at branch junctions [25]. A second postprocessing we found useful is to interpolate linearly with a value $\gamma_r \in [0,1]$ between the bundled graph and its initial layout. This relaxes the bundling, which is desirable when users want to see the individual edges within a bundle and/or where these come from in the initial layout. The effect is similar to the spline tightness parameter in [14].

Figure 6 a,b show the effect of smoothing on a graph whose nodes use a radial layout. Smoothing (b) removes the strong branching effect visible in (a) at the locations indicated by arrows. The result is very similar to the HEB layout [14]. However, it is important to stress that we obtain our bundling with no graph *hierarchy* information. Figures 6 e,f show the effect of smoothing and relaxation on the well-known US airlines graph, whose bundled layout is shown in Fig. 7 j. Smoothing removes the 'skeleton effect' from the bundles, while relaxation makes these thicker with less effect on their curvature. As such, the two effects serve complementary goals.

3.6.2 Rendering

Finally, we propose a simple but effective rendering technique for easier visual following of the rendered bundles (Fig. 6 c,d). The principle

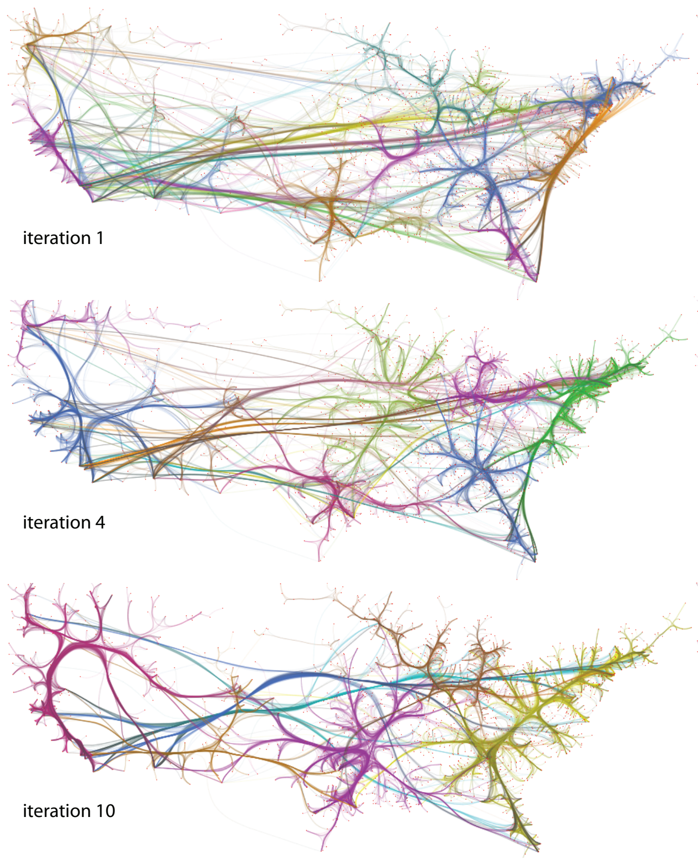

Fig. 4. Iterative bundling of the US migrations graph. Colors indicate edge clusters (see Sec. 3.5).

follows [32]: We render each bundle in back-to-front order, decreasingly sorted by skeleton pixel count $|S|$, as if they were covered by a 3D cushion profile bright at the bundle's center and dark at its periphery. This helps following a given bundle, especially in regions where several bundles cross. In contrast to [32], we use a much simpler technique (see Fig. 5). Edges are rendered as alpha-blended polylines. We modulate the saturation S and brightness B of each polyline point $\mathbf{x}$ based on its distance to the skeleton $d(\mathbf{x}) = DT_S(\mathbf{x})$, which is already computed for the attraction phase (Sec. 3.3). For this, we use

$$S(d) = \sqrt{1 - d/\delta_S} \qquad (9)$$

$$B(d) = 1 - \sqrt{d}/\delta_B \qquad (10)$$

This yields thin, specular-like, white highlights in the middle of the bundles (where the skeleton is located) and darkens the edges as they get further from the skeleton. The parameter δ_B is the local thickness of the bundle. For an edge point $\mathbf{x} \in \Omega$, $\delta_B(\mathbf{x}) = DT_S(FT_{\partial\Omega}(\mathbf{x}))$, *i.e.* the distance of the closest point on the shape boundary $\partial\Omega$ to the shape's skeleton. This does not require any extra computations, since we anyway compute $FT_{\partial\Omega}$ and DT_S as part of the shape construction (Sec. 3.2, see also Sec. 4 for implementation details). The parameter $\delta_S < \delta_B$ controls the highlight thickness and is set to a small fraction (*e.g.* 0.2) of δ_B. This technique has several differences as compared to splatting-based shading techniques for bundles [32, 22]. First, our rendering does not change the screen-space thickness of a bundle, which is determined by the bundling layout – thin bundles stay thin. In contrast, splatting techniques tend to make thin bundles relatively thicker, which consumes screen space and increases occlusion chances. Secondly, if we relax the bundling as described earlier, individual edges become visible but still show up as a coherent whole due to the cushion shading. Figure 6 d shows this. To better illustrate the effect, we decreased here the overall opacity of the edges. The inset shows how bundles appear as shaded profiles even though they are not, technically speaking, compact surfaces. Thirdly, although we could use a physically correct shading model (like [22]), we found our pseudo-illumination adequate in terms of our goal of understanding overlapping bundles.

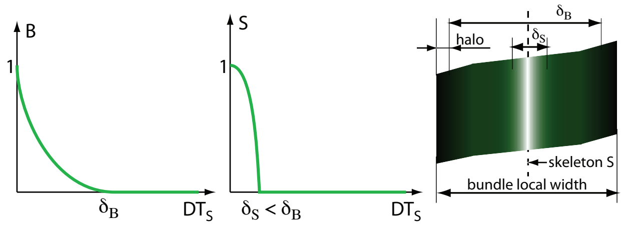

Fig. 5. Cushion shading for bundles (Sec. 3.6.2).

3.6.3 Interaction

We have experimented with several types of interactive exploration atop of our method. In particular, our image-based pipeline and explicit representation of edge clusters allows us to easily brush or select groups of edges showing up as bundles or branches thereof. Three types of selection were found useful, as follows (see also Fig. 8 e-g and example discussed in Sec. 5). Given the mouse position $\mathbf{x}$, we first select all bundled edges within a disc of small radius r centered at $\mathbf{x}$ by computing the feature transform of the *bundled* edges and then selecting all edges which contain feature points in the disc. This query is useful for basic edge brushing and for building the next two queries. Secondly, we want to select all edges in the most prominent bundle, or bundle branch, passing through the disc. We repeat the basic selection, count the number of selected edges having the same cluster id, and retain the ones having the cluster id for which the most edges were found. This selects the thickest bundle branch close to the mouse, since edges within any bundle branch always have the same cluster ids,

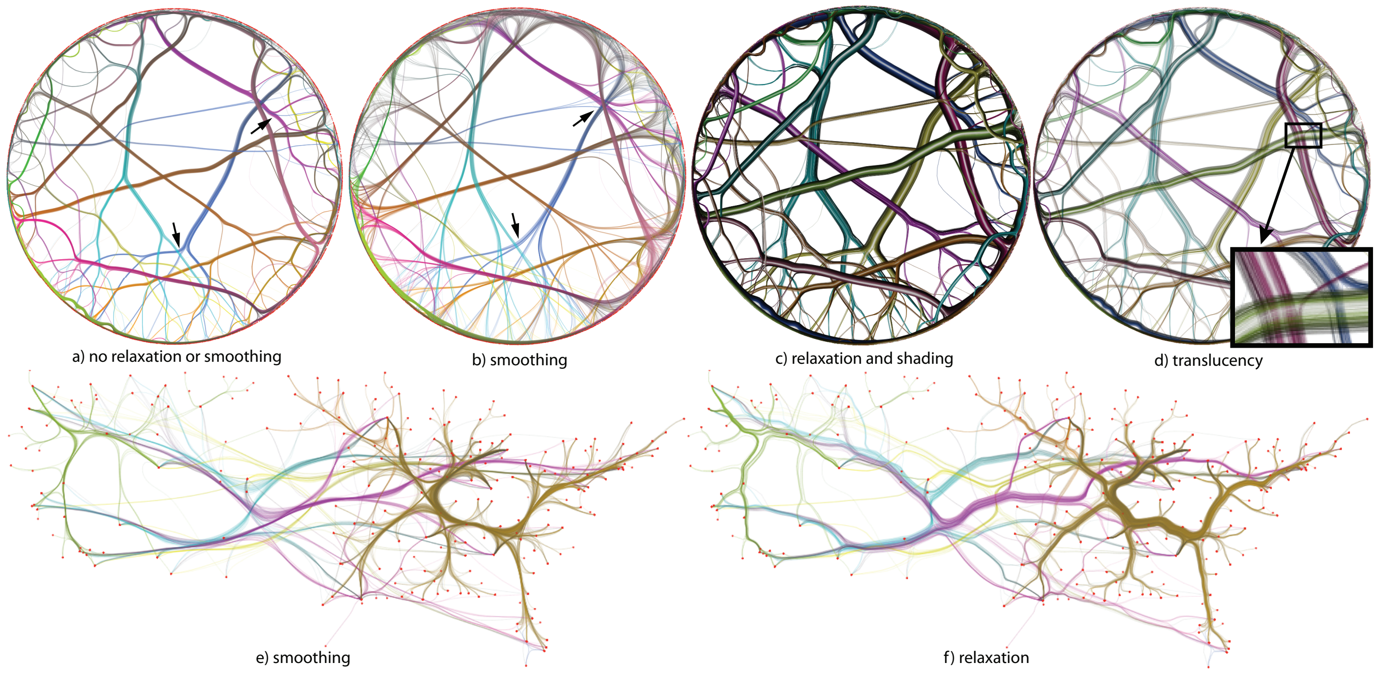

Fig. 6. Layout postprocessing. Edge smoothing (a vs b, Fig. 7 j vs e). Edge relaxation (Fig. 7 j vs f). Cushion shading (c), see-through detail (d).

by construction. Finally, to select an entire cluster, we do the basic selection and return all edges in the cluster whose id is the one for which the most edges were found.

4 IMPLEMENTATION

Several implementation details are crucial to the efficiency and robustness of our method, as follows.

4.1 Image-based operations

We compute shapes, skeletons, skeleton tips, and distance and feature transforms in an image-based setting. First, we render all edges using standard OpenGL polylines. Next, we use a Nvidia CUDA 1.1 based implementation of exact Euclidean distance-and-feature transforms [4]. We extended this technique to compute robust skeletons based on the augmented fast marching method (AFMM) in [33]. In brief, we arc-length parameterize the shape boundary $\partial\Omega$ and detect S_Ω as pixels whose neighbors' feature points subtend an arc on $\partial\Omega$ larger than a given value ρ. The value ρ indicates the minimal detail size on $\partial\Omega$ which creates a skeleton point. Since $\partial\Omega$ is a level-set of a distance transform at value ω of a set of smooth curves (edges), it only contains 'sharp' details at the curve end points. Hence, setting $\rho = \pi\omega$, *i.e.* half the perimeter of a circle of radius ω, guarantees that skeleton tips correspond to edge end points. The skeletonization method choice is essential: the AFMM guarantees that no spurious branches appear due to boundary perturbations, which in turn guarantees stable bundling cues. However, even if all skeleton *tips* correspond to edge end points, this does not mean that all edge *end points* correspond to skeleton tips. Short edges within a large cluster do not produce skeleton tips. This is another reason for using the displacement function ϕ (Eqn. 8) to guarantee that no edge end points move during bundling.

Graph	Nodes	Edges	Clusters/iteration			Total (GPU)
			$I=1$	$I=5$	$I=10$	(sec.)
US airlines	235	2099	90	15	9	6.3
US migrations	1715	9780	57	14	7	4.1
Radial	1024	4021	94	30	24	7.4
France air	34550	17275	207	40	26	29.2
Poker	859	2127	86	28	23	5.2

Table 1. Graph statistics for datasets used in this paper.

Graph ($I=5$)	Tips	Points	Inflation (ms)	Holes (ms)	Skel. (ms)	Paths (ms.)	Attraction (ms)
US airlines	22	8388	77	120	314	98	20
US migrations	28	9780	78	134	339	170	77
Radial	14	21580	80	96	357	45	17
France air	34	23759	81	148	374	222	88
Poker	28	2385	64	117	238	146	13
CUDA implem.			2	8	2	< 12	3

Table 2. SBEB performance. Figures are averages for all clusters at iteration $I=5$ for different graphs. First rows show CPU timings. Last row shows CUDA-based timings (which are uniform for the tested graphs).

The original CPU-based AFMM [33] is too slow for our task. Table 2 show the inflation (Eqn. 2) and skeletonization times (Eqn. 4), the latter also including the skeleton feature transform, on a 2.8 GHz quad-core Windows PC (Sec. 5) for several graphs at an image size of 1024^2. Table 1 gives statistics on these graphs, including the (decreasing) number of clusters at several iterations. On the average, the time needed by the AFMM to process a cluster sums up to 0.4 seconds (in line with [33]). For a graph with 200 clusters (Fig. 7 a-b), this yields 80 seconds/iteration. The AFMM is $O(\delta|C|\ log(\delta|C|))$ where $|C|$ is the number of pixels on all edges in a cluster C, since the AFMM computes within a band of thickness δ around its input shape, *i.e.* $|\Omega| = O(\delta|C|)$. In contrast, our CUDA implementation takes 4 milliseconds per distance, feature transform, and skeletonization for the same image on a Nvidia GT 330M GT card, in line with performance reported in [4], *i.e.* 0.8 seconds per iteration for the graph in Fig. 7 a-b. Graphs with fewer clusters require proportionally less time, since the speed of the CUDA method is $O(N)$ for an image of N pixels, thus image-size-bounded. Overall, the CUDA solution is roughly 100 times faster than the CPU-based AFMM.

The complexity of the skeleton path computations (Sec. 3.4) is discussed next. Following earlier comments on the distance-level-set nature of $\partial\Omega$, the number of skeleton tips $|T|$ for a shape is $O(|\partial\Omega|/(\pi\omega))$. Since we set ω to a fixed fraction of the image size (0.05, see Sec. 3.2), we get on the average a few tens of tips per skeleton, regardless of the number of edges in a cluster (Tab. 2 (Tips)). AFMM guarantees 1-pixel-thin skeletons [33], so all nodes in the skeleton pixel-adjacency-graph are of degree 2, except skeleton junctions which are $O(|T|)$ in number. The length of the skeleton of a shape $\partial\Omega$ is $O(|\partial\Omega|)$. Hence, the depth-first-search finding of skeleton paths between tips (Sec. 3.4) is $O(|T|^2|\partial\Omega|)$ using a brute-force method. Table 2 (Paths) shows the costs for the graphs in this paper using quad-core multithreading with one depth-first-search per thread. The same implementation on CUDA reduces the costs to 12 milliseconds (or less for skeletons with fewer tips) as more cores are available. This cost could be reduced further, if desired, by using the same depth-first search on the much simpler graph whose nodes are skeleton tips and skeleton branch junctions and edge weights given by skeleton branch lengths, or faster all-pairs shortest path algorithms at the expense of a more complex implementation [18].

The attraction step is linear in the number of edge discretization points, *i.e.* tens of thousands for large graphs (Tab. 2 (Points)). Edges are attracted independently to their cluster skeleton, so CUDA parallelization of this step is immediate.

Inflating edges can produce shapes of genus > 0, *i.e.* with holes. Technically, this is not a problem, as skeletonization, path computation, and attraction can handle this. However, we noticed that such holes are rarely meaningful. Holes create loops in the skeleton and thus loops in a *single* bundle, which is supposed to be a tight object. To remove this, we fill all holes in our shapes prior to skeletonization using an efficient CUDA-based scan fill method, as follows: Given a background seed pixel outside the image Ω, *e.g.* the pixel $(0,0)$, we mark it with a special value v. Next, we fill horizontal scan line segments of background value from each v-valued pixel in parallel, one scan line per thread. We repeat alternating horizontal with vertical scan line passes until no pixel is filled any more. Checking the stop condition requires only non-synchronized writing to a global boolean variable, set to false before each pass. This parallelizes more efficiently than classical scan line or flood fill. Marking all non-v pixels as foreground fills all holes in Ω. The entire fill takes under 20 scan iterations for all images we examined. CUDA filling adds around 8 milliseconds/image of 1024^2 pixels in comparison with around 0.15 seconds/image for classical CPU flood fill (Tab. 2 (Holes)) up to a total of roughly 25 milliseconds per cluster per iteration. Note that, due to filling, all skeletons, and thus the created bundles, become trees rather than graphs. Although we do not use this property now, it may enable future interaction work such as user manipulation of the layout by means of bundle handles.

Clustering using HBA is fast. The CPU implementation in [8] constructs the complete dendrogram of a graph of 10K edges in 0.1 seconds on our considered machine. We next added the GPU-based clustering in [5], which is roughly 10 to 15 times faster. Note that only a few clustering passes are needed for a complete layout (Sec. 3.5). Also, we do not need to construct the entire dendrogram, but only the bottom-most part thereof, until we reach the cut value δ (Sec. 3.1) at which we extract the clusters to bundle further.

Finally, postprocessing (Sec. 3.6) poses no performance problems, so we implement it in real-time using standard OpenGL polyline rendering and CPU-based smoothing and relaxation. All in all, the CUDA-based bundling takes 5 to 30 seconds for producing a final layout for the graphs we tested (Tab. 1, right column), *i.e.* 25 milliseconds per cluster times the total number of clusters processed during the $I = 10$ iterations plus the clustering time. In terms of memory, our method is scalable: we only need a few 1024^2 images (distance and feature transforms and skeletons) and discard these once a cluster is processed; all paths between skeleton tips for the current cluster; and the graph edge polylines. For all graphs presented here, this amounts to under 100 MB total application memory requirements per graph.

4.2 Parameter setting

Our entire method has a few parameters: the clustering similarity threshold δ, edge advection factor α, total number of iterations I, and smoothing and relaxation amounts γ_s and γ_r. These parameters allow covering a number of different scenarios, as follows.

Clustering similarity threshold δ: This parameter specifies the granularity level at which we cut the cluster dendrogram to obtain sets of edges to bundle at the current iteration (Sec. 3.1). We set δ as a linearly decreasing function on the iteration number $t \in [1, I]$ from $\delta(1) = 0.95$ to $\delta(I) = 0.7$. This yields strongly coherent clusters in the first iteration, regardless of the initial edge position distribution, and also *locally* strongly coherent clusters in the subsequent iterations (Sec. 3.5).

Edge advection factor α: The advection value $\alpha \in (0,1)$ controls how much edges approach the skeleton at one iteration. This implicitly controls the bundling convergence speed. Too high values yield tight bundles and convergence after the first few iterations, which is fine for graphs which already have relatively grouped edges, but limits the freedom in decluttering complex graphs. Too low values allow the iterative process to adapt itself better to newly discovered clusters as the edges approach each other, but convergence requires more iterations. In practice, we set α as a linearly decreasing function of the iteration number from $\alpha(0) = 0.9$ to $\alpha(I) = 0.2$.

Number of iterations: In practice, after $I \in [10, 15]$ iterations, we obtain tight bundles of a few pixels in width for all graphs we worked with. This is expectable, given that $(1-\alpha)^I$ becomes very small for $\alpha < 1, I > 10$. In practice, we always set $I = 10$ and then use smoothing and relaxation to interactively adjust the result as desired.

Smoothing: The smoothing amount $\gamma_s \in \mathbf{N}$ describes the number of Laplacian smoothing steps executed on the bundled layout (Sec. 3.6). Values $\gamma_s \in [3, 10]$ give an optimal amount of smoothing which keeps the structured aspect of the layout but eliminates the skeleton-like look. Larger values make our layout look similar to the force-directed method of [15]. In practice, we noticed that the smoothing amount strongly depends on the task at hand: In some cases, users attach semantics to the branching structure, *i.e.* want to clearly see which groups of edges get merged together, so no smoothing is needed. In the general case, however, the exact bundle merging events are not relevant, so we use by default $\gamma_s = 5$.

Relaxation: The relaxation amount $\gamma_r \in [0,1]$ controls the interpolation between the fully bundled layout and original one (Sec. 3.6). Relaxation is most conveniently applied interactively, after a bundled layout has been computed. Values $\gamma_r \in [0, 0.2]$ give a good trade-off between bundling and overdraw.

Overall, the entire method is not sensitive to precise parameter settings. For the graphs in this paper and other ones we investigated, we have obtained largely identical bundled layouts with different parameter settings in the ranges indicated above. We explain this by the stability of the inflated shape skeletons to small local variations of the positions of edges, and the smoothing effect of the entire iterative process on the layout. As such, the only two parameters we expose to users are γ_s and γ_r, the others being set to predefined values as explained above.

5 Applications

We now demonstrate our skeleton-based edge bundling (SBEB) method for several large, real-world, graphs. Statistics on these graphs are shown in Tab. 1.

Figure 7 illustrates the SBEB and compares it with several existing bundling methods. Note that in all images here generated with our method, we used simple additive edge blending only, as our focus here is the layout, not the rendering. Images (a,b) show an air traffic graph (nodes are city locations, edges are interconnecting flights). Images (c,d) show a graph of poker players from a social network. Edges indicate pairs of players that played against each other. The node layout is done with the spring embedder provided by the Tulip framework [3]. Given the average node degree and node layout algorithm used, related nodes tend to form relatively equal-size cliques. Bundling further simplifies this structure; here, bundles can be used to find sets of players which played against each other.

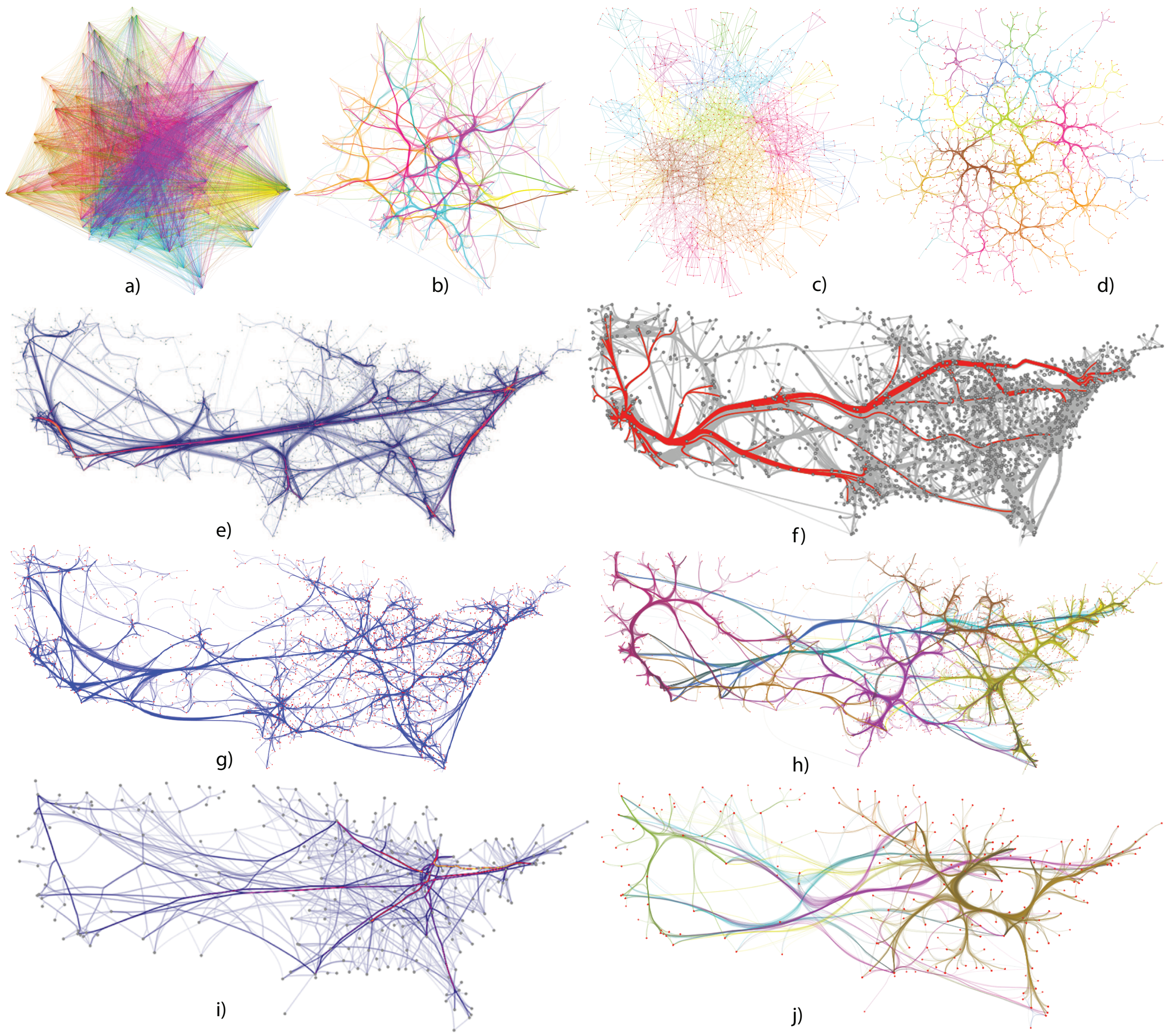

Fig. 7. Air traffic graph (a: original, b: bundled). Poker graph (c: original, d: bundled). US migrations graph (e: FDEB, f: GBEB, g: WR, h: SBEB). US airlines graph (i: FDEB, j: SBEB). Colors in (a-d,h,j) indicate clusters (displayed for method illustration only).

Images (e-h) show the US migrations graph bundled with the WR, GBEB, FDEB, and our method (SBEB) respectively. Overall, SBEB produces stronger bundling, due to the many iterations $I = 10$ being used), and emphasizes the structure of connections between groups of close cities (due to the skeleton layout cues). If less bundling is desired, fewer iterations can be used (Fig. 4). Adjusting the postprocessing smoothing and relaxation parameters, SBEB can create bundling styles similar to either GBEB (higher bundle curvatures, more emphasis on the graph structure) or FDEB (smoother bundles). Finally, images (i,j) show the US airlines graph bundled with the FDEB and SBEB respectively. SBEB generates stronger bundling (more overdraw) but arguably less clutter. Note also that SBEB generates tree-like bundle structures which is useful when the exploration task at hand has an inherent (local) hierarchical nature, *e.g.* see how traffic connections merge into and/or split from main traffic routes.

Figure 8 shows further examples. The images (a,b) show flight paths within France, as recorded by the air traffic authorities [16]. Edge endpoints indicate start and end locations of flight records. The original edges are not straight lines, but actual flight paths (polylines). Note that this dataset is not a graph in the strict sense, since only very few edge endpoints are exactly identical within the dataset. This has to do with the fact that flight monitoring systems record flights (trails). However, edge endpoints are spatially grouped since flights typically start and end in geographically concentrated locations such as airports. Given this, our method is able to create a bundled layout of this dataset with the same ease as for actual graphs. Bundling puts close flight paths naturally into the same cluster. The bundled version emphasizes the connection pattern between concentrated take-off and landing locations, which are naturally the airports. The zoom-in details (Fig. 8 c,d) show the organic effect achieved by bundling.

Figure 8 e-g show a citations graph (433 nodes, 1446 edges). Nodes are InfoVis papers, laid out according to content similarity: close nodes indicate papers within the same, or strongly related, topics. The layout algorithm used for the nodes is multidimensional scaling with least-square projection [23]. Paper similarity is measured using cosine-based distance between term feature vectors [27]. Topics were added as annotations to the image to help explanation. Bundling exposes a structure of the citations between topics. We use the bundle-based selection (Sec. 3.6.3) to highlight one of the bundles, which becomes now dark blue (Fig. 8 f). It appears that this bundle connects papers related to the Graph drawing and Treemap topics. The direction of edges is indicated by node label colors: citing papers are green, cited papers are blue. Green and blue labels are mixed within this bundle, which is expected, since papers in these two topics typically cross-reference each other. Figure 8 g shows a selection of all edges which end at nodes within the ball centered at the mouse cursor. Concretely, we highlighted here all papers citing papers in the Graph drawing topic. Note that this selection is a purely node-based one, *i.e.* it does not use bundles for choosing the edges. However, bundles have now another use: they allow *highlighting* specific edges

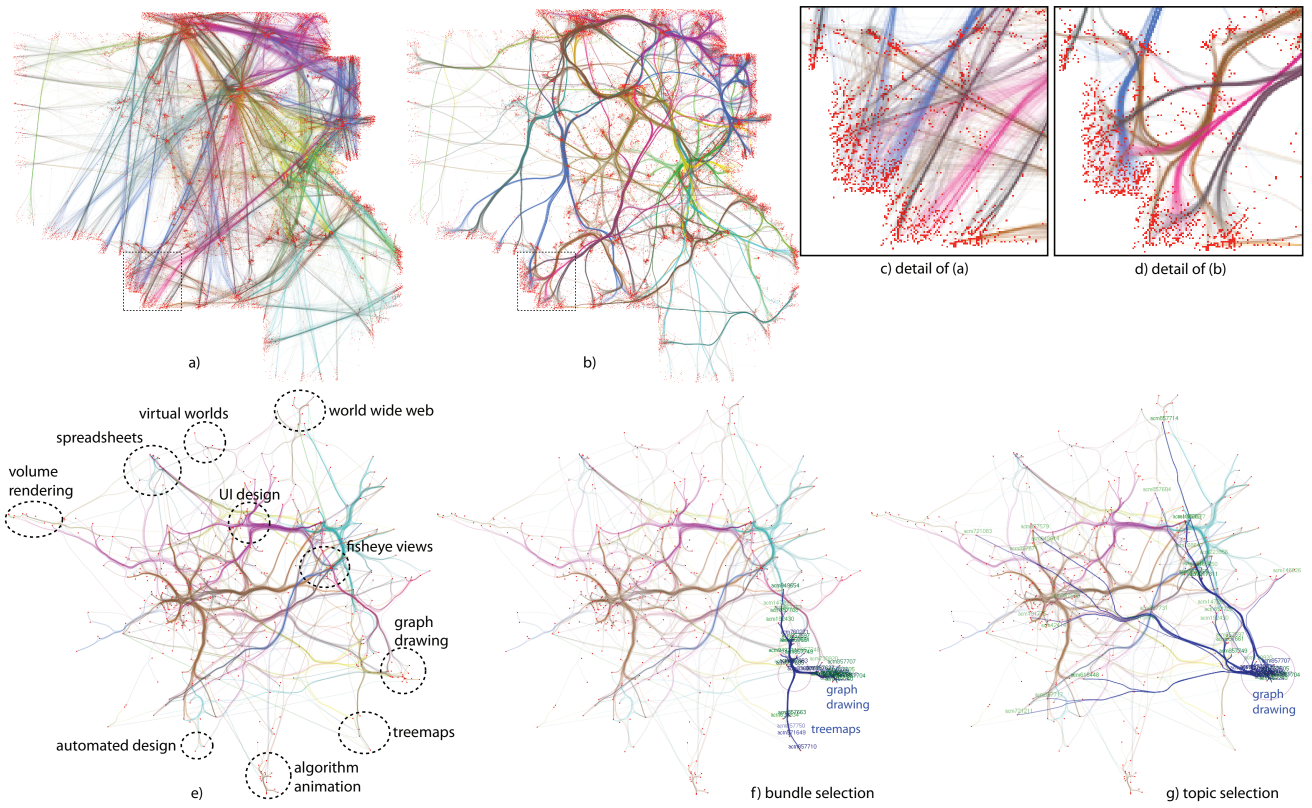

Fig. 8. Bundling of airline trails (a,b) and details (c,d). Bundling of citations graph (e). Selected bundle (in dark blue) shows citations involving two topics (f). Citations to a selected topic (g). In (f,g), node labels indicate edge direction (citing papers=green,cited papers=blue).

in the graph without increasing clutter, since these edges follow the already computed bundles. Also, note that for this type of node layout, our clustering-based bundling makes sense: edges will be grouped in the same bundle if they have similar positions, meaning start/end from similar topics; if the node layout effectively groups nodes into related topics, then bundles have a good chance to show inter-topic relations in a simplified manner.

6 Discussion

In comparison to existing bundling techniques, our method has the following advantages and limitations:

Generality: Our method can treat directed or undirected graphs. By default, we assume the graph is directed, so edges running between the same sets of nodes in opposite directions will belong to different clusters, hence create different bundles. For undirected graphs, we only need to symmetrize the edge similarity function (Eqn. 1).

Structured look control: Users can control the 'structured look' of a bundled layout, ranging between smoothly merging bundles and bundles meeting at sharp angles, by manipulating a single parameter (smoothing γ_S, Sec. 3.6). This implicitly allows removing sharp ramifications when these are meaningless. Other methods, with the exception of HEB, do not allow explicit control of this aspect, since there is no explicit hierarchy aspect in the bundles. In our case, hierarchy is modeled by the cluster skeletons (at fine level) and by the progressively simplified cluster structures (at coarse level).

Robustness: Our method operates robustly on all graphs we experimented on, *i.e.* yields a set of stable skeletons and bundles progressively converging towards an equilibrium state. This is explained by the regularization of the feature transform (Sec. 3.4) and the inherent robustness of the skeletonization method used (Sec. 3.3). Briefly put, adding or removing a small number of nodes or edges will not change the bundling since the distance-based shapes are robust to small changes in the input graph and so are their skeletons too.

Speed and simplicity: Due to the CUDA implementation of its core image-based operations, our method is considerably faster than [15] and slightly faster than [22]. However, we should note that it is not clear if the timings reported in [22] include also the cost of computing the Voronoi diagram underlying the grid graph. The only faster bundled method we are aware of is the MINGLE method [12], which takes 1 second for the US migrations graph and 0.1 seconds for the US airlines graph, in contrast to our 4.1 seconds and 6.3 seconds respectively. MINGLE and SBEB share some resemblance in bottom-up aggregation of edges, but also have some differences. MINGLE compares edges essentially based on end point positions, whereas we use the entire edge trajectory (which may allow us to bundle graphs with curved edges better). The complexity of MINGLE is $O(|E|log|E|)$ for a graph with E edges, whereas SBEB is essentially $O(|C|)$ where C is the average cluster size. By using a better cluster selection than our current iso-linkage cut in the cluster tree (Sec. 3.1), it is possible to reduce $|C|$ and thus make SBEB faster.

Apart from this, our method works entirely image-based, rather than manipulating a combination of hierarchical mesh-based and image-based data structures. The CUDA-based image processing code used by our method is available at [31].

Apart from the above, there are several other differences between our method and recent edge bundling techniques. In contrast to force-directed bundling [15] which bundles pairs of edges iteratively, in a point-by-point manner, we bundle increasingly larger groups of edges (our clusters) along their common center in one single step, using skeletons. In the limit, our method can behave like the force-directed bundling, *i.e.* if we were to treat, at each iteration, only the most cohesive leaf cluster. However, this is practically not interesting, as it would artificially increase the computational cost without any foreseeable benefits. Further, while Lambert *et al.* [22] use shortest paths in a node-based grid graph to route edges, in our method edges bundle themselves using only edge information. As such, there is no relation between the Voronoi diagrams used in [22] and our skeletons (which, formally, can be seen as a Voronoi diagram in which inflated edges are the sites). Distance fields and skeletons are also used in [32], but in different ways; first, an edge distance field is computed using a considerably less accurate quad-splat-based method, whereas our distance transform is pixel-accurate. Secondly, skeletons are used as *shading* cues and not for layout, whereas we use skeletons to actually compute edge layouts. In comparison to [24], where bundles split in exactly two sub-bundles, our bundle splits can have in general any degree, as implied by the underlying skeletons. Also, our method can handle general graphs.

Limitations: There is no fundamental reason why a skeleton-based layout should be preferable to other bundling heuristics, apart from the intuition that a skeleton represents the local center of a shape. Hence, the quality of our layouts (or any other bundled layout) is still to be judged subjectively. Moreover, any bundling inherently destroys information: edges are overdrawn, so cannot be identified separately; and edge directions are distorted. Hence, bundling should be used for those applications where one is interested in coarse-scale connectivity patterns *and* when one cannot apply explicit graph simplification *e.g.* due to the lack of suitable node clustering guidelines and metrics. If desired, SBEB can be modified to incorporate additional bundling constraints *e.g.* maximal deformation of certain edges - the skeletons provide only bundling *cues* but the attraction phase can decide whether, and how much, to bundle any given edge. In the longer run, it is interesting to use shape perception results from computer vision [6, 20] to quantitatively reason about the quality of a bundled layout. Here, our image-based approach may prove more amenable to quantitative analysis than other bundling heuristics which are harder to describe in terms of operators having well-known perceptual properties. However, this is a challenging task and requires further in-depth study.

Acknowledgements

The work of F. V. Paulovich and G. Cantareira has been supported by FAPESP-Brazil. We also thank D. Holten for insightful discussions.

7 Conclusion

We have presented a new method for creating bundled layouts of general graphs. Using the property of 2D skeletons of being locally centered in a shape, we create elongated shapes from a graph with given node positions, and use skeletons as guidelines to bundle similar edges. To guarantee the stability and smoothness of the result, we regularize the feature transforms of 2D skeletons. Our layout amounts to a sequence of edge clustering and image processing operations. Using a CUDA-based implementation we achieve comparable or higher performance than existing comparable methods, and keep implementation simple. Finally, we emphasize edge bundles using shaded cushion techniques computed directly on the bundled edges.

We plan to exploit skeleton properties to generate bundling variations. Modifying the Euclidean distance metric would yield layouts similar to cartographic diagrams [30]. We plan to use bundle-bundle and bundle-node distance fields to globally optimize the layout for maximal readability and incorporate spatial constraints like labels, bundle crossing minimization, and node-edge overlap reduction. In the long run, we plan to study the optimality criteria of bundled layouts by using existing results from shape perception in computer vision which are directly applicable to our skeleton-based layout method.

References

[1] J. Abello, F. van Ham, and N. Krishnan. AskGraphView: A large graph visualisation system. *IEEE TVCG*, 12(5):669–676, 2006.

[2] D. Archambault, T. Munzner, and D. Auber. Grouse: Feature-based and steerable graph hierarchy exploration. In *Proc. EuroVis*, pages 67–74, 2007.

[3] D. Auber. Tulip visualization framework, 2011. `tulip.labri.fr`.

[4] T. Cao, K. Tang, A. Mohamed, and T. Tan. Parallel banding algorithm to compute exact distance transform with the GPU. In *Proc. ACM SIGGRAPH Symp. on Interactive 3D Graphics and Games*, pages 134–141, 2010.

[5] D. Chang, M. Kantardzic, and M. Ouyang. Hierarchical clustering with cuda/gpu. In *Proc. ISCA*, pages 130–135, 2009.

[6] L. Costa and R. Cesar. *Shape analysis and classification: Theory and practice*. CRC Press, 2000.

[7] W. Cui, H. Zhou, H. Qu, P. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE TVCG*, 14(6):1277–1284, 2008.

[8] M. de Hoon, S. Imoto, J. Nolan, and S. Myiano. Open source clustering software. *Bioinformatics*, 20(9):1453–1454, 2004.

[9] M. Dickerson, D. Eppstein, M. Goodrich, and J. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. In *Proc. Graph Drawing*, pages 1–12, 2003.

[10] T. Dwyer, K. Marriott, and M. Wybrow. Integrating edge routing into forcedirected layout. In *Proc. Graph Drawing*, pages 8–19, 2007.

[11] G. Ellis and A. Dix. A taxonomy of clutter reduction for information visualisation. *IEEE TVCG*, 13(6):1216–1223, 2007.

[12] E. Gansner, Y. Hu, S. North, and C. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *Proc. PacificVis*, pages 187–194, 2010.

[13] E. Gansner and Y. Koren. Improved circular layouts. In *Proc. Graph Drawing*, pages 386–398, 2006.

[14] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE TVCG*, 12(5):741–748, 2006.

[15] D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. *Comp. Graph. Forum*, 28(3):670–677, 2009.

[16] C. Hurter, B. Tissoires, and S. Conversy. FromDaDy: Spreading data across views to support iterative exploration of aircraft trajectories. *IEEE TVCG*, 15(6):1017–1024, 2009.

[17] I.Tollis, G. D. Battista, P. Eades, and R. Tamassia. *Graph drawing: Algorithms for the visualization of graphs*. Prentice Hall, 1999.

[18] G. Katz and J. Kider. All-pairs shortest-paths for large graphs on the GPU. In *Proc. Graphics Hardware*, pages 208–216, 2008.

[19] R. Klette and A. Rosenfeld. *Digital geometry: Geometric methods for digital picture analysis*. Morgan Kaufmann, 2004.

[20] I. Kovacs, A. Feher, and B. Julesz. Medial-point description of shape: A representation for action coding and its phychophysical correlates. *Vision research*, 38:2323–2333, 1998.

[21] A. Lambert, R. Bourqui, and D. Auber. 3D edge bundling for geographical data visualization. In *Proc. Information Visualisation*, pages 329–335, 2010.

[22] A. Lambert, R. Bourqui, and D. Auber. Winding roads: Routing edges into bundles. *Comp. Graph. Forum*, 29(3):432–439, 2010.

[23] F. Paulovich, L. Nonato, R. Minghim, and H. Levkowitz. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE TVCG*, 14(3):564–575, 2008.

[24] D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow map layout. In *Proc. InfoVis*, pages 219–224, 2005.

[25] S. Pizer, K. Siddiqi, G. Szekely, J. Damon, and S. Zucker. Multiscale medial loci and their properties. *IJCV*, 55(2-3):155–179, 2003.

[26] H. Qu, H. Zhou, and Y. Wu. Controllable and progressive edge clustering for large networks. In *Proc. Graph Drawing*, pages 399–404, 2006.

[27] G. Salton. Developments in automatic text retrieval. *Science*, 253:974–980, 1991.

[28] K. Siddiqi, S. Bouix, A. Tannenbaum, and S. Zucker. Hamilton-Jacobi skeletons. *IJCV*, 48(3):215–231, 2002.

[29] K. Siddiqi and S. Pizer. *Medial Representations: Mathematics, Algorithms and Applications*. Springer, 1999.

[30] R. Strzodka and A. Telea. Generalized distance transforms and skeletons in graphics hardware. In *Proc. VisSym*, pages 221–230, 2004.

[31] A. Telea. CUDA skeletonization and image processing toolkit, 2011. `www.cs.rug.nl/~alext/CUDASKEL`.

[32] A. Telea and O. Ersoy. Image-based edge bundles: Simplified visualization of large graphs. *Comp. Graph. Forum*, 29(3):543–551, 2010.

[33] A. Telea and J. J. van Wijk. An augmented fast marching method for computing skeletons and centerlines. In *Proc. VisSym*, pages 251–259, 2002.

[34] F. vam Ham. Using multilevel call matrices in large software projects. In *Proc. InfoVis*, pages 227–232, 2003.

[35] R. van Liere and W. de Leeuw. GraphSplatting: Visualizing graphs as continuous fields. *IEEE TVCG*, 9(2):206–212, 2003.

[36] H. Zhou, X. Yuan, W. Cui, H. Qu, and B. Chen. Energy-based hierarchical edge clustering of graphs. In *Proc. PacificVis*, pages 55–62, 2008.