EuroVis 2017
M. Meyer, S. Takahashi, A. Vilanova
(Guest Editors)

Volume 36 (2017), Number 3
STAR – State of The Art Report

State of the Art in Edge and Trail Bundling Techniques

A. Lhuillier[1], C. Hurter[1] and A. Telea[2]

[1]ENAC, Toulouse, France
[2]Institute Johann Bernoulli, Univ. of Groningen, the Netherlands

Abstract

Bundling techniques provide a visual simplification of a graph drawing or trail set, by spatially grouping similar graph edges or trails. This way, the structure of the visualization becomes simpler and thereby easier to comprehend in terms of assessing relations that are encoded by such paths, such as finding groups of strongly interrelated nodes in a graph, finding connections between spatial regions on a map linked by a number of vehicle trails, or discerning the motion structure of a set of objects by analyzing their paths. In this state of the art report, we aim to improve the understanding of graph and trail bundling via the following main contributions. First, we propose a data-based taxonomy that organizes bundling methods on the type of data they work on (graphs vs *trails, which we refer to as paths). Based on a formal definition of path bundling, we propose a generic framework that describes the typical steps of all bundling algorithms in terms of high-level operations and show how existing method classes implement these steps. Next, we propose a description of tasks that bundling aims to address. Finally, we provide a wide set of example applications of bundling techniques and relate these to the above-mentioned taxonomies. Through these contributions, we aim to help both researchers and users to understand the bundling landscape as well as its technicalities.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Computing Methodologies—Picture/Image Generation; I.3.6 [Computer Graphics]: Computing Methodologies—Methodology and Techniques

1. Introduction

Bundling techniques provide a visual simplification of a graph drawing or a set of trails, by spatially grouping graph edges or trails (which we next globally call *paths*). This way, the structure of the visualization becomes simpler and thereby easier to comprehend in terms of assessing *relations* that are encoded by such paths, such as finding groups of strongly interrelated nodes in a graph drawing, finding connections between spatial regions on a map linked by a number of vehicle trails, or discerning the motion structure of a set of objects by analyzing their trajectories. Bundling techniques started with graph simplification by edge concentration [New89] in 1989 and extensions of Sankey graphs [Tuf92]. Graph drawing simplification has since then been a major focus of edge bundling, a term introduced by Dickerson *et al.* for the reduction of clutter in a graph drawing via node placement [DEGM03]. The method was next extended to handle hierarchical graphs drawn in 2D [Hol06] and 3D [CC07, GBE08]; general graphs [HVW09, DMW07]; spatial trail sets in 2D [CZQ*08, EHP*11, LBA10b] and on curved surfaces [LBA10b]; sequence graphs [HET13] and dynamic graphs and eye tracks [NEH12, HEF*14]; directed graphs [SHH11, Mou15, PHT15]; attributed graphs [TE10, PHT15]; parallel coordinate plots [MM08, PBO*14, PW16]; multidimensional projections [MCMT14, RFFT17]; and 3D vector and tensor fields [YWSC12, BSL*14, EBB*15]. Along the growing interest to apply bundling for many data types, a wide array of bundling techniques has been proposed, based on control structures [Hol06], force-directed models [HVW09, DMW07, NEH12, EBB*15]; computational geometry techniques [PXY*05, CZQ*08, LBA10b, EHP*11]; image-based techniques [HET12, BSL*14, Mou15, vdZCT16]; and graph simplification techniques [GHNS11, TE10]. Recent developments using GPU parallelization have made bundling efficiently applicable to datasets of millions of paths [vdZCT16, LHT17].

Bundling goals largely follow those of early methods for simplifying graph drawings [HMM00]. Since then, bundling has become an established tool for the creation of simplified visualizations of edge and trail datasets. However, the rapid development of the field, coupled with the diversity of its application domains, data types handled (*e.g.*, graphs, vehicle trails, eye tracking data, vector and tensor fields, all of them attributed or not and time-dependent or not), and a plethora of algorithmic approaches, make it hard for users to choose the suitable method for a given use-case, and for researchers to focus on important areas of improvement. Bundling is featured, though with limited detail, in a recent survey on image-based information visualization [Hur15], and has gained a prominent place in the set of practical clutter-reduction methods for large graph visualization [SH13]. Another recent survey on large graph visualization [LKS*11] only tangentially touches graph bundling. All above make the case for a dedicated survey on graph and trail bundling.

These challenges have been acknowledged in a recent work [ZXYQ13], which, to our knowledge, is the only survey dedicated to bundling so far. Yet, critical elements for understanding bundling are not covered by [ZXYQ13], such as a large number of existing bundling methods; bundling attributed and time-dependent data; 3D bundling; and interaction techniques for exploring bundled layouts. More generally, the bundling literature lacks a formal discussion on what bundling precisely *is* and which are its exact advantages and limitations. There is so far no framework that allows comparing the different algorithmic solutions from a technical perspective. Relations of bundling with other simplification techniques such as data clustering [JMF99] and image simplification [CM02, SP09] exist but are not fully analyzed. Understanding these can help simplified graph visualization in general. Finally, we lack a taxonomy of all bundling methods, including recent and less mainstream ones. All these issues are frequently mentioned in recent bundling papers and

in discussions in the community. The general perception is that solving such questions is an important step to make bundling evolve from a set of ad-hoc techniques to a more mature sub-field.

In this report, we aim to overcome the above mentioned issues, by the following contributions:

1. **Taxonomy:** We propose a *data-based* taxonomy that organizes bundling methods on the type of data they work on (graphs *vs* trails, which we jointly refer to as *paths*). We show how this taxonomy is more clear-cut than the usual *technique-based* taxonomies that group bundling methods into data-driven, geometric, and image-based (see *e.g.* [vdZCT16, ZXYQ13, HET12]). Moreover, our taxonomy helps users choose suitable algorithms for their data without having to dive into algorithmic details;
2. **Framework:** We propose a formal definition of path bundling, which we next specialize for graph and trail bundling. Next, we propose a generic framework that describes the typical steps of all bundling methods in terms of high-level operations. We next show how such methods implement these steps. This helps one to compare in detail specific steps of specific algorithms, and complements the above-mentioned high-level taxonomy of bundling algorithms with technical insights. Our framework also helps outlining technical limitations of existing algorithms, suggesting future improvement areas.
3. **Task support:** As outlined already, comparing bundling techniques is hard. We address this by proposing a description of *tasks* that bundling aims to address, following [LPP*06, BM13]. We discuss how bundling can address these tasks, and also where salient limitations exist, in terms of the operations of our proposed framework. We also discuss ways (and challenges) to compare the results of different bundling methods. By this, we aim to provide a guide to the practitioner in selecting, customizing, testing, and possibly extending existing bundling methods to support optimally a given task set.
4. **Applications:** Bundling has gone far beyond simplified graph drawing. We overview a wide set of bundling applications and relate these to the above-mentioned tasks. By this, we address limitations of papers which usually focus either on proposing a new technique or discussing a single application.

We start by introducing the main definitions and notations we will work with (Sec. 2). Section 3 presents our data-driven taxonomy and how existing methods fit in it. Section 4 presents a bundling framework that unifies the technical explanation, discussion, and comparison of bundling methods. Section 5 discusses how bundling addresses its main task – clutter reduction – and the further sub-tasks it covers. Section 6 presents a sample of bundling applications that, we argue, covers well the current bundling arena. Section 7 discusses the main advantages, limitations, and potential future work in bundling. Section 8 concludes the report.

2. Definitions

Bundling Objectives: Large-scale, strongly connected, real-world graphs have many more edges than nodes. Hence, classical straight-line node-link drawings thereof quickly become ineffective for most, if not all, tasks they address. This is often referred to as the edge congestion [CR01, WCG03, LKS*11], visual clutter [ED07, BVKW11, NEH13], or hairball problem [SH13]. Bundling is one class of methods that aims to alleviate this problem, along graph clustering and interaction, as further outlined in Sec. 3.1.

Informally put, bundling trades clutter for overdraw [TE10]. However, although there are tens of papers on bundling in the literature, there is – interestingly enough – no formal definition of bundling. We argue that such a definition is needed to be able to understand the process, compare methods, reason about guarantees and limitations, and push further research. We propose such a definition next.

Bundling definition: Let $G = (V, E \subset V \times V)$ be a graph with nodes $V = \{\mathbf{v}_i\}$ and edges $E = \{\mathbf{e}_i\}$. Let d be the dimensionality of the drawing space where the bundled visualization will occur, which is usually 2 or 3. Separately, let $T = \{\mathbf{t}_i\}$ be a so-called trail-set [PXY*05, ZXYQ13]. A trail $\mathbf{t}_i \subset \mathbb{R}^d$ is an oriented curve. Trails typically describe the motion of shapes in space, *e.g.* airplanes [HTC09], eye tracks [PHT15], ships [SWvdW*11], or persons [NPD16]. However, trails can also be curves unrelated to motion, *e.g.* polylines in a parallel coordinate plot (PCP) [Ins09] or DTI tracts [EBB*15]. Note that, in graph theory, the term trail has a different meaning, *i.e*, a type of walk on a graph in which all edges are distinct [HHM08]. Let $\mathscr{G}$ and $\mathscr{T}$ be the spaces of all graphs, respectively trail-sets.

The key unifying element of graphs and trail-sets is a so-called drawing operator D. For graphs, $D : \mathscr{G} \rightarrow \mathbb{R}^d$ is a typical graph layout, or graph drawing, method [TBET99]. By analogy, let $D(\mathbf{e}_i)$ and $D(\mathbf{v}_i)$ be the embedding (drawing) of edges $\mathbf{e}_i \subset E$, respectively nodes $\mathbf{v}_i \subset V$. For trails, the drawing operator is the identity function, *i.e.*, $D(\mathbf{t}_i) = \mathbf{t}_i$, since trails are *already* spatially embedded. Let P denote either a graph G or a trail-set T, called a *path-set*, and $D(P)$ the drawing thereof. A path $\mathbf{p} \in P$ is thus either a graph edge $\mathbf{e}$ or a trail $\mathbf{t}$. Paths can have n additional data attributes, *e.g.* direction, weight, timestamps, name, or type [PHT15, DT14]. Hence, a path $\mathbf{p}$ can be seen as a $n+d$ dimensional, with n data dimensions and d spatial dimensions.

Let $\mathscr{D} \subset \mathbb{R}^d$ be the space of all path drawings $D(P)$. Let $B : \mathscr{D} \rightarrow \mathscr{D}$ be an operator denoting the *bundling* of a path-set; and finally let $B(D(\mathbf{p}))$ denote the curve representing the bundling of path $\mathbf{p}$. B is a bundling method if

$$\forall (\mathbf{p}_i, \mathbf{p}_j) \in P \times P | \mathbf{p}_i \neq \mathbf{p}_j \wedge \kappa(\mathbf{p}_i, \mathbf{p}_j) < \kappa_{max} \rightarrow \delta(B(D(\mathbf{p}_i)), B(D(\mathbf{p}_j))) \ll \delta(D(\mathbf{p}_i), D(\mathbf{p}_j)). \quad (1)$$

Here, δ is a distance metric between $\mathbb{R}^d$ curves, *e.g.* the Hausdorff distance [dBCvKO10]. $\kappa : P \times P \rightarrow \mathbb{R}^+$ is a so-called compatibility function that captures how dissimilar paths are. That is, low κ values indicate very similar paths, and high κ values indicate dissimilar paths, respectively. κ must, in any case, account for spatial similarity in $D(P)$, *i.e.*, when $\kappa(\mathbf{p}_i, \mathbf{p}_j)$ is small, then $\delta(D(\mathbf{p}_i), D(\mathbf{p}_j))$ is small too. In addition, κ can incorporate any of the other n path data-attributes mentioned above, *i.e.*, it can model distance in the $n+d$ layout-plus-attribute space of paths $\mathbf{p}$ [PHT15, LHT17]. Only paths more similar than a threshold κ_{max} should be bundled – otherwise the input drawing $D(P)$ can get too severely distorted to be of any use. Simply put, Eqn. 1 states that the bundled drawings $B(D(\mathbf{p}_i))$ of highly compatible paths are much spatially closer than their unbundled drawings $D(\mathbf{p}_i)$.

Bundling literature often refers to 'edge bundling' or 'graph bundling' indiscriminately, even when the actual data being bundled are trail-sets. It is thus important to clarify both differences and similarities between (the bundling of) graphs and trail-sets:

Graphs *vs* trail-sets – data differences:

- A graph G does *not* have a *given* spatial embedding. Only a graph *drawing* does. So, one bundles graph *drawings*, not graphs. The distinction is crucial, as the drawing $D(G)$ is an extra degree of freedom – the same G can have multiple drawings $D(G)$. Of course, one can use graph information, *e.g.* edge attributes, to influence bundling – see discussion of compatibility function κ above. Yet, having a layout $D(G)$ is mandatory; without it, we cannot bundle a graph as a non-spatial, abstract, object;
- In contrast to graphs, a *trail-set* T is *always* spatially embedded, by definition. This embedding encodes relevant data, *e.g.*, geo-positions of vehicle movements, or variable values in a PCP. Hence, bundling a T is far more delicate than bundling a $D(G)$: Deforming the former can distort spatially-encoded information;

deforming the latter only distorts decisions of the graph-drawing algorithm, but none of the raw data G;
- Both graphs G and trail-sets T can be directed or not. However, most trail-sets are directed, as they represent motion trajetories;
- Nodes $\mathbf{v}_i$ in a graph G are typically shared by multiple edges $\mathbf{e}_i$. After all, this defines a graph. In contrast, endpoints of trails $\mathbf{t}_i$ do not *have* to match. Hence, a trail-set T is a less structured dataset than a graph G.

Graphs and trail-sets – bundling similarities:

- Bundling both graph drawings and trail-sets can be expressed by the same formalism (Eqn. 1). Of course, the meaning of the functions δ and κ can be different for graphs *vs* trail-sets, but the same can hold for two use-cases featuring both graphs or trail-sets;
- Given the above, the same algorithm B can be used to bundle graphs and trail-sets, *if* it delivers a visual simplification (and implicitly, path deformation) which is suitable for the problem at hand. This is well visible in many bundling papers which propose the same bundling method for both graphs and trail-sets;
- The overarching goal of bundling – reducing clutter in a path drawing so that its core structure stands out – is common to both graph drawing and trail-set bundling.

Summarizing: Graphs and trails are completely different data types – the former is not spatially embedded data (for that, we need a graph *drawing*); the latter is spatial by definition. Trails are typically directed, while graphs may not be. Many algorithms can technically bundle both graph drawings and trail-sets, the choice of an algorithm being suitable or not being driven by more subtle *application* factors like definition of compatibilities, amount of bundling deformation, and types of emphasized patterns. All these aspects matter for understanding (and optimally using) bundling, as we shall see next.

2.1. Bundling requirements

To discuss and compare bundling methods next (Secs. 3,4), we need a few general requirements on bundling. Distilled from the bundling techniques reviewed in this paper, these requirements are as follows:

Input: A bundling method B accepts a path-set drawing $D(P)$ as input – that is, a set of spatial positions connected by curves. This is in contrast with graph layout methods which typically *compute* such positions from a graph G;
Output: The output $B(D(P))$ of a bundling method is a path drawing having the same endpoints as the input $D(P)$. No bundling method that we are aware of (except [YWSC12], see Sec. 3.3.1.2) changes path endpoints, as these are assumed to store important information;
Bundle definition: Bundling methods do not explicitly define what a *bundle* is. Bundles are defined implicitly, as sets of paths that share sufficient similarity so as to be represented as a compact graphical shape. Interestingly, this matches the definition of clustering [JMF99] or image segmentation [Sze10] – a segment or cluster shares precisely the same properties. So, bundling can be seen as a clustering or segmentation of the drawing $D(P)$. A constraining criterion in the above is that bundles are assumed to be spatially thin (to reduce edge congestion, see Sec. 2) and reasonably low-curvature (so as to be easily visually traceable, see Sec. 5);
Density sharpening: All bundling methods we know of aim, implicitly or explicitly, to *sharpen* the spatial edge density ρ of $D(P)$, *i.e.*, the number of paths $D(\mathbf{p})$ drawn per unit screen space. Simply put: In areas where ρ is low (few paths), these paths are shifted to make extra empty space, which declutters the overall image; paths are shifted to close regions where ρ is already high (many paths exist). This essentially trades off clutter for overdraw – the number of intersections of *bundles* should be significantly lower than the number of intersections of input *paths*. This matches the known principle of ink minimization in information visualization [Tuf92]. Overall, this makes visual end-to-end tracing of *bundles* easier than end-to-end tracing of individual *paths*;
Scalability: Bundling becomes interesting for large path-sets (tens of thousands up to millions of paths). For small(er) path sets, classical graph drawing methods suffice [GN00, HMM00, LKS*11], as there are too few edges to create the infamous visual clutter discussed above. Bundling *must* thus cope with large path-sets, otherwise its reason to be is not warranted.

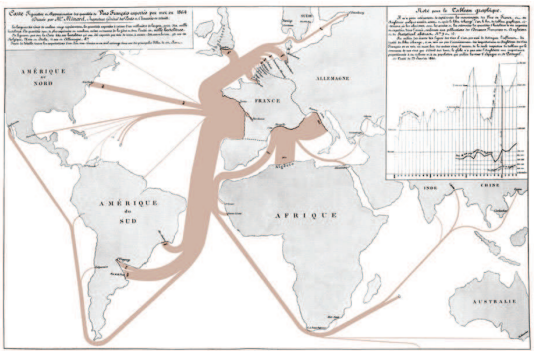

Figure 1: *Early techniques related to bundling: Flow map of French wine exports (Minard, 1864).*

3. Taxonomy of Bundling Methods

3.1. Preliminaries

Graph and trail-set bundling have a long history, stemming from applications related to the visual simplification and clutter reduction in the drawing of graphs using node-link diagram metaphors. We outline next early efforts in the area and how they relate to what is currently understood by bundling.

Graph simplification: An early approach to simplification and clutter reduction was to simplify the structure of G, by creating a smaller G' (with fewer nodes and/or edges) that captures the main structure of G. Edge concentration is such a method [New89] used to simplify Sugiyama-style layouts [STT81]. Here, edge sets having the same set of start and end nodes are replaced by a so-called concentration node, which effectively presents a simple form of polyline-style bundling. Many other graph simplification methods exist, as surveyed in [Sch07]. However, such methods display a smaller and/or different graph than G, and as such specific nodes of potential interest are omitted in the drawing. Finding these requires additional interactive exploration [AvHK06, AMA08, APP10]. Moreover, most graph simplification methods do not fit the scope of bundling, as currently understood by the infovis or graph drawing communities, or as defined in Sec. 2, so we do not explore these further.

Graph drawing simplification: A separate way was proposed by methods that change the graph drawing $G(D)$ (as opposed to the graph G) to reduce clutter. These can be seen as bundling precursors. For example, Brandes *et al.* use a mix of straight lines and Bézier splines to draw a general undirected graph where nodes are train stations and edges are train routes, respectively [BW98]. Spline control points of spatially close train routes are grouped together. This also introduces the concept of a 'control mesh' which is computed to further bundle edges. Dickerson *et al.* introduce the notion of confluent drawing, where "groups of edges are merged together and drawn as tracks" [DEGM03]. Compared to modern bundling methods, this approach can only handle a subset of all possible general graphs. Flow maps extend the idea by hierarchically clustering a set of nodes $\mathbf{v} \in V$, positions $D(\mathbf{v})$, and directed edges $\mathbf{e} \in E$ of a graph (V, E) [PXY*05], yielding organic, branch-rich images of the graph structure which are similar to early hand-drawn Sankey diagrams showing flows over a geographical map (Fig. 1).

Graph drawings								Trail sets				
Static						Dynamic		Static				Dynamic
Hierarchical	General				3D	Sequence	Streaming	2D			3D	(streaming)
compound	undirected	directed	flowmaps	confluent drawing				undirected	directed	PCP		
[Hol06, TE10]	[HVW09, LBA10b]	[SHH11]	[PXY*05]	[DEGM03, DEGM05]	[CZB11]	[HEF*14]	[NEH12]	[HET12]	[PHT15]	[MM08, TA08]	[EBB*15]	[HEF*14]
[PNK10, TDT13]	[CZQ*08, GHNS11]	[LDB11]	[VBS11]	[VBS11, NB13]	[BSL*14]	[HET13]		[vdZCT16]	[LHT17]	[HLK*12, ZYQ*08]	[LBA10a]	[HET13]
[HvW08, TA08]	[NHE11, LLCM12]		[BSV11]	[BSV11, BRH*16]	[GBE08]	[Han13]		[EHP*11]	[Mou15]	[PBO*14, PW16]	[YWSC12]	

Table 1: *Data-based taxonomy of graph and trail-set bundling methods. For space reasons, only the main methods in each class are listed. Bundling papers that present applications, but do not introduce a new bundling technique, are not listed here.*

Interaction: A third way to alleviate clutter is to use interaction, *e.g.* to navigate a simplification hierarchy [DS13, AvHK06, vHP09, HFM07] or interactively declutter focus areas of interest by removing or bending drawn edges [WCG03, GKN04, TAvHS06, WC07]. In contrast to bundling, methods offer a local, on-demand, decluttering and simplification, rather than a global, automatic, one.

In 2006, several papers that introduce bundling as we understand it today were published. Gansner and Koren's improved circular layouts by clustering edges (drawn as straight-lines) based on spatial proximity, and next bending edges in a cluster towards the cluster's centroid line [GK06]. Qu *et al.* propose bundling for general straight-line graph drawings using NURBS splines whose control points are constructed from a Delaunay triangulation of the graph nodes [QZW06]. Most notably, Holten presented hierarchical edge bundling which was able, for the first time, to bundle compound graphs of thousands of edges and having arbitrary node layouts [Hol06]. With this, the age of modern bundling had begun.

Taxonomy: We propose a *data-based* taxonomy that organizes bundling methods along the type of data they work on – *i.e.*, graphs *vs* trails. We argue that this taxonomy is more clear-cut than existing *technique-based* taxonomies that group bundling methods into data-driven, geometric, and image-based (see *e.g.* [vdZCT16, ZXYQ13, HET12]). Moreover, our taxonomy helps both researchers and users to understand the bundling landscape without having to dive into the technicalities of specific methods. A data-based taxonomy (though, a different one) has also been used in a recent survey to classify graph visualization methods [LKS*11].

We organize bundling methods based on the type of *data* they work on. At the highest level, such data can be split into *graphs* (Sec. 3.2) and *trail-sets* (Sec. 3.3). Further taxonomy levels refine graphs and trail-sets based on additional data characteristics, such as type of graph, direction information, time-dependency, and dimensionality d of the drawing space (see Tab. 1). The proposed taxonomy is detailed next.

3.2. Graph Bundling Methods

Graph bundling methods expect as input a graph drawing $G(D)$. Edges $D(\mathbf{e} \in G)$ in such drawings are typically straight lines. When this is not the case, edge drawings do carry information, so deforming them by bundling should be done with great care. Such cases fit better in the class of trail-set bundling, which is discussed separately (Sec. 3.3).

3.2.1. Static graphs

Static graphs have nodes V and edges E which do not change in time. For such graphs, bundling methods can be further classified as follows.

3.2.1.1. Hierarchical compound graphs Hierarchical compound graphs are graphs $G = (V,E)$ where V are the leafs of a separate tree $T = (V^T, E^T)$. Such graphs are often encountered in relational datasets whose items can be organized via a hierarchy, such as dependencies between software components [Die08]. Edges in E are typically called associations.

The most famous bundling method for hierarchical compound graphs is Hierarchical Edge Bundles (HEB) [Hol06]. HEB draws the tree T using *e.g.* a radial (also called a chord diagram [Mun14]), balloon, or treemap layout. In case of the radial layout (Fig. 2a), the hierarchy T is shown by a technique known as icicle plots [KL83]. This yields a tree drawing $D(T)$ where nodes $\mathbf{v} \in E$ that have a close common ancestor (*e.g.*, father or grandparent) are close to each other. Next, edges $\mathbf{e} \in E$ are drawn as B-splines whose control points are the node positions in $D(T)$ – in other words, $D(T)$ serves as a bundling control mesh (see Sec. 3.1). Since control points are shared, the drawn (curved) edges get closer to each other than the original straight-line edges, thus the second part of Eqn. 1 is respected.

Given the relation between node closeness in the hierarchy T and their placement in $D(T)$, association edges starting or ending in the same subtree of T are bundled together – in other words, the compatibility function $\kappa(\mathbf{p}_1, \mathbf{p}_2)$ (Eqn. 1) reflects how close the endpoints of two edges $\mathbf{p}_1$ and $\mathbf{p}_2$ are in T. HEB is very simple to implement and scales very well with the sizes of G and T. HEB has been used in many applications in software engineering [HvW08, CZH*08, DT14, RVET14], social sciences [KS10, JGH11],

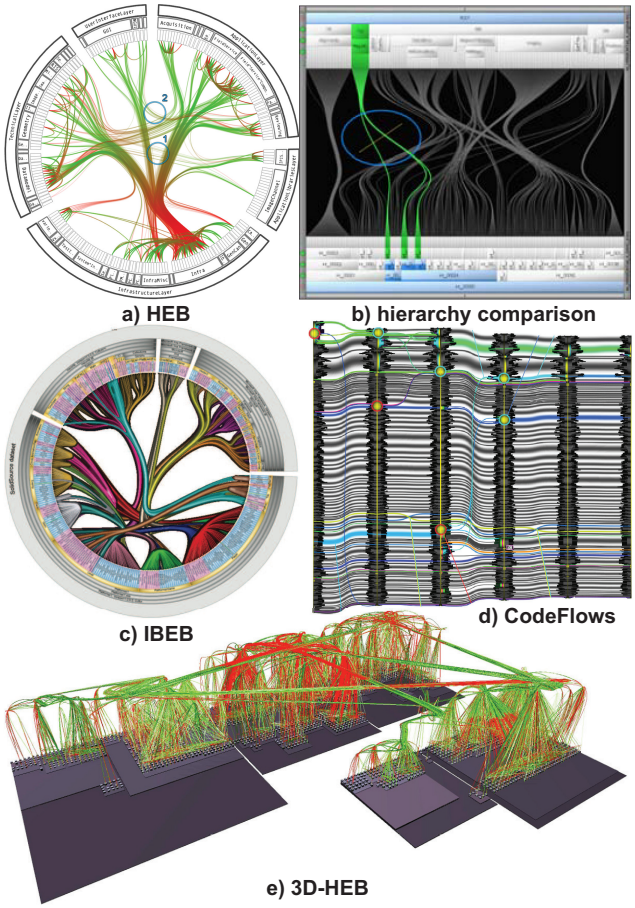

Figure 2: *Hierarchical bundling methods.*

web ontologies [HdRFH12], text data [CC07], and life sciences [BSL*14,AP08,EBB*15]. HEB enhancements that simplify the bundled drawings [TE10] (Fig. 2c) are further discussed in Sec. 4.3.3.

The hierarchy T required by HEB can be part of the input data or can be constructed from this data. For the latter, Jia *et al.* use bottom-up graph and data clustering techniques [Sch07, JMF99] to identify strongly-related node groups, thereby extending HEB to general attributed graphs. In other words, when T is given, κ reflects its structure (as explained earlier); while, when T is computed, this is done based on a given κ defined on the data. Other HEB extensions include the comparison of two hierarchies T_1 and T_2. Here, association edges E link leaf pairs in T_1 and T_2 rather than leafs in the same tree. Applications include the comparison of two [HvW08] (Fig.2b) or multiple [TA08] software hierarchies (in a kind of 'structural diff' metaphor, Fig. 2d), where T_1, T_2 are given by the software structure; comparison of program execution traces [TDT13]; and linking related items in coordinated multiple views to show multiple relations and datasets; here, the hierarchies T_i are computed by similarity-based node clustering, as discussed earlier. HEB was also used for 3D bundling (Fig. 2e), further detailed in Sec. 3.2.1.3. Image-based simplification of HEB drawings (Fig. 2c) is further discussed in Sec. 4.3.3.

Pupyrev *et al.* [PNK10] improve the classical Sugiyama-style graph drawing algorithm [STT81] for directed acyclic graphs (DAGs) by bundling the curved edges created by Sugiyama in case these have the same start and end nodes. Hence κ reflects the similarity of endpoints of edges. The method is tested on relatively small graphs (tens of nodes). Although this method, strictly speaking, requires a DAG (which is more general than the hierarchical compound graphs required by HEB), we put this method in the same class as HEB, since the Sugiyama algorithm operates by extracting a tree from the input DAG.

Hierarchical compound graph bundling methods are arguably the most successful (and best known) use-case for graph bundling. Key to this is the ability of bundling to summarize groups of similar relations (edges) *and* the fact that a hierarchy allows a simple, consistent, and scalable way to compute and/or encode the similarity κ. However, the quality of drawings produced by such methods visibly depends on the way the tree T is laid out, as this next influences how bundles are routed. Yet, most papers in this area comment little on different ways to lay out T, beyond the fact that standard force-directed or radial tree-drawing algorithms [TBET99] can be used. As an exception, the original HEB method [Hol06] and several of its refinements [RVET14, RVT11] propose controlling of the distances between the circles on which the same-depth-from-leaves layers of T get laid out, thereby allowing one to spread or compress the bundled drawing in the available visual space. However, which layouts for T are best for certain tasks or drawing styles, is still a topic for further research study.

3.2.1.2. General graphs General graphs do not have a hierarchy structure. Bundling methods are further specialized on directed *vs* undirected graphs, as follows.

Undirected graphs Undirected graphs are the most general class of static graphs targeted by bundling. The key challenge here is to define the compatibility function κ to take into account both spatial information present in the graph layout $D(G)$ and attribute information present in G itself.

Force-directed edge bundling (FDEB, Fig. 4b), the earliest method in this class, defines κ to include only geometric information in $D(E)$ [HVW09]. Given two segments $D(\mathbf{e}_1)$ and $D(\mathbf{e}_2)$ represented the drawings of two edges $\mathbf{e}_1$ and $\mathbf{e}_2$, κ includes their angle, relative distance in $\mathbb{R}^2$, ratio of lengths $\|D(\mathbf{e}_1)\|$ and $\|D(\mathbf{e}_2)\|$, and skewness (for more details, see Sec. 4.1.2.1). Next, edges are sampled into sample points $\mathbf{x}_i \in D(E)$, and each $\mathbf{x}_i$ is iteratively displaced to get closer to all other sample points $\mathbf{x}_j$ of compatible edges. Interestingly, FDEB did not cover the case of directed graphs, although this would have been reasonably easy to do.

Nguyen *et al.* propose TGI-EB to extend the compatibility measures in FDEB to account for importance-based compatibility (defined in terms of a function κ based on the Euclidean distance of n-dimensional edge attributes), and topology compatibility, based on the position of an edge in the graph G [NHE11]. This richer palette of compatibilities allows more precise control of which edges get bundled, which in turn supports application-dependent analyses and a rich set of drawing styles.

Both FDEB and TGI-EB have no explicit control mesh – edges are drawn closer to each other rather than to a unique skeleton. Contrasting this, Geometry-Based Edge Bundling (GBEB, Fig. 4g) uses the control mesh strategy [CZQ*08]: Edges in a graph drawing are clustered in a bottom-up fashion, based on the edges' positions and orientations, yielding a control mesh whose edges tend to orthogonally 'cut' across regions having many similar-orientation edges. Mesh-edge intersections are furthered clustered to yield the shared control points through which the curved edges are finally routed. GBEB allows control meshes to be generated either automatically or with user input, the latter allowing local spatial control over which regions of $G(D)$ one wants to bundle. However, it has been noted that the result quality highly depends on the control mesh's quality, which in turn is hard to guarantee [LLCM12].

Luo *et al.* further refine the idea of control meshes to produce a so-called ambiguity-free bundling [LLCM12]. They observe that highly bundled images, such as produced *e.g.* by HEB or FDEB, have difficulties in tracing edges end-to-end. To alleviate this, they propose a simple compatibility κ which is non-null only for edges sharing a node and which are also close in $D(G)$. They also remove ambiguities caused by the bundled edges $B(D(\mathbf{e}))$ passing close to unrelated nodes in $D(G)$ by re-routing (repelling) the former from the latter. Finally, by using a small number of control points, the smoothness and low-curvature of edges is favored, which also allows their visual following. To reason about the relative positions of nodes and edges, a special quadtree, built from node positions, but storing also which cells are crossed by which edges, is built. The method produces easy-to-follow bundlings on small graphs (under hundred nodes and edges). For large graphs (thousands of nodes or edges), the method is arguably less effective, as its relatively weak bundling will still cause visual clutter.

Related to [CZQ*08] and [LLCM12], Winding Roads (WR, Fig. 4c) computes a control mesh using quadtrees and Voronoi diagrams [LBA10b]. WR also supports routing bundles to avoid unrelated nodes, being the first general-graph method that demonstrates such results for relatively large graphs (thousands of edges).

All general-graph methods discussed so far are quite limited in scalability, either computationally [HVW09, NHE11, CZQ*08] or in terms of the largest graph they can bundle with limited clutter [LLCM12]. MINGLE [GHNS11] addresses the former by a computationally-scalable ink-saving principle, similar to [GK06]: The ink for drawing a bundle $\bigcup_i B(D(\mathbf{e}_i))$, *i.e.* number of pixels covered by $\bigcup_i B(D(\mathbf{e}_i))$, should be smaller than the ink used for drawing the same unbundled edges $\bigcup_i D(\mathbf{e}_i)$. Note that the latter is roughly equal to $\sum_i \|D(\mathbf{e}_i)\|$, as edges overlap very little in a typical straight-line graph drawing. MINGLE proceeds bottom-up, by finding close edges (that have a high bundling chance, see Eqn. 1 and related text), and bundle these in a greedy way as long as ink is saved. The process is repeated recursively by adding parts of the so far unbundled edges to existing bundle parts. A polygonal control mesh is thus created, based on the centroids of the edge-sets identified as compatible. The process is very similar to bottom-up hierarchical clustering using average linkage [JMF99]. MINGLE can bundle graphs of up to a mil-

lion edges that no other general-graph method discussed so far can handle. However, it creates less smooth, and thus harder to visually follow, overall results (see Fig. 4a).

Directed graphs An already early criticism of general-graph bundling methods is that they do not take into account edges' directions: For many applications, finding if two groups of nodes $V_1 \subset D(G)$ and $V_2 \subset D(G)$ are connected by a bundle is not sufficient; we want to specifically see if there are edges from V_1 to V_2, or conversely, or both. This is essential when edge directions carry semantics, such as when exploring a software system's call graph to assess modularity [DT14]. A simple way to do this is to color-code edges using a source-to-destination categorical color gradient [Hol06, CZH*08, RVET14]. However, when edges of opposite directions co-exist in a bundle, color mixing occurs, which makes the assessment of categorical colors very hard, let alone seeing how many edges of each direction the bundle has (see further Sec. 4.3.1).

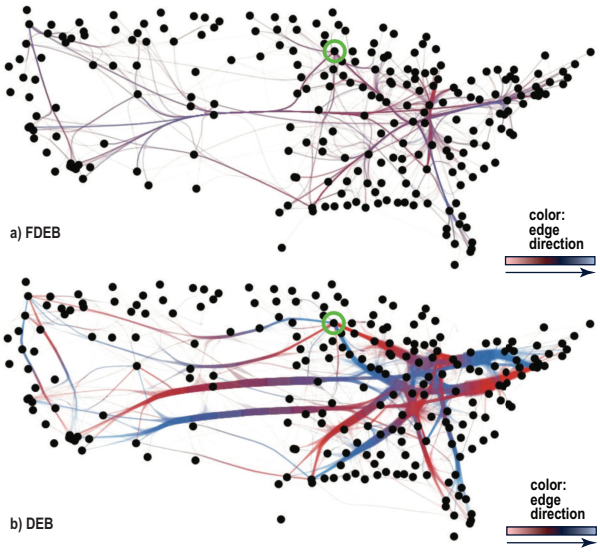

Figure 3: *Comparison of FDEB and DEB for the* US airlines *graph (|N|=235,|E|=2101). See Sec. 3.2.1.2.*

To solve this, methods for directed graphs have been proposed. Key to these is that the compatibility $\kappa(\mathbf{e}_i, \mathbf{e}_j)$ includes the directions of edges $\mathbf{e}_i$ and $\mathbf{e}_j$. The first method of this type is Divided Edge Bundling (DEB) [SHH11]. Simply put, DEB extends FDEB to incorporate edge direction in κ: Same-direction edges have a positive κ, while opposed direction ones have a negative κ, respectively. Since FDEB uses κ as the amount to shift edge sample points during its iterative bundling process (see above in Sec. 3.2.1.2), same-direction edges are treated as in FDEB, whereas opposed direction ones are repelled from each other. Atop the above, DEB also enhances κ to include edge weights, so that more important edges are bundled less, thus determine the outcoming $B(D(E))$ more than less important ones. Finally, DEB adds a connectivity compatibility term equal to $1/(1+\Delta(\mathbf{e}_i, \mathbf{e}_j))$ for two edges $\mathbf{e}_i$ and $\mathbf{e}_j$, where $\Delta : E \times E \to \mathbb{N}$ is the shortest-path distance in G between the nodes of $\mathbf{e}_1$ and $\mathbf{e}_2$. Overall, DEB can separate opposed direction edges quite well, and makes direction color-coding effective; however, in contrast to undirected methods, *e.g.* FDEB, DEB takes more screen space, and thus increases clutter (Fig. 3). Separately, WR (introduced earlier) is refined by using a quadtree-only control mesh, as opposed to its more general triangle control mesh, to create directed and orthogonal bundles [LDB11] following the style of metro map drawing [Wol07].

Flow maps Flow maps can be seen as a particular subcase of directed graphs. More specifically, these are directed acyclic graphs (DAGs) having a single (or a very few) source node(s). In a transportation or data flow network, they describe how information flows from the source to reach all nodes in the graph. Since the source is unique, flow maps do not need to show edge directions explicitly such as in directed graph bundlings – the direction of data flow can be inferred by doing a visual path tracing from the source to every node of interest. Another feature of flow maps is the ability to show the amout of flow between adjacent nodes, quantified as the (weighted) number of edges linking such nodes. This allows discovering how the total amount of data outflowing from a source is spread over the graph. This is usually done by scaling the thickness of a bundle by the amount of information flowing through it, a technique pioneered by Sankey diagrams [Tuf92]. The first automated flow map generation used a straight-line single-level tree drawing linking a source with all destinations, using edge thicknesses to indicate flow amounts [Tob81].

The first, and most known, flow map algorithm using bundling [PXY*05] was already mentioned in Sec. 3.1. Given a DAG drawing $D(G)$ and a source node $\mathbf{s} \in G$, a spanning tree $ST(\mathbf{s}, G) \subset G$ rooted at $\mathbf{s}$ is constructed. Next, curved edges linking $\mathbf{s}$ to all other nodes in G are created and routed along $ST(\mathbf{s}, G)$, using a control point technique similar to HEB (Sec. 3.2.1.1). Edge fragments sharing the same path are coalesced to yield bundles of variable thickness. Multiple sources $\mathbf{s}_i$ can be treated by superimposing the flow maps created by each of them – note, though, that this is not the same as having a true multiple source flow. Verbeek *et al.* [VBS11] further reduce the confusing bundle crossings produced by [PXY*05], and produce overall lower-curvature bundles, by using the spiral tree drawing algorithm of Buchin *et al.* [BSV11] to compute the skeleton to route edges along. Computation of the optimal tree is done by moving its nodes to optimize a cost function that accounts for avoiding obstacles and producing smooth bundles. Bundled flow maps have been found to scale less well with respect to the number of trails as compared to other visualizations of geographical trail sets, such as OD Maps [WDS10] and MapTrix [YDGM17].

Confluent drawings Confluent drawings have closely related aims to flow maps, *i.e.*, show end-to-end relations between nodes in a graph drawing with as little ambiguity as possible. They inherently propose bundling in the sense of merging parts of edges that (a) simplify the drawing but (b) do not adversely affect the above-mentioned edge tracing task. Early methods can handle relatively small graphs of a particular category called confluently drawable [DEGM03, DEGM05]. Latter methods use a help structure, the power graph [RRAS08], which can be computed by various heuristics [DMM*14, DHRMM13]. In this sense, they resemble flow maps which also use a tree (*e.g.* spanning [PXY*05], Steiner [BSV11] or spiral [VBS11]) to route edge bundles. In brief, power graphs provide a way to group nodes and edges in a graph in terms of how they are connected. Power graph nodes are next used to route edges between node groups so as to make the group-level interconnections easier separable visually than when using standard bundling. Although older confluent drawings have been restrictied to planar graphs, recent work proposed a confluent bundling technique that can effectively handle general graphs [BRH*16]. Separately, the spiral trees in [BSV11] are further refined in [NB13] to allow for more inflection points along a bundle, which in turn allows easily tracing bundles end-to-end.

3.2.1.3. 3D graph layouts All bundling techniques discussed so far expect as input a two-dimensional (2D) graph drawing, *i.e.*, $D(V) \subset \mathbb{R}^2$. However, graph layouts can also produce 3D node positions. These are particularly useful when the underlying graph attributes, or problem to be solved, has a 3D nature. In such cases, one needs to bundle a 3D graph drawing. In theory, all bundling methods presented so far could be extended to handle 3D layouts. However, the many design and implementation decisions they are based on make this impractical from a computational complexity and/or im-

plementation simplicity perspective. As such, specific 3D bundling algorithms have emerged.

A first way to bundle in 3D is to extend HEB [Hol06] to handle 3D node positions [GBE08]. This has a low computational complexity, but only works for compound graphs, as outlined in Sec. 3.2.1.1. In [GBE08], the 3D layout is given by a treemap augmented with bar charts to show the structure, respectively quality metrics, of a software system, a metaphor known as a 'code city' [WL07]. A similar idea, called 3D-HEB, used also to visualize compound graphs from software engineering, is described in [CZB11] (Fig. 2e). In contrast to [GBE08], the third dimension is used here to pull the bundles above the 2D treemap layout so as to reduce occlusion. 3D bundling is used also to visualize bike journeys in a city [NPD16]: Given two drawings of the same city map, placed parallel to each other in 3D, straight lines connecting the start and end of journeys (one end in each map) are bundled to yield a simplified traffic view.

In contrast to the above, Böttger *et al.* consider a graph capturing functional brain connectivity [BSL*14]. Here, nodes represent 3D positions in a human brain. Straight-line edges are bundled using the 3D kernel-density bundling method (KDEEB [HET12]) discussed next in Sec. 3.3.1.1 adapted to 3D. In contrast to 3D path bundling (Sec. 3.3.1.2, edge deformation is not a problem, as these edges represent only abstract connections. The method produces convincing results, but is relatively slow, due to the high complexity of 3D kernel density estimation. A similar method is proposed in [ZWHK16]. Here, FDEB is used instead of KDEEB for bundling, and bundling speed is increased by adding a similar edge pre-clustering step, thereby reducing the number of pairs $(\mathbf{p}_i, \mathbf{p}_j)$ on which the compatibility κ needs to be computed (Eqn. 1).

3.2.2. Dynamic graphs

A dynamic graph $G(t) = (V(t), E(t)), t \in \mathbb{R}^+$ is a graph where both nodes and edges are time-dependent. That is, at each moment t, we have a potentially different graph $G(t)$ to explore. $G(t)$ is also called a *frame*, by analogy with motion video. Two different forms of dynamic graphs are known: streaming graphs and sequence graphs. The difference between them, and bundling methods targeted at them, are outlined in Secs. 3.2.2.1 and 3.2.2.2.

However, several aspects are common to bundling both streaming and sequence graphs, so we outline these commonalities first. First, bundling dynamic graphs is strongly related to drawing dynamic graphs [HEW98, SFPY07, BBD*10]. A recent survey on the field was proposed by Beck *et al.* [BBDW14]. Two main classes of methods exist here: Small multiple methods draw graphs $G(t_i)$ at a user-selected set of sample moments $\{t_i\}$ side-by-side, using the same visual mapping. To allow comparison, the layout algorithm used to construct $D(G(t_i))$ should be *stable* – that is, small changes in $G(t_i)$, as opposed to close time moments t_j, should correspond to small changes in the drawing $G(t_i)$ as opposed to $D(G(t_j))$. This requirement is also known as maintaining the user's mental map [BBDW14]. Animated methods continuously display $D(G(t))$ for ranges of interest of t, again using the same visual mapping. Layout stability is also required; if met, it allows users to see changes in areas where $D(G(t))$ changes and stable data where the drawing stays unchanged, respectively. Small multiple methods have the advantage that they allow, in principle, comparing any two frames $G(t_i)$ and $G(t_j)$. However, they typically do not scale to more than a few tens of frames. Animated approaches scale, in theory, to an unbounded number of frames. However, users cannot memorize a drawing's evolution over long periods of time, so comparing frames far apart in time requires interactive seek-and-replay of the animation. Separately, animation should be smooth, as too many sharp transitions between consecutive frames are perceived as disruptive.

3.2.2.1. Streaming graphs In a streaming graph $G = (V, E)$, each edge $\mathbf{e}_i \in E$ has a so-called *lifetime* $[t_i^{start}, t_i^{end}]$, of duration $\lambda_j = t_i^{end} - t_i^{start}$, where $t_i^{start} < t_i^{end}$. That, is, $\mathbf{e}_i$ exists only between t_i^{start} and t_i^{end}. The dynamic graph $G(t)$ thus contains all edges $\mathbf{e}_i \in E$ that are alive at t, *i.e.* for which $t_i^{start} \leq t \leq t_i^{end}$. The same holds for the streaming graph's nodes $\mathbf{v}_i \in V$. Streaming graphs can be available in an online manner – that is, one does not know upfront all moments t_i^{start} and t_i^{end}.

Nguyen *et al.* proposes StreamEB to deal with streaming graphs. Given two edges $\mathbf{e}_i$ and $\mathbf{e}_j$ of a streaming graph, StreamEB extends the TGI-EB method for undirected graphs [NHE11] to add to the compatibility κ a temporal term, based on the lifetime overlap $|t_i^{start} - t_j^{start}| \cdot |t_i^{end} - t_j^{end}|$ and the duration difference $|\lambda_i - \lambda_j|$ of the two edges. After this, FDEB is applied on all edges falling in a time-window $[t, t + \Delta t]$ that slides to cover the entire time-range of $G(t)$. Here, Δt is a time interval small enough so $G(t)$ doesn not exhibit too many changes, but large enough to show enough interesting changes to the user. If the speed of change of $G(t)$ is small in comparison with the speed of advancing of the sliding window, and the underlying static bundling method $B(\cdot)$ being used is continuous (in a Cauchy or Lipschitz sense) with respect to small changes in the graph, *i.e.* adding or removing a few edges from $D(G)$ only slightly changes $B(D(G))$, then the dynamic bundling $B(\cdot, t)$ proposed by StreamEB will also be continuous in time. As explained earlier, this is a desirable property for maintaining the user's mental map. However successful in this respect, StreamEB has a very high computational cost: Stable bundling static methods, such as FDEB [HVW09] or GBEB [CZQ*08] are quite expensive, as already explained. Faster bundling algorithms, *e.g.* MINGLE [GHNS11] or [LBA10b, EHP*11] are significantly more sensitive with respect to small changes in the input graph drawing. These problems are solved by more recent bundling methods for dynamic graphs and trail-sets, see Sec. 3.3.2.

3.2.2.2. Sequence graphs Sequence graphs are, as their name says, ordered sets $G = \{G^i\}$ of static graphs $G^i = (V^i, E^i)$. In contrast to streaming graphs, edges E_i do not have a lifetime, but belong to a single frame i. They typically capture a system's structure at several discrete time moments t_i. Well-known examples are the set of call graphs mined from the several revisions of software system stored in a software repository [DT14, RVET14]. In practice, the frames G^i are usually not unrelated, but have nodes and edges which capture the evolution in time of the same *items*. For instance, two edges $\mathbf{e}_a^i \in E^i$ and $\mathbf{e}_b^{i+1} \in E^{i+1}$ can represent the same call relation in two consecutive frames of a software system. Such links relating information between different sequence frames can be modeled by a so-called correspondence function $c : E^i \to \{E^{i+1}\}$. Here, $c(\mathbf{e} \in E^i)$ yields an edge $\mathbf{e}' \in E^{i+1}$ which logically corresponds to $\mathbf{e}$, if such an edge exists in E^{i+1}, or the empty set, if there is no correspondence, *i.e.* if $\mathbf{e}$ disappears (dies) in the transition from G^i to G^{i+1}.

Sequence graphs can be bundled by using the sliding-window technique in StreamEB described earlier. A much simpler and faster method is proposed in [HET13]: A static bundling method is applied to each G^i independently, yielding a sequence of bundled layouts $\{B(G^i)\}$. Next, for each $\mathbf{e} \in G^i$, if $c(\mathbf{e}) \neq \varnothing$, the bundled edge $B(D(\mathbf{e}))$, represented as a polyline, is linearly interpolated towards $B(D(c(\mathbf{e})))$, else $B(D(\mathbf{e}))$ is interpolated towards the straight-line segment linking the endpoints of $D(\mathbf{e})$ and the interpolated edge drawing is faded out. This signals that $\mathbf{e}$ disappears from G^i to G^{i+1}. A symmetric procedure is applied to interpolate edges that appear from G^i to G^{i+1}. Linear interpolation guarantees smooth (piecewise first-order continuous) changes in the bundled edges' positions and opacities, which preserves the mental map. To account for changes in the node set V^i, a single global layout of the union graph $\cup_i G^i$ is done upfront, so node positions $D(V^i)$ do not change. If a stable static bundling technique B is used, the method thus guarantees that big changes in the visualization correspond to appearing

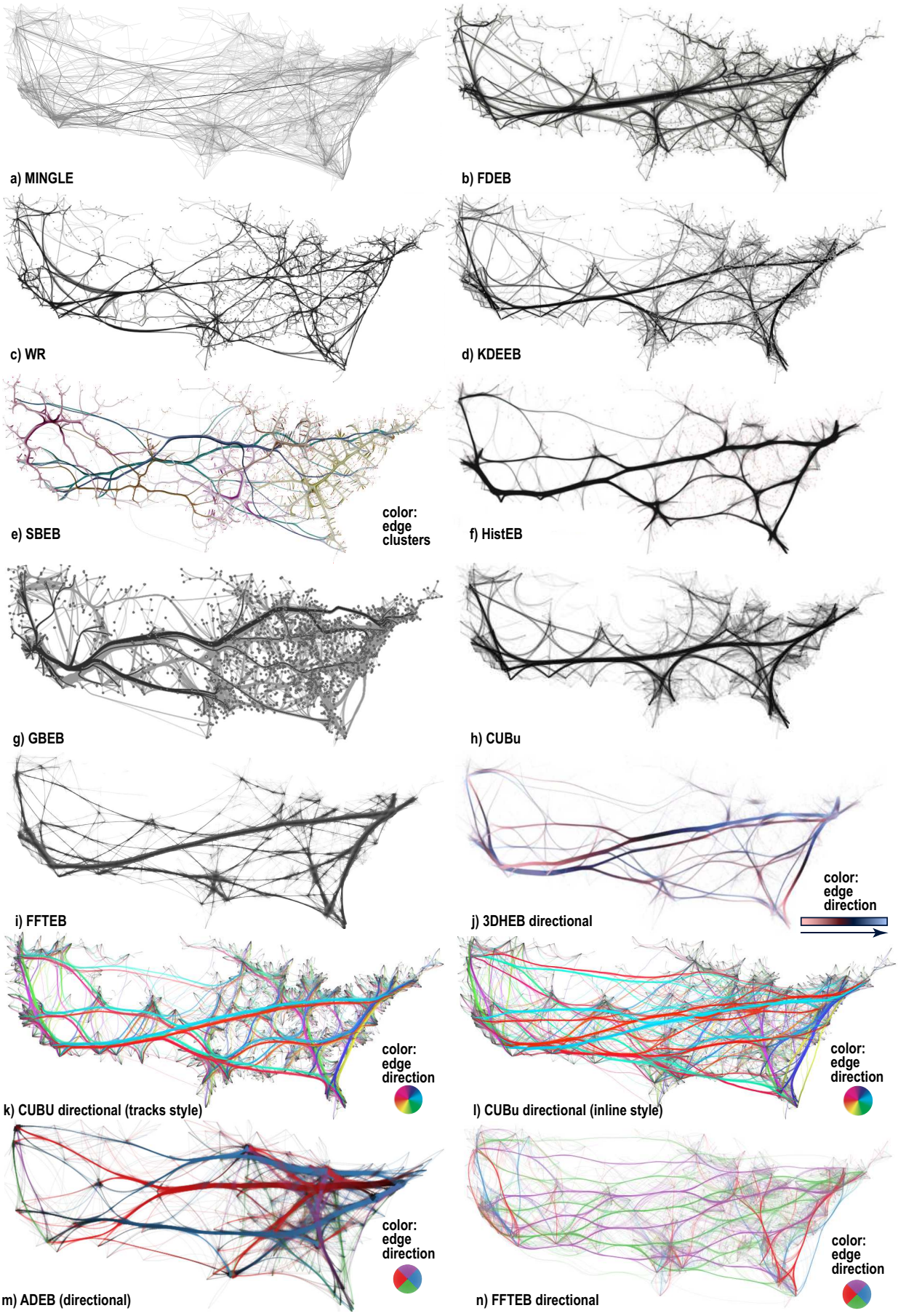

Figure 4: *General undirected (a-h) and directed (i-k) bundling methods,* US migrations *dataset (|N|=1715,|E|=9780). See Sec. 3.2.1.2.*

(progressively bundled) and disappearing (progressively unbundled) edges. Edges that do not appear nor disappear move less in the dynamic bundling. Hence, the amount of visual change encodes well the amount of change in the graph. However simple and scalable, this method cannot directly handle streaming graphs, where edges have arbitrary lifetimes, and no explicit frames nor inter-frame correspondence data exists. Hanjalić further adapted this method for the visualization of code clones from software repositories [Han13].

3.3. Trail-set Bundling Methods

As introduced in Sec. 2, trail-sets consist of (typically non-straight) oriented curves $\mathbf{t}$ that describe the motion of objects in Euclidean $\mathbb{R}^2$ or $\mathbb{R}^3$ space. Hence, trail-set bundling methods have extra information and constraints to consider as compared to graph bundling methods. Conversely, the only family of graph bundling methods that uses graph-specific information not present in trail-sets are the hierarchical ones (Sec. 3.2.1.1). Hence, apart from hierarchical graph bundling, trail-set bundling can be seen as a superset of general graph bundling – one can use trail-set bundling methods to bundle general graph drawings, but, in general, not conversely. We next group trail-set bundling methods based on the data type they work on.

3.3.1. Static trail-sets

Static trail-sets are trails which do not change in time. However, time information *can* be present on such trails, *e.g.*, the time moments when a vehicle has reached each point of a trail $\mathbf{t}$ [HTC09, SWvdW*11, SHVDWVW16]. We distinguish the bundling of 2D *vs* 3D trail sets, as follows.

3.3.1.1. 2D trail-sets 2D trail sets can be further classified in undirected and directed ones, as follows.

Undirected trails: Kernel density estimation edge bundling (KDEEB, Fig.4d) [HET12] observed, first, that a bundled drawing $B(D(T))$ has a locally either lower (outside bundles) or higher (within bundles) spatial trail density than the unbundled drawing $D(T)$. Hence, B can be cast as a density-sharpening operator, like the well-known mean shift [CM02]. Following this analogy, bundling consists of repeatedly computing the gradient of the density of $D(T)$ and shifting trails $D(\mathbf{t})$ upstream in this gradient until convergence (tight bundles) has been achieved. The method parallelizes well on graphics hardware (GPUs), leading performance increases of over one magnitude order as compared to all earlier general-graph bundling methods. KDEEB also opened the area of so-called *image-based* bundling methods, where B is implemented via image processing operations, as opposed to purely geometry techniques as in earlier methods (see further Sec. 4.1.2.2).

Skeleton-Based Edge Bundling (SBEB, Fig. 4e) is another image-based method [EHP*11]. SBEB computes a kernel density estimation of the drawing $D(T)$ using GPU texture splatting. Next, the KDE map is segmented to obtain a morphologically dilated version $D_{dil}(T)$ of $D(T)$ [Har94]. Following the observation that bundles should gather trails towards their local center, 2D medial axes, or skeletons [SP09], of $D_{dil}(T)$ are next computed, and trails in $B(T)$ are attracted to the skeleton. SBEB yields smooth and highly-branching bundles, following known properties of 2D medial axes. However, the method relies on a pre-clustering of similar trails in $B(T)$, which is a relatively expensive and parameter-sensitive step.

Directed trails: Following the need outlined by DEB (Sec. 3.2.1.2), directional bundling is also considered, especially for trails capturing vehicle motion, where it is very important to distinguish opposite flows. Attribute-Driven Edge Bundling (ADEB, Fig. 4m) extends KDEEB by taking edge attributes (direction and/or time) in the definition of the compatibility κ, while keeping KDEEB's high speed [PHT15]. Histogram Edge Bundling (HistEB, Fig. 4f) also performs directional bundling, but computes κ by binning the tangent direction space of trails $\mathbf{t}$ and applying KDEEB to each bin separately [Mou15]. CUDA Universal Bundling (CUBu, Fig. 4h,k,l) further enhances KDEEB and ADEB by proposing a far more efficient density estimation, also implemented on the GPU, making it possible for the first time to bundle sets of up to a million trails at interactive framerates [vdZCT16]. Separately, Texture Edge Bundling (TEB) proposes a GPU-based implementation making heavy use of texture synthesis and processing, optimized for web access [WYY15]. Lastly, Fast Fourier Transform Edge Bundling (FFTEB, Fig. 4i,n) refines CUBu to use the FFT for an even more efficient KDE, allowing one to bundle trail-sets whose sampling does not fit in the GPU memory [LHT17]. With this, bundling can now effectively handle 'big data' collections.

Parallel Coordinate Plots: Parallel coordinate plots (PCPs) [Ins09] can be seen as trail-sets Given a n-dimensional dataset of N observations $\mathbf{x}_i$, each trail $\mathbf{t}_i$ has n control points representing the n attribute values of observation $\mathbf{x}_i$. For large N, PCPs suffer from the same clutter as large straight-line graph or trail-set drawings. Bundling can effectively reduce this and also help one to find clusters of similar observations easier [HLK*12, HvW10]. For this, Zhou *et al.* used a method similar to FDEB (Sec. 3.2.1.2), where PCP trail compatibility (κ, Eqn. 1) considers PCP trail distance and angular similarity [ZYQ*08]. In contrast to FDEB, the bundling solution is not computed by iterative gradient descent of the cost function κ, but by linear programming. Illustrative parallel coordinates (IPCs) bundle PCPs by first clustering $D(T)$ via k-means, and next bundle the trails $\mathbf{t}_i$ in the same cluster as B-splines [PT97] which use shared control points computed from the cluster's average trail [MM08]. More recently, Palmas *et al.* bundle PCPs by explicitly computing averages for each PCP axis and next linking these, for neighbor axes, by compact tubes, whose widths indicate the number of bundled lines [PBO*14]. This produces a highly summarized, but largely clutter-free, visualization, which reminds the code flow metaphor used in [TA08] (see also Sec. 3.2.1.1). A variant of this technique is also available for Continuous Parallel Coordinates (CPCs), where the trail density is drawn instead of individual trails [PW16], much like it was used elsewhere to show vessel routes [SWvdW*11].

Bundling is not the only way to reduce clutter and enhance PCP readability. PCP trails can be rendered as smooth cubic curves, allowing one to trace them end-to-end easier [GK03]. Alternatively, parametric transformations can be used to yield similar smooth curves [MW02]. While yielding smooth curves similar to bundles, these techniques do not explicitly aim at grouping PCP trails following the bundling definition in Eqn. 1.

3.3.1.2. 3D trail-sets 3D trail sets are spatial curves embedded in $\mathbb{R}^3$. A good example are Diffusion Tensor Imaging (DTI) trails, or *tracts*, that show anatomical brain structures [ALLF07]. DTI tracts form a highly complex 3D structure consisting of multiply intersecting surfaces, fanning out into many-direction fibers, so *abstraction* of such visualizations is highly needed to reveal the brain's white matter structure [TWHW07]. Originally done by fiber clustering [MVvW05], bundling offers advantages in terms of a finer-level simplification control. For this, Everts *et al.* [EBB*15] adapt the compatibility κ and iterative bundling process proposed by FDEB (Sec. 3.2.1.2) to include nearest-neighbor distance $\min_{\mathbf{x}_i \in \mathbf{t}_i, \mathbf{x}_j \in \mathbf{t}_j} \|\mathbf{x}_i - \mathbf{x}_j\|$ between two tracts $\mathbf{t}_i$ and $\mathbf{t}_j$ in $D(T)$. A similar approach is illustrated in [Tel15] (Ch. 7), where KDEEB (Sec. 3.3.1.1) is used to bundle DTI fibers by constraining motion to follow the DTI field anisotropy. Trail bundling is also used to bundle 3D streamlines for multiscale flow visualization [YWSC12] and 3D geographical routes over a height map [TP15]. A salient difference of this use-case as compared to most other bundling cases discussed here is that a streamline's endpoints are not fixed, as they do not represent important spatial information that needs to be preserved in

the visualization. Compared to 3D graph bundling, additional care is taken by all 3D trail approaches above to constrain trail displacement so as to respect, as much as possible, the original data – zones of high linear or planar anisotropy for DTI fields, tancency to the flow for vector fields, and trail fit to a 3D landscape for geographical routes, respectively.

Besides true 3D bundling, methods also exist that bundle trails defined over a curved surface. WR extends the 2D bundling proposed in [LBA10b] to cover the case of 3D trails representing flight routes [LBA10a]. The bundles are displayed over the surface of the Earth, offering a better estimation of distances than when a 2D cartographic projection is used. A good review of the challenges of 3D geospatial trail visualization, including a discussion of bundling, is given in [BTD12]. 3D trail bundling is also prominently featured in a review of visualization methods for climate networks [NBD*15].

3.3.2. Dynamic Trail Sets

Dynamic trail sets parallel the concept of streaming graphs (Sec. 3.2.2.1) but add explicit spatial information along trails. Examples are paths of vehicles over a given space-time range, such as ships [SWvdW*11, SHVDWVW16], airplanes [HTC09, EHP*11, HEF*14], or eye tracks [PHT15]. Compared to static trail bundling, one wants here to show how the set of trails being live at a given time moment changes in time. For this, Hurter *et al.* extend KDEEB (Sec. 3.3.1.1) to use the sliding time-window technique of StreamEB (Sec. 3.2.2.1) [HET13]. Compared to StreamEB, this approach is much faster, as it continuously bundles, in a loop, the trails present in the current time-window, rather than restating bundling from scratch each time the window is shifted. The method is extended to cover use-cases considering eye tracking trails, and animation to show point-like textures flowing in the direction of the trails $\mathbf{t}_i$, to indicate direction [HEF*14, KvdZT14].

4. Bundling Framework

Given their large number and diversity, comparing all bundling methods listed in Sec. 3 from a technical viewpoint is clearly challenging. Yet, this is necessary for developers interested to understand how such methods work, and for researchers who aim to extend existing methods. To help this, we propose next a generic bundling framework. Our framework, which is based on the notations introduced in Sec. 2, describes the main steps and technical choices of most bundling methods, and outlines specific advantages and limitations of such choices. The framework has four steps (see Fig. 5). It starts with either a graph G or a trail-set T. In the former case, a graph layout method is used to construct a graph drawing; in the latter case, the trail-set is its own drawing, as explained in Sec. 2. The bundling proper thus starts having a path-set drawing $D(P)$ as input. Next, the similarity functions κ and δ must be defined (Sec. 4.1). Having these, a bundling operator B can be defined following Eqn. 1 and applied on $D(P)$ to yield the bundled drawing $B(D(P))$, as discussed in Sec. 4.2. Finally, this drawing is visually explored (Sec. 4.3).

4.1. Similarity Definition

To bundle a path-set, we need, first and foremost, to specify which edges are compatible (to be bundled), and how much to bundle.

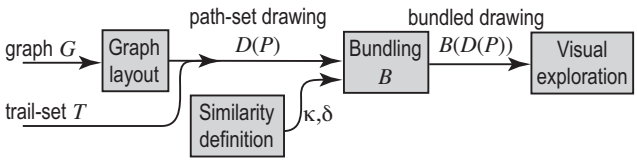

Figure 5: *Bundling framework steps.*

These steps are achieved by defining the functions κ and δ, respectively (see Eqn. 1 and related text). As explained in Sec. 2, κ can account for similarities in the *data* (*e.g.* graph structure and trail attributes), and must in any case account for similarities in the *drawing*. These two components of κ are discussed next.

4.1.1. Data-based similarities

Given two paths $\mathbf{p}_i$ and $\mathbf{p}_j$, data-based similarity quantifies the difference between $\mathbf{p}_i$ and $\mathbf{p}_j$ over the space of all possible paths. This can be done in several ways, as follows.

Structure-based: In contrast to trail-sets, graphs have additional structure (topology). This structure can be used to define the similarity of edges. For instance, hierarchical bundling methods (Sec. 3.2.1.1) use the explicit structure of the graph's hierarchy to define edge similarity in terms of distance, in this hierarchy, of the nodes of the two edges $\mathbf{p}_i$ and $\mathbf{p}_j$. Intuitively put, edges that start and end at nodes which are close in the hierarchy, are deemed to be close, and thus bundled. Other methods extract such structural data from the input graph, and use it to define edge similarity. These include flow maps which extract a tree (spanning [PXY*05], Steiner [BSV11] or spiral [VBS11]); and confluent drawings, which compute a power graph to find out which edges in the original graph are strongly related [DMM*14, DHRMM13, BRH*16]. Separately, TGI-EB [NHE11] adds new graph-theoretic metrics to define edge compatibility, such as the centrality of edges in a graph [WF94, vHW08], or the fact that edges belong to the same cluster in a simplified version of the graph. Similarly, DEB [SHH11] proposes a connectivity term based on the graph-theoretic distance between the endpoints of two edges. A similar term, called trace compatibility, appears in StreamEB [NEH12]. Finally, StreamEB introduces a so-called ego compatibility: For a node $\mathbf{v} \in V$, $ego(\mathbf{v})$ is defined as a small neighborhood in G of $\mathbf{v}$, nodes and edges included. For an edge $\mathbf{e} = (\mathbf{v}_i, \mathbf{v}_j)$, $ego(\mathbf{e}) = ego(\mathbf{v}_i \cup \mathbf{v}_j)$. Next, the ego compatibility of two edges $\mathbf{e}_i$ and $\mathbf{e}_j$ is a decreasing function of $ego(\mathbf{e}_i) \cap ego(\mathbf{e}_j)$. This prevents bundling edges which link weakly-connected communities in G.

Structure-based similarity serves two independent goals. First, it allows specifying which edges are compatible from an application perspective, and thus may end in the same bundle, as outlined above. Separately, it allows constructing groups of possibly compatible edges which are next examined for actual bundling. This essentially reduces the bundling complexity from analyzing all edge pairs in G to only analyzing pairs within a cluster. Clustering can be done using k cores [BE05], see [NHE11]; bottom-up hierarchical aggregation [dHINM04], see [TE10, EHP*11, ZWHK16]; k means, see [PT97]; or kd-trees, see [GHNS11].

Attribute based: When edge or trail data attributes are available, these can be used to compute additional similarity terms. This is important when one does not want to bundle edges of different types together. Good examples are software engineering graphs which contain edges of various kinds, such as inheritance, call, or dependency [CZH*08, RVET14, DT14]; and airline trail-sets which contain trails representing flights with different IDs or types [HCGT14]. Each edge type has a different meaning, so edges of different types should arguably not be bundled together. To model this, a simple similarity function based on the edge's categorical attribute 'type' can be used [RVET14, TE10]. Edges can also have quantitative attributes, such as duration and starting-time for calls in program traces [TDT13, KTD13], or timestamps for eye tracks [PHT15] which can be used to define similarities. Separately, for dynamic path-sets, time provides an additional compatibility factor, used to bundle only paths that fall within the same (usually small) time-window [NEH12, HEF*14, KvdZT14]. This makes bundling computationally and visually scalable to very large streaming datasets.

4.1.2. Drawing-based similarities

While incorporating *data*-based similarity in the compatibility κ is optional, all bundling methods must account for the *spatial* similarity of paths to bundle. This can be done by using either geometric methods (Sec. 4.1.2.1) or image-based methods (Sec. 4.1.2.2).

4.1.2.1. Geometric-based similarity: These methods use a piecewise-polygonal sampled representation of the path drawings $D(\mathbf{p})$ to evaluate their similarity. As such, various aspects of a curve can be taken into account, as follows. Let $\mathbf{d}_i \subset \mathbb{R}^d$ and $\mathbf{d}_j \subset \mathbb{R}^d$ be two such polyline path drawings. Similarity δ typically is a product (or, in [NEH12], a weighted sum) of the following terms:

Distance: First, the distance between $\mathbf{d}_i$ and $\mathbf{d}_j$ is considered, as one does not want to bundle far-away path drawings (see Eqn. 1 and related text). In turn, this distance can be computed as the sum of distances of the closest endpoints of $\mathbf{d}_i$ and $\mathbf{d}_j$ [GHNS11]; or the distance between the midpoints of $\mathbf{d}_i$ and $\mathbf{d}_j$ (see FDEB [HVW09] and its refinements [NEH12, NHE11, BSL*14, ZWHK16], and WR [LBA10b]). However simple, such similarities cannot be used for trail bundling, as trails are usualy not straight lines.

Angle: The absolute value of the cosine of the angle formed by the straight-line segments $\mathbf{d}_i$ and $\mathbf{d}_j$ is used to prevent bundling of orthogonal edges [HVW09, NHE11, NEH12]. This idea is refined by DEB [SHH11] to consider the actual signed angle, which allows directional bundling.

Scale: The length difference of the two segments $\mathbf{d}_i$ and $\mathbf{d}_j$ is used to prevent bundling very long and very short edges, and therefore minimizes the deformation of the latter, which are best drawn as (almost) straight [HVW09]. The same effect is achieved by CUBu [vdZCT16] by limiting the deformation factor of paths based on their length.

Visibility: FDEB proposes that edge-pairs which would create a skewed parallellogram should be bundled less than those which create a rectangle [HVW09]. This produces smoother, but less tight, bundles, and as such this similarity term has not been widely adopted by later methods.

Area and ink: Compatible edges can also be defined implicitly as those edges which, when bundled, optimize a global cost function describing the quality of the bundling $B(D(T))$. For instance, Gansner *et al.* [GK06] recursively cluster paths bottom-up as long as the resulting bundling improves the usage of the drawing area. Similarly, MINGLE [GHNS11] groups edges as long as the amount of ink being used to draw $B(D(T))$ is lower than the amount used to draw the unbundled $D(T)$ (see also Sec. 3.2.1.2). Both methods are greedy, so they cannot guarantee a global minimum.

4.1.2.2. Image-based similarity: Apart from assessing which paths can be bundled (κ), one needs to make sure that bundled paths are closer than the unbundled ones. To measure bundled path distance, we need thus to define the function δ introduced in Eqn. 1. As stated there, δ is typically a form of Hausdorff distance: Given two sampled paths $\mathbf{d}_i = (\mathbf{x}_i^k)$ and $\mathbf{d}_j = (\mathbf{x}_j^k)$ which are compatible, *i.e.* make sense to be bundled, we have

$$\delta(\mathbf{d}_i,\mathbf{d}_j) = \sum_{\mathbf{x}_i^k \in \mathbf{d}_i} \min_{\mathbf{x}_j^k \in \mathbf{d}_j} \|\mathbf{x}_i^k - \mathbf{x}_j^k\| + \sum_{\mathbf{x}_j^k \in \mathbf{d}_j} \min_{\mathbf{x}_i^k \in \mathbf{d}_i} \|\mathbf{x}_i^k - \mathbf{x}_j^k\|. \quad (2)$$

If we do not know upfront which paths are compatible with each other, we thus need to find, for each sample point of a path, the closest sample point of *all* other paths. While this can be accelerated by using various spatial structures such as *kd*-trees [GHNS11], quadtrees [LBA10b], this process is very expensive.

Image-based methods address the evaluation of Eqn. 2 *and* the finding of compatible paths (evaluation of κ) by using imaging and/or GPU methods. The first method in this class is SBEB [EHP*11] (Sec. 3.3.1.1). Here, $\delta(\mathbf{d}_i,\mathbf{d}_j)$ is evaluated as the sum $\delta(\mathbf{d}_i,S) + \delta(\mathbf{d}_j,S)$ where S is the medial axis of the group of compatible paths that $\mathbf{d}_i$ and $\mathbf{d}_j$ are part of. Path-to-skeleton distances are evaluated as

$$\delta(\mathbf{d}_i,S) = \sum_{\mathbf{x}_i^k \in \mathbf{d}_i} DT_S(\mathbf{x}_i^k) \quad (3)$$

where $DT_S : \mathbb{Z}^2 \to \mathbb{R}^+$ is a pixel map (image) capturing the so-called Euclidean distance transform of the shape S [FCTB08]. Since there are far fewer clusters (thus skeletons) than paths in $D(T)$, and Eqn. 3 can be efficiently implemented using fast-marching methods [TvW02, Set02], but also GPU methods [CTMT10], δ can be efficiently computed.

An alternative is proposed by kernel density estimation (KDE) methods. As stated in Sec. 3.3.1.1, a path density map $\rho : \mathbb{Z}^2 \to \mathbb{R}^+$ is computed from $D(T)$. Next, path sampling points $\mathbf{x}_i^j$ are moved upstream in $\nabla\rho$. This *implicitly* minimizes δ by gradient ascent without having to explicitly find closest sample point-pairs. The basic KDEEB method [HET12] supports only undirected bundling. KDEEB was enhanced by weighting the computation of ρ by various attributes like path directions and data attributes [PHT15, Mou15, vdZCT16, LHT17]. Moreover, computation speed-ups of ρ are proposed: CUBu improves with respect to KDEEB by using a gathering, rather than scattering, convolution strategy, which parallelizes much more efficiently on GPUs [vdZCT16]; and FFTEB improves on CUBu by up to an order of magnitude by executing the convolution in frequency space (using the properties of the Fast Fourier Transform) rather than in image space.

4.2. Bundling Operator Definition

Having the functions δ and κ implemented as described in Sec. 4.1, one can now define the core of a bundling method – the bundling operator B, *i.e.*, propose an implementation of Eqn. 1. We classify existing bundling methods in *explicit* and *implicit* ones, as follows.

We discuss next *implicit* methods which define B recursively by an iterative optimization process. Methods in this class are all force-directed techniques working in geometry space $I \subset \mathbb{R}^2$ and image-based techniques working in discrete image space $I \subset \mathbb{Z}^2$.

4.2.1. Explicit Methods

Explicit methods compute an intermediary structure I from $D(T)$ and define B based on I. The structure I is computed typically only once, after which paths are routed according to it. As such, explicit methods are in general fast and predictable, but less flexible in terms of bundling control.

As already outlined, different types of structured I exist. Hierarchical methods use the explicit hierarchy provided by a compound graph [Hol06, CZH*08, WL07, GBE08, CZB11] or a tree extracted from a DAG [PNK10]. General-graph methods extract spanning trees [PXY*05], Steiner trees [BSV11], spiral trees [VBS11], Voronoi diagrams [LBA10b, CZQ*08], and Delaunay triangulations [QZW06, CZQ*08]. Once I exists, edges are simply routed along it, using various forms of smooth curves, *e.g.* B-splines [Hol06, WL07, GBE08, CZB11, PT97]; Bézier splines [BW98]; NURBS [QZW06]; and cubic curves [GK03].

4.2.2. Implicit Methods

Implicit methods work in a self-organizing way, without computing an explicit control structure I upfront. As such, they avoid the problems of explicit methods due to computation of a suboptimal I – bundling can 'self-correct' itself during the process. These methods work typically in an iterative way, similar to optimization processes which aim to find the extremum of a global cost function.

Force-based techniques are the first, and best known, class [HVW09,NHE11,NEH12,SHH11,BSL*14]. They compute the gradient of a cost function of the form

$$C(D(T)) = \sum_{\mathbf{d}_i \in D(T), \mathbf{d}_j \in D(T), \kappa(\mathbf{d}_i,\mathbf{d}_j) < \kappa_{min}} \delta(\mathbf{d}_i, \mathbf{d}_j), \qquad (4)$$

with δ given by Eqn. 2), and iteratively shift, or advect, sample points $\mathbf{x}_i^j$ to minimize C, *i.e.*, yield tight bundles. This idea is very similar to force-based graph layouts [KKH89], albeit using a different cost function. These methods are much slower than implicit ones, as the cost C (Eqn.4) has to be recomputed at each iteration, and computation of δ is expensive, as explained in Sec. 4.1.2.2. Using spatial search or clustering structures (Sec. 4.1.2) does not alleviate this, as path sampling points are continuously changing over iterations, so such structures would need repeated re-initialization.

Image-based methods are very similar to force-based methods, with the key difference that the cost computation (Eqn. 4) is much lower due to the fast GPU-based evaluation of δ (Sec. 4.1.2.2). Various update ideas are proposed here: Sample points are shifted upstream in the gradient of the skeleton's distance transform ∇DT_S by SBEB; and in the gradient of the density map $\nabla\rho$ by KDEEB, ADEB, CUBu, and FFTEB, respectively. Additionally, recent implicit image-based methods (CUBu,FFTEB) implement B fully on the GPU (sampling paths $\mathbf{d}_i$ to points $\mathbf{x}_i^j$; computation of the density map ρ; shifting sample points in $\nabla\rho$; and rendering the final results). This yields speed-ups of up to two order of magnitude with respect to earlier image-based methods (KDEEB,ADEB).

Implicit methods add the extra capability of producing *multiscale* bundling in a scale space sense [Koe84]: Given the κ_{min} constraint in Eqn. 4, only edges closer than a certain distance δ_{max} are considered to interact. Setting δ_{max} to small values produces fine-grained bundles, where the amount of deformation of any path point is lower than δ_{max}. Setting δ_{max} to large valus produces coarse bundles which exhibit more deformation but reduce clutter more too. Using a δ_{max} bound also massively reduces the costs of computing the all-pair path similarities $\delta(\mathbf{d}_i,\mathbf{d}_j)$ to a small subset. Image-based methods implement multiscale bundling efficiently, as κ_{max} is essentially controlled by the KDE kernel radius R (Sec. 4.1.2.2). Multiscale bundling is much harder to achieve by explicit methods, as these do not provide a *continuous* parameter to control the scale of the simplification.

4.3. Bundling Visualization

After bundling B computes a bundled drawing $B(D(T))$ of a path-set $D(T)$, several mechanisms are available for visually presenting $B(D(T))$. These serve multiple purposes: display the bundling, enhance important structural elements thereof, add attribute data atop it, further reduce visual clutter, simplify the displayed image, and ger details on demand. Visualization techniques for bundled data can be grouped into the following classes: blending, data color mapping, shading, smoothing and deformation, animation, and interaction, as described next.

4.3.1. Blending

As outlined in Sec. 2, bundling trades clutter for overdraw. Multiple paths (or path fragments) of $D(P)$ become overlapping in $B(D(P))$. However, many tasks require assessing the connection strength, or number of paths, between groups of nodes or endpoints in P. To support this, blending can be used: $B(D(P))$ is drawn using alpha blending or variations thereof. This renders bundles containing many paths more opaque than sparse ones, thereby attracting the user's attention more [Hol06]. Essentially, this maps the local bundle density ρ (Sec. 4.1.2.2) to opacity, color, or shading, an idea pioneerd first by Van Liere and De Leeuw for drawing unbundled straight-line graphs [vLdL03], and used next in many other contexts [SWvdW*11,SWvdWvW11,LH11]. All examples in Fig. 4 are rendered using density-based blending. Figure 6 shows blending for a detail of HEB [Hol06].

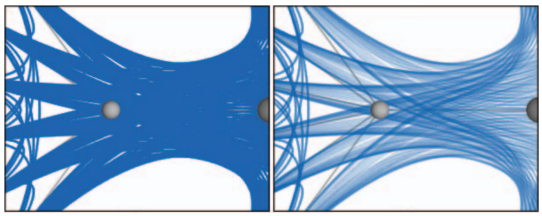

Figure 6: *Usage of blending [Hol06]. Left: Drawing $B(D(P))$ without blending. Right: Same drawing, with blending.*

While conceptually simple, blending requires some care: Simply drawing the bundled paths $B(D(P))$ with OpenGL alpha blending will not work, as the typical resolution of an OpenGL alpha channel is 8 bits, which can capture only 256 different values. As many more paths can overlap, the result can easily be fully saturated (opaque). The solution is to map the local bundled-path density ρ, computed using accurate floating-point operations (Sec. 4.1.2.2), to the 8-bit alpha channel [vdZCT16]. Separately, short paths can be easily obscured by long ones. The solution to this is to set path opacity inversely proportional to path length [Hol06, vdZCT16]. Additionally, CUBu [vdZCT16] draws short paths atop longer ones, so that the former get a fairer chance to stand out in the final image (Fig. 4k,l). However, this requires sorting paths by length, which can be costly ($O(NlogN)$ for N paths in $D(P)$).

4.3.2. Data color mapping

Bundled path colors can map various path attributes, *i.e.* map data from P to the color visual variable in $B(D(P))$ [Ber83]. Data includes path spatial density (extends the blending idea in Sec. 4.3.1 to multiple-hue colormaps); path start-to-end direction using a color gradient (Fig. 2a [Hol06], Fig. 11b [CZH*08]) following earlier methods used for the same goal when drawing straight-line graphs [Die08]; cluster containing the path (Fig. 2c and 8 [EHP*11]; see also [MM08]); type of path (Fig. 11a [RVET14]); path spatial orientation, before bundling, using an angle-to-color map (Fig. 2k-l [vdZCT16] and 4m-n [LHT17], Fig. 14a-c [PHT15] and 14d [KvdZT14]); and local height along spatial airline trails [HCGT14]. Path directions are rarely indicated by arrows [RVT11, CZH*08], as these clutter quite easily and become hard to discern when many paths end at the same (small) node, as also discussed in [HIVWF11].

A serious issue here is color blending: When paths overlap, what should be the resulting color? This is especially hard to solve if color maps categorical attributes, which cannot be averaged [TE10]. To date, no conclusive answer exists to this issue. This is further discussed in [HEF*14, KvdZT14, vdZCT16].

4.3.3. Shading

Although bundling reduces the visual complexity of a drawing, it is still hard to discern salient (important) connections, even when using alpha blending (Sec. 4.3.1). To emphasize the spatial extent of a bundle (group of spatially close paths in $B(D(T))$), *pseudo-shading* can be used, whih was pioneered by Image-Based Edge Bundles (IBEB) [TE10], see Fig. 2c. For this, IBEB, but also WR, SBEB, and CUBu create a false height map increasing parabolically from the borders of a bundle to its center, and then shade this by classical Phong shading, yielding similar effects to the well-known cushion treemaps [VWvdW99]. This allows better visual separation of crossing bundles, which show up as shaded tubes, based on the luminance variation from the tube borders to their centers, given a 3D stacking

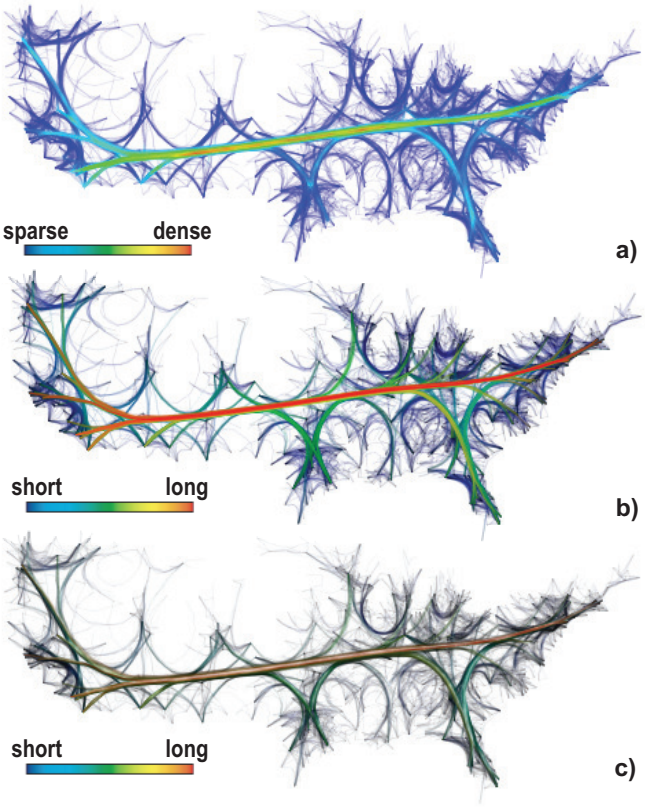

Figure 7: *Color mapping and shading using CUBu [vdZCT16].*

effect (see Figs. 2c and 14a,b). To do this, IBEB and SBEB need to explicitly cluster the bundled paths, which, as discussed in Sec. 4, is delicate. The same explicit path clustering is used by the geometric-shaded bundles in [TDT13] (Fig. 11c). WR and CUBu do the path grouping implicitly, which makes them the methods of choice. A different approach is to use bump mapping to highlight high-density bundles based on the slope of the density map ρ (Sec. 4.1.2.2). While simple to implement, this yields less well-separated bundles, and creates artificial visual discontinuities in their middle [HET12]. Shading and color mapping can be both combined (Fig. 7): We can use colors to map either local edge density (Fig. 7a) or edge length (Fig. 7b), thereby emphasizing dense regions, respectively long edges, respectively. Adding shading to the latter combines the effects – salient shaded tubes show dense regions, while color maps edge length, respectively.

4.3.4. Smoothing and deformation

Explicit bundling techniques control well the local curvature of resulting bundles, as they route the bundled paths along a precomputed global structure I (Sec. 4.2.1). Implicit techniques (Sec. 4.2.2) have less control, so they postprocess $B(D(T))$ to achieve smoothness. The by far most common way for this is to apply 1D Laplacian smoothing [HDZ05] on the bundled paths, as introduced in [HVW09]. This technique is simple, fast ($O(N)$ for a drawing $B(D(T))$ sampled with N points), and easily controllable.

Bundling can also include *deformation* constraints. For instance, SBEB [EHP*11] can route bundles to avoid landmarks in the drawing space $\mathbb{Z}^2$, by using essentially the same principle as 'dust & magnets' [YMSJ06], but with repulsion instead of attraction (Fig. 8a). The technique can be easily integrated in any image-based implicit bundling method, either during the iterative bundling or as a postprocessing step. WR [LBA10b] and TGI-EB [NHE11] can include bundle orientation constraints, yielding results that follow the so-called 'metro map' drawing style (Fig. 8b,c). Other deformation mechanisms support interactive exploration (see Sec. 4.3.6 further).

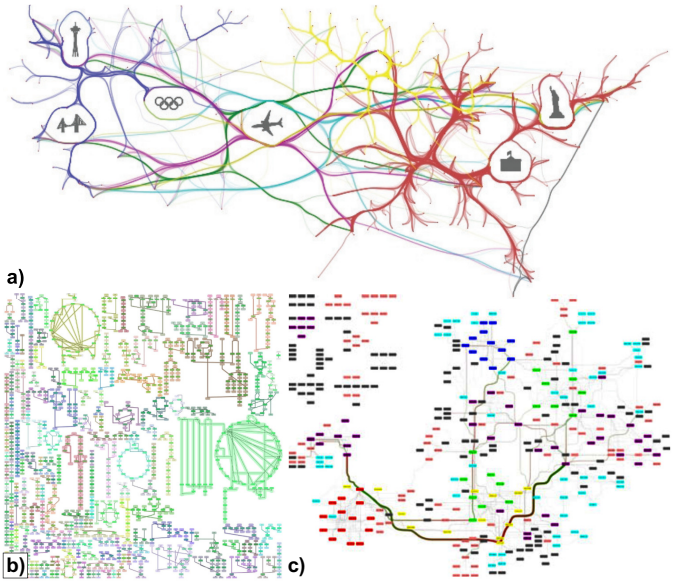

Figure 8: *Bundle deformation to (a) avoid landmarks [HET12] and (b,c) favor orthogonal drawing [LDB11, NHE11].*

4.3.5. Animation

Animation can be used for two main goals. First, it can convey path directions in a trail-set. For this, small point-like textures, also called particle systems, encode the local path direction [HIVWF11] and are drawn at regular intervals along the bundled paths and temporally shifted in the path direction [HEF*14, KvdZT14]. This yields a moving effect along bundles (several such textures coherently move in the same direction) that conveys a bundle's direction. This design is related to older methods for visualizing flow fields [vW02]. Figure 9a shows an example using tapered arrow-like textures [HIVWF11] that encode path directions. However, as for data color-mapping (Sec. 4.3.2), animation has the issue that, if a bundle contains paths having different directions, the result can show random noise patterns from which we cannot discern the fraction of paths going one way *vs* the opposite way. Note that this is not a problem for the underlying technique (IBFV [vW02]) since, for that case, a 2D vector field has a *single* data value per location. Secondly, animation can show changes in a time-dependent path-set $P(t)$, by interpolating the bundled trails either during keyframes (for sequence graphs, see Sec. 3.2.2.2) or by continuously morphing a trail-set to account for incoming and leaving trails (for streaming graphs, see Sec. 3.2.2.1). Figure 9b shows three frames from a bundled sequence-graph depicting code clones between software components in three revisions of a software system [HET13]. Red edges show newly appearing clone relations, which are bad for system maintenance, and thus should be spotted and removed.

4.3.6. Interaction

Interaction is a key technique to the analysis of bundled drawings. Following [BM13], interaction allows answering *how*-type questions; and enables bundled drawing $B(D(P))$ to support *select*, *navigate*, *filter* and *arrange* tasks. Several types of interaction exist in this context, as follows:

Relaxation: Linear interpolation between the input drawing $D(P)$ and its bundling $B(D(P))$ is used by virtually all bundling methods, starting with [Hol06], to control the bundling 'tightness'. Interactively changing the interpolation parameter to-and-fro allows users to visually link paths in $D(P)$ and $B(D(P))$ and thus resolve (parts of) the bundling ambiguities created by overlap. Similar techniques are used to link items in other visualizations [HTCT14]. Figure 10a-d shows four frames from a relaxation process;

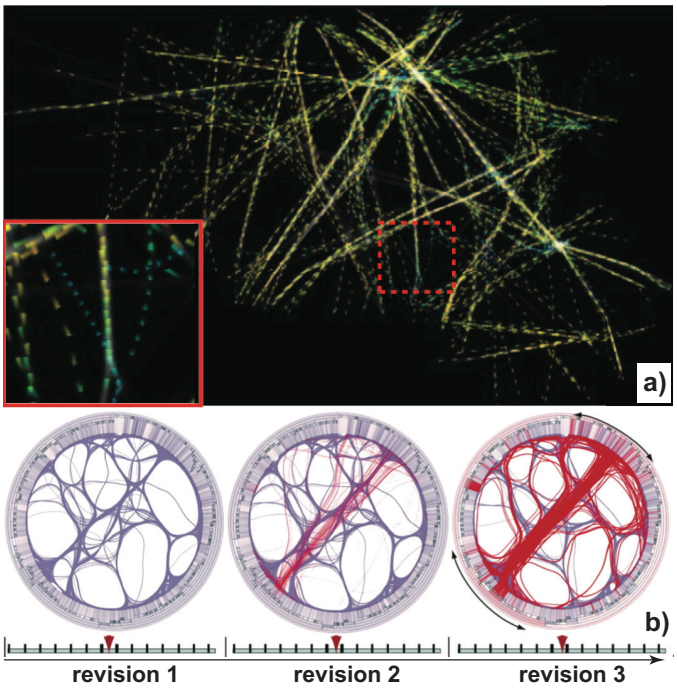

Figure 9: *Animation techniques: (a) Textures showing bundle directions [HEF*14]. (b) Sequence graphs [HET13].*

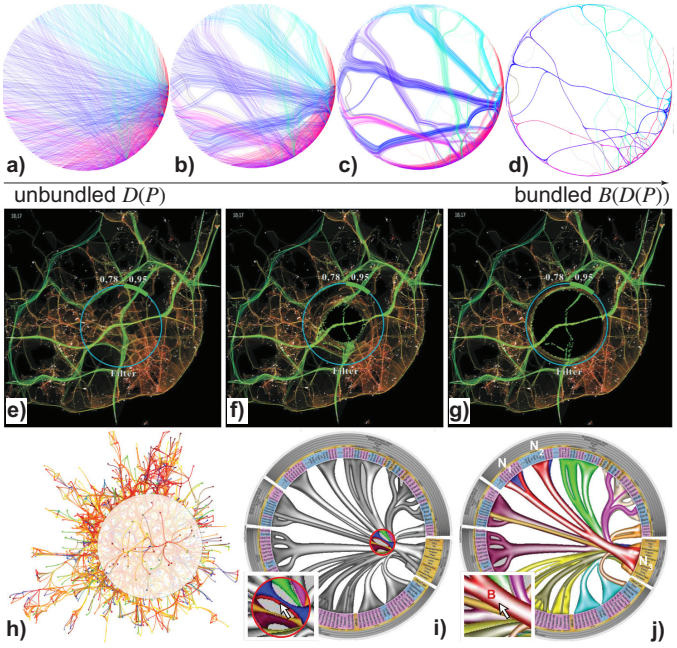

Figure 10: *Interacting with bundled drawings: (a-d) Global bundle relaxation [HET12] and (e-g) local relaxation [HET11]. (h) Magic lens [LAM10]. (i,j) Digging lens [TE10].*

Lenses: Bundling can be applied (or prevented) locally on $D(P)$. This way, one can combine views of the unbundled $D(P)$ with the bundled $B(D(P))$. This is essentially a local relaxation variant. Figure 10e-g shows three frames from such a local lens, used to get detail on a road-traffic dataset [HET11]. Lambert *et al.* propose additional variants, such as fisheye and bring & go techniques [TAvHS06], using GPU-computed splines for interaction fluidity [LAM10] (Fig. 10h). Techniques such as edge plucking [WC07] can be easily added. The digging lens [TE10] alleviates occlusion in shaded-tube bundle renderings (Sec. 4.3.3): Bundles are thinned close to the interaction focus using image-processing techniques (Fig. 10i) so that one can see, and bring to front, occluded bundles from beneath (Fig. 10j).
Brushing: Brushing bundles can reveal aggregated attributes of the overlapping paths in $B(D(P))$, thereby alleviating the problem outlined in Sec. 4.3.2 [CZH*08, RVET14];
Navigation: HEB-like methods (Sec. 3.2.1.1) can support navigation of the hierarchy T by interactively collapsing (clustering) or opening (refining) nodes in T. Such operations trigger the display of a different set of bundled edges [CZH*08, RVET14, Han13].

Riche *et al.* define interactions and design guideline to support node and edge manipulations in bundled drawings [RDLC12]. Separately, Luo *et al.* propose interactive techniques to decrease path ovelapping issues [LLCM12].

5. Task Support

Using bundling fits very well into Shneiderman's "overview first, zoom and filter, then details-on-demand" [Shn96] metaphor. The key task a bundled drawing $B(D(P))$ supports is to present an *overview* of $D(P)$ – thus, implicitly of P too – which trades off clutter for overdraw. If users find interesting patterns in it, they can next *zoom* and *filter* on such patterns, and next get *details on demand* on them using the interaction techniques discussed in Sec. 4.3.6. We analyze this further by first discussing how bundling reduces clutter (Sec. 5.1). Next, we detail the tasks supported by bundled drawings (Sec. 5.2).

5.1. Bundling Clutter Reduction Taxonomy

To understand how (well) bundling reduces clutter, we analyze it following Ellis and Dix's clutter reduction taxonomy [ED07]. From all clutter reduction techniques (see [ED07], Tab. 1), bundling uses path ([ED07] refer to 'point/line'; as our input is a (possibly curved) path drawing $D(P)$, we use the term path) displacement, clustering, and opacity techniques. Indeed, B displaces paths to create path-clusters and renders them using opacity. Separately, Tab. 3 in [ED07] outlines eight key benefits given by clutter reduction. Table 2, columns 2…4 lists them, showing how opacity, clustering and displacement contribute to each. Column 5 in Tab. 2 shows our (arguably subjective) view on how well bundling does this, as detailed below.

	opacity	clustering	path displacement	**bundling**
avoids overlap	partly	possibly	✓	partly
keeps spatial information	✓	partly	✗	partly
can be localized	✓	✗	✓	partly
is scalable	✗	✓	✗	✓
is adjustable	✓	✓	possibly	✓
can show path attributes	✗	partly	✓	✓
can discriminate paths	✓	✓	possibly	partly
can see overlap density	✓	possibly	✗	✓

Table 2: *Bundling compared with opacity, clustering, and displacement techniques* vs *yielded clutter-reduction benefits [ED07].*

a) Avoids overlap: Bundling avoids overlap of *coarse-scale* patterns: Take a set of unbundled, but highly-similar, paths $H \subset D(P)$. Following Eqn. 1, their bundling $B(H)$ contains much closer paths (*cf.* the spatial distance δ) than H. Hence, for two such disjoint sets H_i and H_j of $D(P)$, the chance that their bundled versions $B(H_i)$ and $B(H_j)$ overlap is (much) smaller than that of the unbundled sets H_i and H_j to overlap. Bundling yields more overlap of compatible edges (*cf.* the compatibility κ), but less overlap for incompatible ones;
b) Keeps spatial information: B does not change path endpoints, so this type of spatial information is kept. Spatial information

of other path points *is* distorted. However, bundle scale control (Sec. 4.2.2), smoothing (Sec. 4.3.4), and relaxation (Sec. 4.3.6) limit the deformation amount;
c) Can be localized: In general, clutter is locally inversely proportional to the bundling amount: Strongly bundled areas exhibit less clutter than weakly bundled ones, assuming an input $D(P)$ with uniform spatial clutter distribution. Hence, local control of the bundling amount can localize the presence of clutter [HET11];
d) Is scalable: Image-based bundling is clearly scalable to large million-size path-sets (Sec. 4.1.2.2);
e) Is adjustable: Bundles can be widely adjusted in terms of path similarity, tightness, smoothness, shape, obstacle avoidance, and visual appearance (Sec. 4);
f) Can show path attributes: Bundling can show path local density, direction, and several other data attributes, either implicitly (by their presence in the compatibility κ (Sec. 2)) or explicitly (by mapping them to opacity, color, shading, and animation (Sec. 4.3));
g) Can discriminate paths: Directional and confluent bundling techniques do precisely that, favoring different types of paths to discriminate [PNK10, SHH11, LLCM12, ZYC*08, BISP16];
h) Can show overlap density: Virtually all bundling techniques map local bundled-path density to opacity or color to show precisely that (Sec. 4.3.1,4.3.2).

Overall, we see that bundling supports well the intended benefits of clutter reduction mentioned in [ED07]. This helps understanding next which tasks bundling can support, and how much. For instance, [ED07] (Sec. 3) mentions that the 'avoid overlap' benefit supports the ability to see and identify patterns [AS94]; 'being localized' helps examining small details while keeping context.

5.2. Task Taxonomy

We analyze which tasks bundling supports using two well-known taxonomies, as follows.

A) Lee *et al.* [LPP*06]: Bundling focuses on links and clusters. For these data types, bundling supports:

a) Path following: Specific bundling techniques support path following, see Sec. 5.1, point (g);
b) Edge density visualization: Bundling clearly covers this, see Sec. 5.1, point (h);
c) Identify edge clusters: If edge (or, more generally, path) similarity can be captured by a function κ, then bundling does this by default, see its definition (Eqn. 1);
d) Identify strongly connected cliques: This is indeed possible. See Sec. 5.1, point (a), last sentence;
e) Overview: As already explained, B can be seen as a coarsening/simplification operator that keeps *and* enhances the core structure of a drawing $D(P)$;
f) Find patterns: Since B enhances the density distribution of $D(P)$, it follows that patterns which are (vaguely) visible in $D(P)$ will only become clearer in $B(D(P))$. For a formal discussion, see [CM02]; for practical examples, see *e.g.* Sec. 6.2. Yet, the converse is not always true: A $D(P)$ having no salient patterns *can* yield a $B(D(P))$ showing false patterns (see next Sec. 7.2.2);
g) Compare flows: Directional bundling combined with directional coloring supports this task, see *e.g.* Fig. 4k-n.

B) Brehmer and Munzer [BM13]: Following this typology, bundling can:

a) answer the ***why*** part of queries related to the above tasks (A). This is typically done by interaction, see Sec. 4.3.6;
b) *identify*, *compare* and *summarize*: Bundling hepls finding salient connection patterns; comparing path-sets from a sequence or stream (Secs. 3.2.2.2, 3.2.2.1, 3.3.2) and summarizing complex path drawings $D(P)$, as clear from all discussions so far;
c) *present*, *discover* and *enjoy*: Bundling can *e.g.* present big world-wide flight datasets [KvdZT14]; help discover user patterns in eye-track data [PHT15]; and enjoy organic presentations of complex, abstract data (see [Tel15], cover). In this typology, *how* tasks answered by bundling are summed up as *encoding*, *aggregating* and *recording*.

6. Applications

We next illustrate the application of graph and trail bundling in several application areas – software engineering (Sec. 6.1), vehicle trajectory analysis (Sec. 6.2), eye track analysis (Sec. 6.3), multidimensional visualization (Sec. 6.4), and vector and tensor field visualization (Sec. 6.5). For each area, we outline a few relevant use-cases, including the goals to be addressed, examples of bundling results, and outline some limitations.

6.1. Software engineering and data mining

Some of the earliest applications of graph bundling emerged from program comprehension. A key aim of program comprehension is to help developers understand the structure and execution of large programs. This helps various types of maintenance, such as discovering and fixing performance problems and bugs, refactoring the software, and recovering its architecture [CHK*01]. Static and dynamic program mining produces a wealth of data, of which an important component are attributed compound graphs, whose nodes describe software entities (*e.g.* methods, classes, and packages) and edges describe inter-entity relations (*e.g.* call, inherit, data transfer, compilation dependency) [EN08, CZvD*09]. Such graphs can have up to hundreds of thousands of nodes and edges, so displaying them using classical straight-line node-link drawings is not effective [TEHR09]. Drawing software graphs is an important subfield of software visualization, for which good surveys exist [Kos03, Die08]. Bundling produces simplified drawings, where tasks such as finding how groups of related software entities (*e.g.*, in the same package) are connected with other similar groups.

Early applications include the simplified visualization of relatively small state diagrams, based on a DAG layout, such as the bundling method of Pupyrev *et al.* [PNK10] (Fig. 11d, see also Sec. 3.2.1.1). HEB yielded a major breakthrough, allowing tens of thousands of edges to be bundled [HvW08]. The original method (Fig. 2a) was next enhanced to handle graphs of hundreds of thousands of elements by allowing for interactive opening and collapsing of hierarchy nodes and automatic aggregation of children edges [TEHR09, RVET14]. Besides using a radial layout for the graph nodes, treemaps were also used, with edges bundled in 3D (Fig. 2e [WL07, CZB11]). The main advantage here is that treemap cells can be used to show more data, such as software metrics, than the (small) cells in a radial icicle plot. Another extension allowed for the comparison of two code bases [HvW08] or multiple versions of the same code base [TA08] (Figs. 2b,d). The key task here is to find how elements in one code base correspond to the other one, and thus spot structural changes. HEB was also used to visualize code duplication (clones) in a software system [VT14, RVET14] and how these evolve in time [Han13, HEF*14]. This supports the planning of clone removal with minimal impact on system architecture.

HEB-like bundling has also been used to visualize program execution, *e.g.* to compare two or more traces of a program [TDT13] (to detect anomalous behavior) or to visualize the behavior of multi-threaded programs [KTD13] (to find performance problems).

In data mining, Bothorel *et al.* [BSH13] used a general-graph bundling method (KDEEB) to display frequent itemsets. These are arranged on multiple (rather than a single) circular layout, so as to minimize edge lengths. Bundling was used to highlight groups of strongly-connected itemsets (Fig. 12). The method was found to help knowledge engineers in extracting association rules from the itemsets.

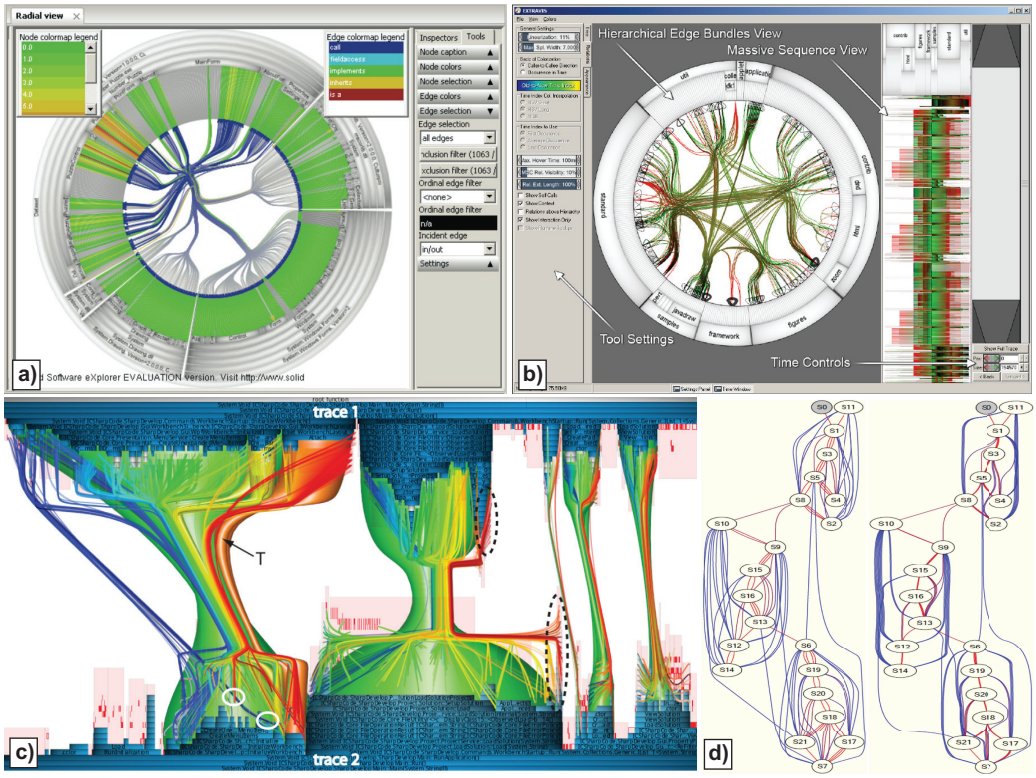

Figure 11: *Bundling in program comprehension. See Sec. 6.1.*

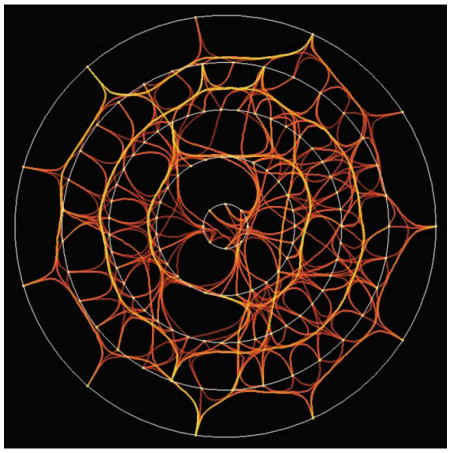

Figure 12: *Nested circular layout and KDEEB bundling for frequent itemsets [BSH13].*

6.2. Vehicle trajectory analysis

Vehicle trajectory analysis is important in many applications such as air-traffic [WHS90], nautical vessel [SWvdW*11, SWvdWvW11, SHVDWVW16], and roadmap [TP15] planning and control. Early on, bundling was used to simplify the depiction of large trail-sets so as to help inferring the main vehicle routes over a country [HVW09]. This use-case, as well as the *US airlines* dataset featured in [HVW09] (Fig. 3), stayed visible in most trail-bundling papers since then [CZQ*08, LBA10b, EHP*11, GHNS11, HET12, vdZCT16, LHT17]. To support source-to-target trail analysis, directional bundling methods were proposed [SHH11, Mou15].

Hurter *et al.* [HEF*14] extend the above to handle streaming trail-sets, obtained from monitoring flights over a given spatial region over a period of time (US territory, 6 days). The obtained animation allows the detection of variations in position and density of the main flight routes depending on the time of the day, and comparing same-time patterns over several days. Recently, this method was extended, by using the fast GPU-based CUBu technique [vdZCT16], to visualize around 800K flights collected from the entire world over June 2013 [KvdZT14], see Fig. 13d. Given the streaming nature of [HEF*14], this approach can be used to visualize large-scale trail-sets that evolve over unbounded time ranges. A more involved use-case is described in [PHT15]. The data consists of 24 hours of flight traffic over France (18K trails). The bundled visualization (Fig. 13a-c) helps air-traffic controllers to cross-reference actual flows with known theoretical air routes (on a global scale) and theoretical approach routes to the Paris airports (locally). This way, problems can be spotted early on, and traffic planning can be adjusted next.

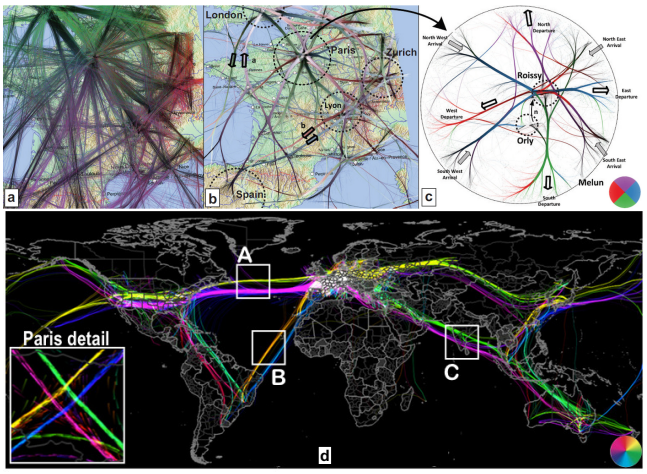

Figure 13: *Aircraft trail analysis. (a-c) Raw trails, directional bundling over France, and zoom-in over Paris area [PHT15]. (d) Worldwide flights [KvdZT14].*

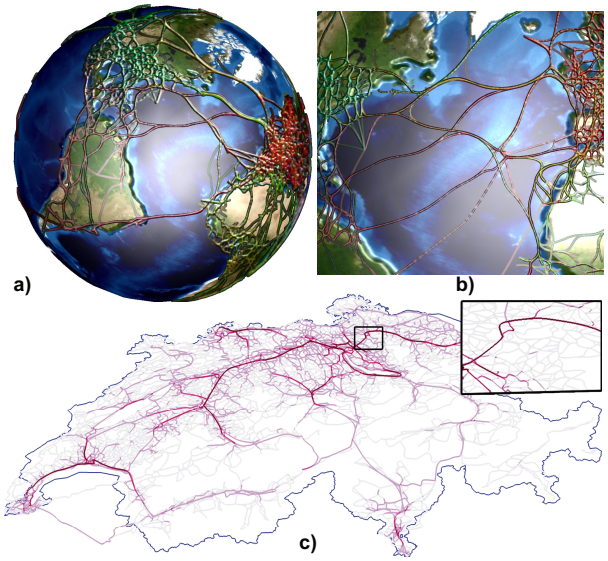

Figure 14: *3D bundling of 2000 worldwide flight trails (a), and detail (b) [LBA10a]; Bundling commuter trails along the Swiss road network [TP15] (detail top-right).*

For the same use-case (worldwide flight analysis), Lambert *et al.* [LBA10b] extend WR to bundle 2000 flight trails over the Earth surface. They show how 3D bundling is superior to bundling on a 2D map for assessing flight lengths and for more exact trail-to-trail distance assessment (Fig. 14a,b). Finally, Vector Maps [TP15] bundle commuter paths on the 3D Swiss road network. The key parts of the road network are used as a 'skeleton' to attract bundles. The visualization shows which main routes (*e.g.* highways) are important for which parts of the traffic (Fig. 14c).

6.3. Eye-track analysis

Eye tracking delivers datasets consisting of 2D points (so-called fixation points of the human gaze) linked by transitions (called saccades) [TWK*10]. Analyzing such tracks is important for many applications, *e.g.* assessing user performance when using new interfaces [CFL10, KPAA10] and finding how users read a display and whether they do it efficiently [SL87, KRDC97]. An important part of the analysis of eye trails is detecting the so-called fixation areas (FA's), defined as groups of many close fixation points; and finding how FA's are linked by saccades.

Trail bundling perfectly suits such tasks. Recent works in eye-tracking also strongly consider bundling as a viable solution to reduce clutter induced by the large amount of ocular trail sets [BKR*14]. Application-wise, ADEB [PHT15] was used to show how bundled trails can be used to assess the proficiency of users (Fig. 15a,b). Here, the ocular behavior of a novice and expert user during a multi-task experiment are compared. The background image shows the GUI that the users had to monitor and interact with during the experiment. From this, the authors show how and where the expert performed better, which can lead to improvements in either the training procedure or the GUI being proposed to users. Separately, Hurter *et al.* considered the time information present in eye trails, by bundling only trails that are compatible with respect to occurring close to each other in time using a streaming method [HEF*14]. Figure 15c,d shows how dynamic bundling highlights salient eye-movement patterns of an aircraft pilot during a landing sequence. the background images show actual views from the cockpit, with dashboard instruments in focus. Bundles show the pilot's so called main visual strategies during landing. This helps pilots to analyze and correct their behavior (improves training) but also designers of new aircraft landing-assisting cockpit instruments in finding if such instruments have been effectively used by pilots.

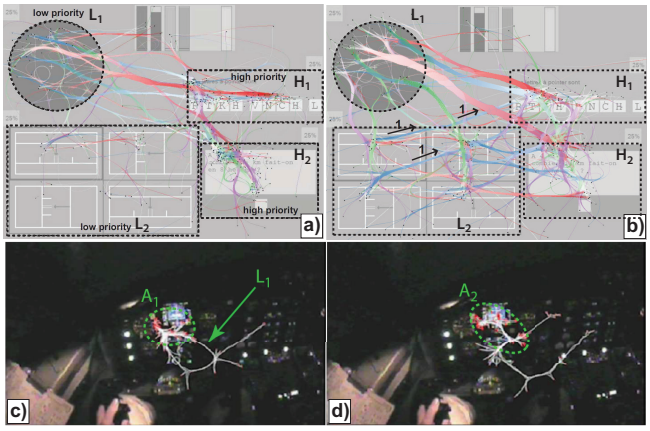

Figure 15: *Bundled eye-tracking trails of novice (a) and expert (b) users in a multitask experiment [PHT15]. (c) Visual analysis of pilot eye-tracking data with dynamic bundling [HEF*14].*

6.4. Multidimensional data exploration

Multidimensional data exploration is a very challenging field: Datasets whose samples have many dimensions (tens or even hundreds) have to be mapped to 2D or 3D. Relevant tasks include finding groups of similar samples, outlier samples, and correlations and trends of subsets of the existing dimensions.

Such data can be visualized by Parallel Coordinate Plots (PCPs) (see Sec. 3.3.1.1). As explained there, PCPs for thousands of samples or more quickly become cluttered. Bundling can help here, much in the same way it helps declutter straight-line graph drawings. Figure 16 shows four bundling methods for PCPs discussed in Sec. 3.3.1.1. As visible, each method targets different tasks: McDonnell *et al.* (Fig. 16a) separate the data into compact clusters, emphasizing data cluster differences (which sample groups are different). Coloring and shading are very similar to IBEB [TE10] (Sec. 4.3.3). Heinrich *et al.* (Fig. 16b) emphasize the continuity of the PCP polylines, helping end-to-end tracing, as opposed to all other methods. Zhou *et al.* offer a proposition similar to [MM08], but with less overlap. Finally, Palmas *et al.* offer the strongest clutter reduction (but overdraw increase) (Fig. 16d). Their visual design is very similar to HEB [Hol06] – just as HEB lets one see how *groups* of nodes in a graph are connected, so do they show how *ranges* of variables occur together in a multidimensional dataset.

Dimensionality-reduction (DR) methods are another way to display high-dimensional data. Given an n-dimensional dataset, a DR method creates a 2D scatterplot where inter-point distances reflect the corresponding nD distances [SVPM14]. It is well known that DR techniques cannot perfectly map nD distances to 2D. So, ways are needed to assess errors in such 2D scatterplots. Martins *et al.* classify such errors into false neighbors (points too close in 2D *vs* nD and missing neighbors (points too far in 2D *vs* nD) [MCMT14, MMT15]. They next extend these notions to groups (clusters) of points representing concepts. Bundling, done with CUBu [vdZCT16] helps showing missing (group) neighbors: All 2D scatterplot point-pairs which are farther apart, as compared to their nD counterparts, than a given value, are connected by edges, whose opacity reflects the 2D-nD distancec discrepancy. Next, these edges are bundled. This effectively shows which zones in a DR plot miss information, and where this information is. For example, in Fig. 17a, bundles show that many of the elements of the group Γ_{left} in the scatterplot miss neighbors which the DR method erroneously places in group Γ_{top}, but miss no neighbors with respect to the group Γ_{bottom}. Bundling is also used to compare two plots created by different DR methods.

Bundling is also used to visualize training of artificial deep neural networks (DNNs). Understanding how such networks learn from

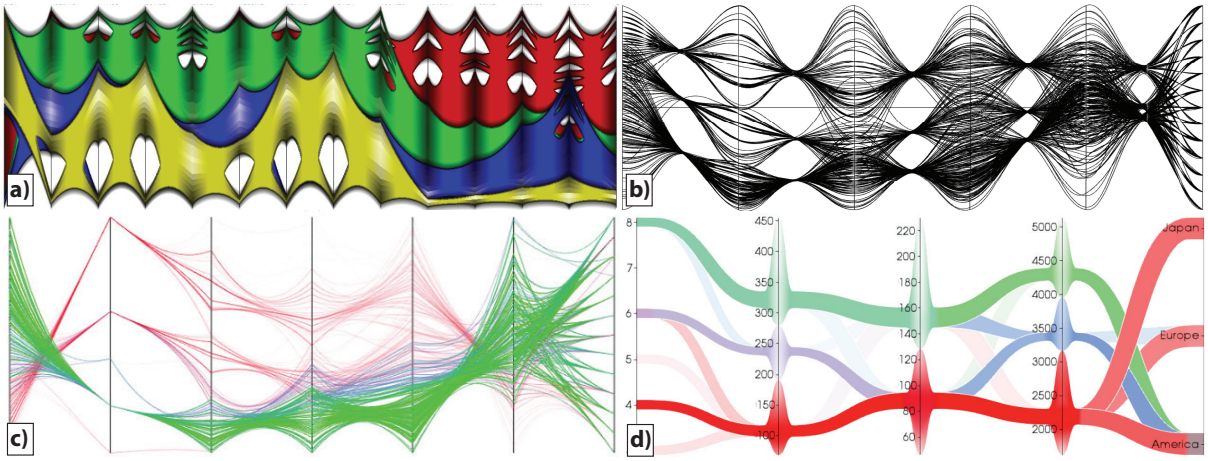

Figure 16: *PCP bundling with four methods. (a) [MM08], (b) [HLK*12], (c) [ZYQ*08], (d) [PBO*14].*

examples is notoriously hard, as they usually operate as black boxes [MPG*14]. Rauber *et al.* [RFFT17] use bundling to help this: Imagine that all neurons of a DNN are points in *n*D space, with coordinates given by their so-called activations. Such a DNN is typically trained by feeding it a sequence of *N* examples. Hence, activations change with each learned example. This can be visualized by projecting all neuron activations to 2D (using the well-known t-SNE DR technique [vdMH08]) and next constructing trails linking the *N* 2D positions of the same neuron. This yields a streaming trail-set, which next can be bundled, as in Fig. 17b. Here, luminance encodes training time, and color encodes neurons specialized for the same task (in [RFFT17], this is classifying images). The bundle structure, highlighted by the arrows, shows how originally untrained neurons (image center, dark) progressively diverge from each other. This serves in assessing the training performance: For a DNN, we want indeed that neurons progressively differentiate from each other and specialize for doing different tasks.

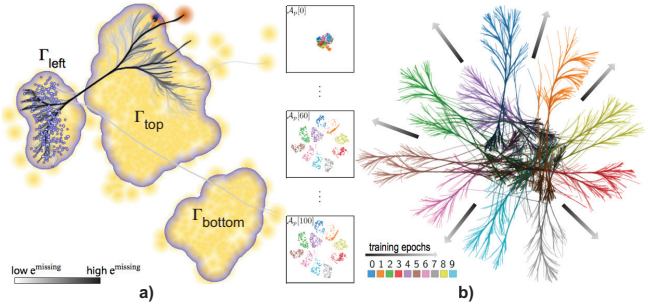

Figure 17: *a) Visualizing errors in multidimensional projections [MCMT14]. b) Visualizing learning in a neural network [RFFT17].*

6.5. Vector and Tensor Fields

Trail bundling has also been used to simplify displays of vector and tensor fields. For vector fields, Yu *et al.* [YWSC12] bundle 2D and 3D streamlines to produce simplified visualizations of the respective fields. The tasks addressed cover reduction of clutter and occlusion (in 3D) and easily spotting salient field patterns such as separatrices, laminar flow regions, and turbulent regions [PVH*03]. Compared to other hierarchical vector field simplification methods [TvW99], this approach can better capture salient vector field structures at similar levels of detail. It is however important to note that bundling is done here *purely* to find groups of very similar streamlines (following a similarity definition analogous to δ, Eqn. 2). After such groups are found, an actual physically correct streamline best representing each group is rendered. Thus, no geometric deformation takes place.

Böttger *et al.* [BSL*14] bundle 3D trails to help neuroscientists visualize brain connectivity captured by fMRI techniques. The input data is a graph G, with nodes V representing 3D locations in the brain and edges E linking locations that are related with respect to function. Straight-line drawings of G produces highly cluttered pictures (Fig. 18c). KDEEB bundling (adapted to 3D) on G massively reduces occlusion and allows one to see how groups of spatially close nodes inter-relate (Fig. 18d). For this use-case, edge deformation is not an issue, as edges only carry connectivity information.

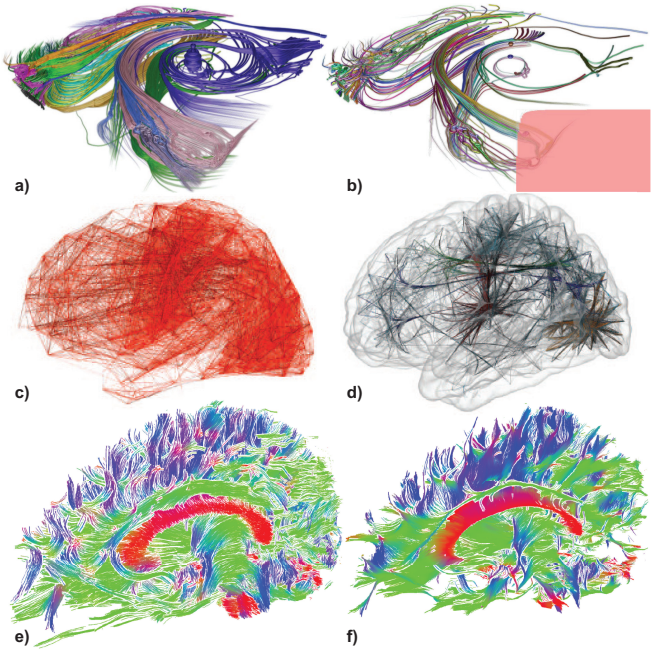

Figure 18: *Vector and tensor fields. 3D streamlines, raw (a) and (b) bundled [YWSC12]. Functional brain connectivity [BSL*14], (c) raw and (d) bundled. DTI tracts [EBB*15], (e) raw and (f) bundled.*

Bundling is also applied to Diffusion Tensor Imaging (DTI) fields. From these fields, trail (or tract) sets indicating the locations of important white-matter neural fibers are extracted by tractography [AP08]. Rendering the raw tracts yields relatively cluttered images, depending on actual tract extraction settings (Fig. 18d). Everts *et al.* bundle tracts (for details, see Sec. 3.3.1.2) to yield simplified, less cluttered, images (Fig. 18e). Tracts are next colored to indicate 3D orientation (for details, see [Tel15], Ch. 7). In contrast to [BSL*14], deformations are now constrained, as actual track locations are important. The bundled views are typically used to assess how (strongly) different anatomical regions in the brain are connected, which can next help planning minimally disruptive surgery.

7. Discussion

We now discuss several aspects related to the state of graph and trail bundling. We focus on current capabilities (Sec. 7.1) and challenges (Sec. 7.2), and also outline gaps and future work directions.

7.1. Current State of Bundling

Bundling methods have become quite mature and widely used since their inception over a decade ago. Several aspects are relevant here:

Scalability: We distinguish five 'generations' of bundling techniques. Pre-HEB bundling techniques were able to handle graphs of hundreds of edges and worked solely on the CPU [New89, BW98, DEGM03, PXY*05]. Following HEB [Hol06], the second generation targeted graphs and trail-sets having thousands (up to roughly 10K) paths; MINGLE [GHNS11] is an exception here, as it targeted graphs up to 1 million edges. These techniques worked mostly geometrically [HVW09, NEH12, CZQ*08, NHE11, LLCM12, SHH11] and on the CPU. Starting with SBEB [EHP*11] and KDEEB [HET12], third-generation techniques are image-based, using a mixed CPU-GPU implementation to massively parallelize both similarity computation (Sec. 4.1.2.2) and the bundling itself (Sec. 4.2, and can bundle tens of thousands of paths in seconds [LBA10b, LBA10a, Mou15]. Fifth-generation techniques work solely on the GPU, can handle multiple GPUs (CUBu [vdZCT16]), can bundle up to a million paths in subsecond time, and remove GPU RAM limitations by data streaming(FFTEB [LHT17]). With even faster and larger-memory GPUs emerging continuously, we believe that the scalability problem of path bundling has been sufficiently addressed, so that bundling can approach 'big data' sets.

Data coverage: Following our taxonomy in Sec. 3, bundling methods can cover a wide spectrum of data types: graph drawings (trees, compound, DAGs, general oriented or not); 2D trails (vehicle movements, eye tracks, streamlines); and 3D trails (vehicle movements, DTI fibers, streamlines). All these can be either attributed (with several attributes per path) or not, and time dependent or not (see also Tab. 1). As such, we believe that most data types amenable to bundling are covered by existing methods.

Adoption: The development of bundling techniques has grown parallel to widening their application. Bundling has been arguably best accepted in software engineering and geospatial trail analysis (Sec. 6). Other salient application fields are eye-tracking analysis, DTI tract visualization, and network visualization. Several mature software tools offer bundling, *e.g.* GraphViz [G*17], Tulip [A*17], Gephi [Gep17], D3 [Bos17], and Protégé [T*17, HdRFH12]. Yet, except a few methods like HEB [Hol06], FDEB [HVW09], and MINGLE [GHNS11], most other bundling methods have still not been integrated in such mainstream packages.

7.2. Bundling Challenges

Despite its success, bundling techniques haves still unsolved challenges. Of these, we discuss next the quality assessment of bundled drawings (Sec. 7.2.1), the issue of faithfulness, or how much information is kept (or not) in a bundled drawing (Sec. 7.2.1), and how one can control the result of bundling (Sec. 7.2.3).

7.2.1. Quality assessment

There is no accepted way to measure the quality of a bundling. The problem core is that it is hard to define objective criteria for what a 'good' bundling is. Quality metrics have been discussed, and advocated for, since long in information visualization [Bra97, MHNW97, EG06]. Closer to our scope, these include the ink-ratio and 'lie factor' of a visualization [Tuf92]; defining visual clutter [ED07]; and measuring edge congestion [CR01], edge crossings [KPS14], readability [DS09, EHKN15], aesthetics [PCJ95, WPCM02], and faithfulness [NEH13, NEH17] in graph visualizations. However, no such set of metrics fully covers path bundling. For bundled drawings quality, only a handful of studies exist, touching upon the visual navigability of small bundled drawings [PNK10], comparing the effectiveness of bundled *vs* unbundled drawings [TEHR09], studying the comprehensibility [MD12] and ambiguity [BRH*16] of bundled drawings, and visualizing edge deformation [HET12].

Quality assessment can be approached by defining what quality is. Following well-established principles in software engineering [Ken03], we can define the quality of a bundled drawing by either measuring its 'fitness for purpose' (how well it helps solving a certain problem) or by comparing it to a ground-truth whose quality is known. Both paths have, however, challenges, as outlined next.

Fitness for purpose: To quantify this, we need first to define what the goal(s) of a bundled drawing are. Section 5.2 outlines the tasks that bundling aims to cover. This (or a similar) proposal could be next used to scope, and assess the value of, their contributions. This can be done by user studies where the percentage, correctness, and time of completing, a given task is measured. Besides the know challenges that organizing large-scale user studies (and generalizing their results) have, an extra difficulty is that the same method B can generate a wealth of different drawing styles from the same dataset $D(P)$, see *e.g.* [vdZCT16]. The data-based taxonomy proposed in Sec. 3 can help here in narrowing the focus of methods to be compared against each other based on the type of input data they work on.

Ground truth comparison: Quality can be measured by computing the difference between a bundling $B(D(P))$ produced by the method under study and a so-called ground truth $B_g(D(P))$, *i.e.*, a bundled drawing known to be good for a certain task. Most existing bundling papers do this implicitly, by comparing their results with earlier methods on the same dataset $D(P)$. Yet, in most (if not all) cases, the comparison is only visual, such as shown in Fig. 4. This can be improved by using quantitative metrics to compare two bundled images, *e.g.* by measuring their Hausdorff distance, or metrics to assess global quality parameters of a drawing $B(D(P))$, *e.g.* amount of overdraw, spread of path displacements, spatial path-density distribution, amount of overlap of different-direction bundles, amount of bundle crossings, and bundle curvature distribution. We can next infer good values for such metrics *e.g.* by extrapolating from the graph-drawing and graph-aesthetic principles mentioned earlier, and compare actual metric values with the desired good values. Another way to measure quality is to compare $B(D(P))$ with the unbundled drawing $D(P)$, using a similar set of quality metrics. A good bundling is, in this case, one that increases the metrics' values.

A separate problem for ground truth comparison is that there is, so far, no established benchmark of graphs and trail-sets (and bundling method implementations) that the infovis community could use. Creating such a benchmark is reasonably easy and should be highly useful for the community. A starting point for graphs (including node layouts) can be the well-known Florida collection [DH11].

7.2.2. Bundling faithfulness

A separate issue regards the information alteration or loss produced by bundling, or the so-called faithfulness of the produced drawings [NEH13, NEH17]. Simply put, we need to measure (a) how much of the original information conveyed by $D(P)$ is kept by $B(D(P))$, and (b) how much incorrect information $B(D(P))$ adds as compared to $D(P)$. In other words, we need to measure the precision and recall for B. The above are related to the concept of inferrence affordance that measures the informational equivalence of two displays [SL87, CFL10, TWK*10] – in our case, $D(P)$ and $B(D(P))$.

Concerning (a), it is clear that bundling looses some of the information present in the original $D(P)$. This is a higher problem for trail-sets than for graphs, since the former contain relevant spatial information. To assess this, we can measure (1) the amount of trail

distortion created by B, *i.e.*, $\sum_{\mathbf{p}_i \in D(P)} \delta(\mathbf{p}_i, B(\mathbf{p}_i))$, with δ given by Eqn. 2. Next, depending on the task, we can decide whether this amount is acceptable or not. To help this, distortion can be visualized on $B(D(P))$ [HET12]. When distortion is too large, relaxation can be used (Sec. 4.3.6), or bundling can be selectively stopped when it reaches a (local or global) maximum value. Further, a ratio of clutter reduction to amount of distortion can be computed, analogously to Tufte's ink-space ratio [Tuf92] to measure the faithfulness of B. Another way to increase faithfulness is to animate $D(P)$ towards $B(D(P))$ during the bundling [HEF*14]. This helps users match the input and output of B, and thus reduce the information loss.

Concerning (b), it has been shown that, for some datasets, bundling *can* yield false insights. Figure 19 illustrates this for KDEEB [HET12]: A random graph whose nodes and edges are uniformly spread over a 2D region is bundled. The result shows emerging structures which, however, do not reflect any actual patterns in the input data. This effect is mainly due to the fact that implicit bundling methods expect that their input *has* salient structures; when this is not the case, the bundling's underlying density-sharpening principle (Sec. 2) will simply accentuate small-scale noise.

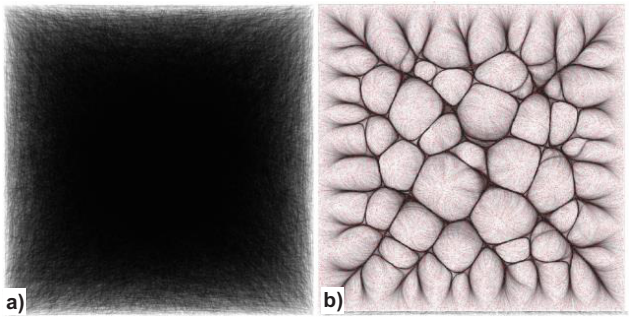

Figure 19: *Pseudo-random graph (a) and its KDEEB bundling (b). Red dots represents graph nodes [HET12].*

7.2.3. Bundling control

Controlling bundling methods is our final challenge. Earlier bundling methods have a relatively small, and intuitive, set of parameters. For instance, HEB (Sec. 3.2.1.1) allows controlling the strength of bundling, the amount of relaxation, detection of the common ancestor in the hierarchy that bundles are routed through, and type of edge blending (for full details, we refer to [Hol06]). Recent bundling methods have a higher number of parameters. For instance, CUBu (Sec. 3.3.1.1) allows controlling the resolution of the image used for kernel density estimation, radius of the kernel, number of sampling points along paths, advection step size, number of bundling iterations, directional bundling style, type of bundle shading, and offers additionally four drawing styles (full details in [vdZCT16]). Concerning parameters, we see the following challenges:

Semantics: Existing parameters have semantics which are typically bound to a specific bundling technique. This makes it hard for users to reproduce results when changing techniques. Our mathematical framework presented in this survey helps this by identifying parameters having identical meanings over different methods, but of course cannot fully solve the problem. A promising direction would be to provide parameters linked to user tasks rather than method technicalities. For instance, if one wants to bundle trails so that a given subset thereof is easily followable, the method could automatically select suitable technical parameter values.

Impact: Changing any of the many bundling parameters produces a different result. While reasonable parameter presets exist, these are not always optimal for all datasets and/or visual insights sought. This issue is less critical with modern bundling methods which work near-real-time (Sec. 3.3.1.1), as trial-and-error parameter exploration is less costly. Still, supporting users to express their interest more effectively, by offering high(er) level parameter control, similar to work done elsewhere in infovis [SvW08, SvW09], is a potential improvement direction for bundling. Another solution would be setting parameter values based on the characteristics of the input data $D(P)$ to bundle.

Coupling: The same bundling result can be, usually, obtained by setting different parameters to different values. For example, the same bundle tightness in implicit methods [HVW09, HET12, Mou15, vdZCT16] can be obtained by changing either the number of bundling iterations or the advection step size. We call such parameters *coupled*. Few papers discuss coupling, making the parameter space unnecessarily large. This can be alleviated by grouping coupled parameters under a single high-level parameter, similarly to the idea discussed above for impact.

8. Conclusion

We have presented a survey of the state-of-the-art in graph and trail bundling techniques. We organize such techniques via a data-based taxonomy, which allows readers to find the types of techniques that best fit their specific data at hand. Next, we propose a mathematical framework to define bundling, which allows one to compare specific technical aspects of the different existing bundling algorithms in a detailed way. Based on this framework, we discuss a wide set of bundling techniques, spanning the whole spectrum present in the literature. We next discuss how bundling supports clutter reduction and proposed a taxonomy for the tasks it supports. We present a wide sample of applications that rely on bundling from five domains (software engineering, vehicle trajectory analysis, eye track analysis, multidimensional data visualization, and vector and tensor visualization). Finally, we outline the key aspects which modern bundling methods have solved and also the main open issues the field has, thereby highlighting important directions for future research.

We believe our survey systematizes and clarifies several so-far open points in the bundling literature and will serve both practitioners in understanding how to choose, parameterize, and use bundling techniques to solve concrete problems, and also researchers to compare, discuss, understand, and refine future bundling algorithms.

9. Acknowledgments

The authors acknowledge the support of the Federal University of Toulouse (Université Fédérale Toulouse Midi-Pyrénées – UFTMiP) under the grant "MEMOIRE", the French National Agency for Research (Agence Nationale de la Recherche – ANR) under the grant ANR-14-CE24-0006-01 project "TERANOVA" and the SESAR Research and Innovation Action Horizon 2020 under project "MOTO" (The embodied reMOte TOwer).

References

[A*17] AUBER D., ET AL.: Tulip visualization framework, 2017. tulip.labri.fr. 19

[ALLF07] ALEXANDER A. L., LEE J. E., LAZAR M., FIELD A. S.: Diffusion tensor imaging of the brain. *Neurotherapeutics 4*, 3 (2007), 316–329. 9

[AMA08] ARCHAMBAULT D., MUNZNER T., AUBER D.: Grouseflocks: Steerable exploration of graph hierarchy space. *IEEE TVCG 14*, 4 (2008), 900–913. 3

[AP08] ASSAF ï., PASTERNAK O.: Diffusion tensor imaging DTI-based white matter mapping in brain research: A review. *J Mol Neurosci 34* (2008), 51–61. 5, 18

[APP10] ARCHAMBAULT D., PURCHASE H. C., PINAUD B.: The readability of path-preserving clusterings of graphs. *CGF 29* (2010), 1173–1182. 3

[AS94] AHLBERG C., SHNEIDERMAN B.: Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (1994), ACM, pp. 313–317. 15

[AvHK06] ABELLO J., VAN HAM F., KRISHNAN N.: ASK-GraphView: A large scale graph visualization system. *IEEE TVCG 12*, 5 (2006), 669–676. 3, 4

[BBD*10] BINUCCI C., BRANDES U., DIBATTISTA G., DIDIMO W., GAERTLER M., PALLADINO P., PATRIGNANI M., SYMVONIS A., ZWEIG K.: Drawing trees in a streaming model. In *Proc. Graph Drawing* (2010), Springer, pp. 292–303. 7

[BBDW14] BECK F., BURCH M., DIEHL S., WEISKOPF D.: The state of the art in visualizing dynamic graphs. *EuroVis STAR* (2014). 7

[BE05] BRANDES U., ERLEBACH T.: *Network analysis: methodological foundations*. Springer, 2005. 10

[Ber83] BERTIN J.: *Semiology of Graphics*. University of Wisconsin Press, 1983. 12

[BISP16] BOURQUI R., IENCO D., SALLABERRY A., PONCELET P.: Multilayer graph edge bundling. In *Proc. IEEE PacificVis* (2016), pp. 184–188. 15

[BKR*14] BLASCHECK T., KURZHALS K., RASCHKE M., BURCH M., WEISKOPF D., ERTL T.: State-of-the-art of visualization for eye tracking data. In *EuroVis - STARs* (2014). 17

[BM13] BREHMER M., MUNZNER T.: A multi-level typology of abstract visualization tasks. *IEEE TVCG 19*, 12 (2013), 2376–2385. 2, 13, 15

[Bos17] BOSTOCK M.: Hierarchical edge bundling implementation in D3, 2017. https://gist.github.com/mbostock. 19

[Bra97] BRATH R.: Metrics for effective information visualization. In *Proc. IEEE InfoVis* (1997), pp. 55–63. 19

[BRH*16] BACH B., RICHE N., HURTER C., MARRIOTT K., DWYER T.: Towards unambiguous edge bundling: Investigating confluent drawings for network visualization. *IEEE TVCG 23*, 1 (2016), 541–550. 4, 6, 10, 19

[BSH13] BOTHOREL G., SERRURIER M., HURTER C.: Visualization of frequent itemsets with nested circular layout and bundling algorithm. In *Proc. ISVC* (2013), Springer, pp. 396–405. 15, 16

[BSL*14] BÖTTGER J., SCHÄFER A., LOHMANN G., VILLRINGER A., MARGULIES D. S.: Three-dimensional mean-shift edge bundling for the visualization of functional connectivity in the brain. *IEEE TVCG 20*, 3 (2014), 471–480. 1, 4, 5, 7, 11, 12, 18

[BSV11] BUCHIN K., SPECKMANN B., VERBEEK K.: Angle restricted Steiner arborescences for flow map layout. In *Proc. ISAAC* (2011), pp. 250–259. 4, 6, 10, 11

[BTD12] BUSCHMANN S., TRAPP M., DÖLLNER J.: Challenges and approaches for the visualization of movement trajectories in 3D geovirtual environments. In *Proc. GIScience Workshop on GeoVisual Analytics – Time to Focus on Time* (2012). 10

[BVKW11] BURCH M., VEHLOW C., KONEVTSOVA N., WEISKOPF D.: Evaluating partially drawn links for directed graph edges. In *Proc. Graph Drawing* (2011), pp. 56–67. 2

[BW98] BRANDES U., WAGNER D.: Using graph layout to visualize train interconnection data. In *Proc. Graph Drawing* (1998), Springer, pp. 44–56. 3, 11, 19

[CC07] COLLINS C., CARPENDALE S.: VisLink: Revealing relationships amongst visualizations. *IEEE TVCG 13*, 6 (2007), 1192–1199. 1, 5

[CFL10] CÖTELKIN A., FABRIKANT S. I., LACAYO M.: Exploring the efficiency of users visual analytics strategies based on sequence analysis of eye movement recordings. *Int. J. Geogr. Inf. Sci. 24*, 10 (2010), 1559–1575. 17, 19

[CHK*01] CHAPIN N., HALE J. E., KHAN K. M., RAMIL J. F., TAN W.-G.: Types of software evolution and software maintenance. *J. Soft. Maint. Evol. Res. Pract. 13* (2001), 3–30. 15

[CM02] COMANICIU D., MEER P.: Mean shift: A robust approach toward feature space analysis. *IEEE TPAMI 24*, 5 (2002), 603–619. 1, 9, 15

[CR01] CARPENDALE M. S. T., RONG X.: Examining edge congestion. In *Proc. ACM CHI* (2001), pp. 115–116. 2, 19

[CTMT10] CAO T., TANG K., MOHAMED A., TAN T.: Parallel banding algorithm to compute exact distance transform with the GPU. In *Proc. ACM SIGGRAPH Symp. on Interactive 3D Graphics and Games* (2010), pp. 134–141. 11

[CZB11] CASERTA P., ZENDRA O., BODÉNÈS D.: 3D hierarchical edge bundles to visualize relations in a software city metaphor. In *Proc. IEEE VISSOFT* (2011), pp. 34–42. 4, 7, 11, 15

[CZH*08] CORNELISSEN B., ZAIDMAN A., HOLTEN D., MOONEN L., VAN DEURSEN A., VAN WIJK J. J.: Execution trace analysis through massive sequence and circular bundle views. *Journal of Systems and Software 81*, 12 (2008), 2252–2268. 4, 6, 10, 11, 12, 14

[CZQ*08] CUI W., ZHOU H., QU H., WONG P. C., LI X.: Geometry-based edge clustering for graph visualization. *IEEE TVCG 14*, 6 (2008), 1277–1284. 1, 4, 5, 7, 11, 16, 19

[CZvD*09] CORNELISSEN B., ZAIDMAN A., VAN DEURSEN A., MOONEN L., KOSCHKE R.: A systematic survey of program comprehension through dynamic analysis. *IEEE Trans Softw Eng 35*, 5 (2009), 684–702. 15

[dBCvKO10] DE BERG M., CHEONG O., VAN KREVELD M., OVERMARS M.: *Computational Geometry: Algorithms and Applications*. Springer, 2010. 2

[DEGM03] DICKERSON M., EPPSTEIN D., GOODRICH M. T., MENG J. Y.: Confluent drawings: Visualizing non-planar diagrams in a planar way. In *Proc. Graph Drawing* (2003), Springer, pp. 1–12. 1, 3, 4, 6, 19

[DEGM05] DICKERSON M., EPPSTEIN D., GOODRICH M., MENG J.: Confluent drawings: Visualizing non-planar diagrams in a planar way. *Journal of Graph Algorithms and Applications 9*, 1 (2005), 31–52. 4, 6

[DH11] DAVIS T. A., HU Y.: The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS) 38*, 1 (2011), 1. http://www.cise.ufl.edu/research/sparse/matrices. 19

[dHINM04] DE HOON M., IMOTO S., NOLAN J., MYIANO S.: Open source clustering software. *Bioinformatics 20*, 9 (2004), 1453–1454. 10

[DHRMM13] DWYER T., HENRY RICHE N., MARRIOTT K., MEARS C.: Edge compression techniques for visualization of dense directed graphs. *IEEE TVCG 19*, 12 (2013), 2596–2605. 6, 10

[Die08] DIEHL S.: *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, 2008. 4, 12, 15

[DMM*14] DWYER T., MEARS C., MORGAN K., NIVEN T., MARRIOTT K., WALLACE M.: Improved optimal and approximate power graph compression for clearer visualisation of dense graphs. In *Proc. IEEE PacificVis* (2014), pp. 105–112. 6, 10

[DMW07] DWYER T., MARRIOTT K., WYBROW M.: Integrating edge routing into force-directed layout. In *Proc. Graph Drawing* (2007), pp. 8–19. 1

[DS09] DUNNE C., SHNEIDERMAN B.: *Improving graph drawing readability by incorporating readability metrics: A software tool for network analysts*. Tech. rep., Technical Report HCIL-2009-13, University of Maryland, 2009. 19

[DS13] DUNNE C., SHNEIDERMAN B.: Motif simplification: Improving network visualization readability with fan, connector, and clique glyphs. In *Proc. ACM CHI* (2013), p. 3247âĂŞ3256. 4

[DT14] DIEHL S., TELEA A.: Multivariate networks in software engineering. In *Multivariate Network Visualization* (2014), Kerren A., Purchase H., Ward M., (Eds.), Springer, pp. 13–35. 2, 4, 6, 7, 10

[EBB*15] EVERTS M. H., BEGUE E., BEKKER H., ROERDINK J. B. T. M., ISENBERG T.: Exploration of the brain's white matter structure through visual abstraction and multi-scale local fiber tract contraction. *IEEE TVCG 21*, 7 (2015), 808–821. 1, 2, 4, 5, 9, 18

[ED07] ELLIS G., DIX A.: A taxonomy of clutter reduction for information visualisation. *IEEE TVCG 13*, 6 (2007), 1216–1223. 2, 14, 15, 19

[EG06] E.BERTINI, G.SANTUCCI: Visual quality metrics. In *Proc. AVI BELIV* (2006). 19

[EHKN15] EADES P., HONG S.-H., KLEIN K., NGUYEN A.: Shape-based quality metrics for large graph visualization. In *Proc. Graph Drawing* (2015), pp. 502–514. 19

[EHP*11] ERSOY O., HURTER C., PAULOVICH F., CANTAREIRO G., TELEA A.: Skeleton-based edge bundles for graph visualization. *IEEE TVCG 17*, 2 (2011), 2364–2373. 1, 4, 7, 9, 10, 11, 12, 13, 16, 19

[EN08] EMANUELSSON P., NILSSON U.: A comparative study of industrial static analysis tools. *Electronic Notes in Theoretical Computer Science 217* (2008), 5–21. 15

[FCTB08] FABBRI R., COSTA L. D. F., TORELLI J. C., BRUNO O. M.: 2D Euclidean distance transform algorithms: A comparative survey. *ACM Comput Surv 40*, 1 (2008). 11

[G*17] GANSNER E., ET AL.: GraphViz graph drawing framework, 2017. www.graphviz.org. 19

[GBE08] GIERETH M., BOSCH H., ERTL T.: A 3D treemap approach for analyzing the classificatory distribution in patent portfolios. In *Proc. IEEE VAST* (2008), pp. 189–193. 1, 4, 7, 11

[Gep17] GEPHI CONSORTIUM: Gephi large network visualization framework, 2017. gephi.org. 19

[GHNS11] GANSNER E., HU Y., NORTH S., SCHEIDEGGER C.: Multilevel agglomerative edge bundling for visualizing large graphs. In *Proc. IEEE PacificVis* (2011), pp. 187–194. 1, 4, 5, 7, 10, 11, 16, 19

[GK03] GRAHAM M., KENNEDY J.: Using curves to enhance parallel coordinate visualisations. In *Proc. IEEE Information Visualisation* (2003), pp. 163–171. 9, 11

[GK06] GANSNER E., KOREN Y.: Improved circular layouts. In *Proc. Graph Drawing* (2006), pp. 386–398. 4, 5, 11

[GKN04] GANSNER E., KOREN Y., NORTH S.: Topological fisheye views for visualizing large graphs. In *Proc. IEEE InfoVis* (2004), pp. 175–182. 4

[GN00] GANSNER E. R., NORTH S. C.: An open graph visualization system and its applications to software engineering. *Software – Practice and Eperience 30*, 11 (2000), 1203–1233. 3

[Han13] HANJALIĆ A.: ClonEvol: Visualizing software evolution with code clones. In *Proc. IEEE VISSOFT* (2013). 4, 9, 14, 15

[Har94] HARALICK R. M.: *Mathematical Morphology: Theory and Hardware*. Oxford University Press, 1994. 9

[HCGT14] HURTER C., CONVERSY S., GIANAZZA D., TELEA A.: Interactive image-based information visualization for aircraft trajectory analysis. *Transportation Research C 47*, 2 (2014), 207–227. 10, 12

[HdRFH12] HOP W., DE RIDDER S., FRASINCAR F., HOGENBOOM F.: Using hierarchical edge bundles to visualize complex ontologies in GLOW. In *Proc. ACM Applied Computing* (2012), pp. 304–311. 5, 19

[HDZ05] HANSEN G. A., DOUGLASS R. W., ZARDECKI A.: *Mesh enhancement*. Imperial College Press, 2005. 13

[HEF*14] HURTER C., ERSOY O., FABRIKANT S., KLEIN T., TELEA A.: Bundled visualization of dynamic graph and trail data. *IEEE TVCG 20*, 8 (2014), 1141–1157. 1, 4, 10, 12, 13, 14, 15, 16, 17, 20

[HET11] HURTER C., ERSOY O., TELEA A.: MoleView: An attribute and structure-based semantic lens for large element-based plots. *IEEE TVCG 17*, 12 (2011), 2600–2609. 14, 15

[HET12] HURTER C., ERSOY O., TELEA A.: Graph bundling by kernel density estimation. *CGF 31*, 3 (2012), 865–874. 1, 2, 4, 7, 9, 11, 13, 14, 16, 19, 20

[HET13] HURTER C., ERSOY O., TELEA A.: Smooth bundling of large streaming and sequence graphs. In *Proc. IEEE PacificVis* (2013). 1, 4, 7, 10, 13, 14

[HEW98] HUANG M., EADES P., WANG J.: On-line animated visualization of huge graphs using a modified spring algorithm. *JVLC 9*, 6 (1998), 623–645. 7

[HFM07] HENRY N., FEKETE J.-D., MCGUFFIN M. J.: Nodetrix: a hybrid visualization of social networks. *IEEE TVCG 13*, 6 (2007), 1302–1309. 4

[HHM08] HARRIS J., HIRST J. L., MOSSINGHOFF M.: *Combinatorics and Graph Theory*. Springer, 2008. 2nd edition. 2

[HIVWF11] HOLTEN D., ISENBERG P., VAN WIJK J. J., FEKETE J.-D.: An extended evaluation of the readability of tapered, animated, and textured directed-edge representations in node-link graphs. In *Proc. IEEE PacificVis* (2011), pp. 195–202. 12, 13

[HLK*12] HEINRICH J., LUO Y., KIRKPATRICK A. E., ZHANG H., WEISKOPF D.: Evaluation of a bundling technique for parallel coordinates. In *Proc. Information Visualization* (2012), pp. 240–248. 4, 9, 18

[HMM00] HERMAN I., MELANCON G., MARSHALL M. S.: Graph visualization and navigation in information visualization: A survey. *IEEE TVCG 6*, 1 (2000), 24–43. 1, 3

[Hol06] HOLTEN D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE TVCG 12*, 5 (2006), 741–748. 1, 4, 5, 6, 7, 11, 12, 13, 17, 19, 20

[HTC09] HURTER C., TISSOIRES B., CONVERSY S.: FromDaDy: Spreading data across views to support iterative exploration of aircraft trajectories. *IEEE TVCG 15*, 6 (2009), 1017–1024. 2, 9, 10

[HTCT14] HURTER C., TAYLOR R., CARPENDALE S., TELEA A.: Color tunneling: Interactive exploration and selection in volumetric datasets. In *Proc. IEEE PacificVis* (2014), pp. 225–232. 13

[Hur15] HURTER C.: *Image-Based Visualization: Interactive Multidimensional Data Exploration*. Morgan & Claypool Publishers, 2015. 1

[HvW08] HOLTEN D., VAN WIJK J. J.: Visual comparison of hierarchically organized data. *CGF 27*, 3 (2008), 759–766. 4, 5, 15

[HVW09] HOLTEN D., VAN WIJK J. J.: Force-directed edge bundling for graph visualization. *CGF 28*, 3 (2009), 983–990. 1, 4, 5, 7, 11, 12, 13, 16, 19, 20

[HvW10] HOLTEN D., VAN WIJK J. J.: Evaluation of cluster identification performance for different PCP variants. *CGF 29*, 3 (2010), 793–802. 9

[Ins09] INSELBERG A.: *Parallel Coordinates: Visual Multidimensional Geometry and its Applications*. Springer, 2009. 2, 9

[JGH11] JIA Y., GARLAND M., HART J. C.: Social network clustering and visualization using hierarchical edge bundles. *CGF 30*, 8 (2011), 2314–2327. 4

[JMF99] JAIN A., MURTY M., FLYNN P.: Data clustering: a review. *ACM Comput Surv (CSUR) 31*, 3 (1999), 264–323. 1, 3, 5

[Ken03] KEN S. H.: *Metrics and Models in Software Quality Engineering*. Addison-Wesley, 2003. 19

[KKH89] KAMADA T., KAWAI S., HEHNER E.: An algorithm for drawing general undirected graphs. *Inform Process Lett 31*, April (1989), 7–15. 12

[KL83] KRUSKAL J. B., LANDWEHR J. M.: Icicle plots: Better displays for hierarchical clustering. *JSTOR 37*, 2 (1983), 162–168. 4

[Koe84] KOENDERINK J.: The structure of images. *Biological Cybernetics 50* (1984), 363–370. 12

[Kos03] KOSCHKE R.: Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. *J Softw Maint Evol: Research and Practice 15*, 2 (2003), 87–109. 15

[KPAA10] KIM J., PALMISANO S. A., ASH A., ALLISON R. S.: Pilot gaze and glideslope control. *ACM Trans. Appl. Perception 7*, 3 (2010). 17

[KPS14] KOBOUROV S., PUPYREV S., SAKET B.: Are crossings important for drawing large graphs? In *Proc. Graph Drawing* (2014). 19

[KRDC97] KRYGIER J. B., REEVES C., DIBIASE D., CUPP J.: Multimedia in geographic education: Design, implementation, and evaluation. *J. Geogr. Higher Educ. 21*, 1 (1997), 17–39. 17

[KS10] KIENREICH W., SEIFERT C.: An application of edge bundling techniques to the visualization of media analysis results. In *Proc. IEEE Information Visualisation* (2010). 4

[KTD13] KARRAN B., TRÜMPER J., DÖLLNER J.: SyncTrace: Visual thread-interplay analysis. In *Proc. IEEE VISSOFT* (2013). 10, 15

[KvdZT14] KLEIN T., VAN DER ZWAN M., TELEA A.: Dynamic multiscale visualization of flight data. In *Proc. VISAPP* (2014). 10, 12, 13, 15, 16

[LAM10] LAMBERT A., AUBER D., MELANCON G.: Living flows: enhanced exploration of edge-bundled graphs based on GPU-intensive edge rendering. In *Proc. Information Visualisation* (2010), IEEE, pp. 523–530. 14

[LBA10a] LAMBERT A., BOURQUI R., AUBER D.: 3D edge bundling for geographical data visualization. In *Proc. Information Visualisation* (2010), pp. 329–335. 4, 10, 17, 19

[LBA10b] LAMBERT A., BOURQUI R., AUBER D.: Winding roads: Routing edges into bundles. *CGF 29*, 3 (2010), 853–862. 1, 4, 5, 7, 10, 11, 13, 16, 17, 19

[LDB11] LAMBERT A., DUBOIS J., BOURQUI R.: Pathway preserving representation of metabolic networks. *CGF 30*, 3 (2011), 1021–1030. 4, 6, 13

[LH11] LAMPE O. D., HAUSER H.: Interactive visualization of streaming data with kernel density estimation. In *Proc. IEEE PacificVis* (2011), pp. 171–178. 12

[LHT17] LHUILLIER A., HURTER C., TELEA A.: FFTEB: Edge bundling of huge graphs by the Fast Fourier Transform. In *Proc. IEEE PacificVis* (2017). 1, 2, 4, 9, 11, 12, 16, 19

[LKS*11] LANDESBERGER T. V., KUIJPER A., SCHRECK T., KOHLHAMMER J., VAN WIJK J., FEKETE J. D., FELLNER D.: Visual analysis of large graphs: State-of-the-art and future research challenges. *CGF 30*, 6 (2011), 1719–1749. 1, 2, 3, 4

[LLCM12] LUO S. J., LIU C. L., CHEN B. Y., MA K. L.: Ambiguity-free edge-bundling for interactive graph visualization. *IEEE TVCG 18*, 5 (2012), 810–821. 4, 5, 14, 15, 19

[LPP*06] LEE B., PLAISANT C., PARR C. S., FEKETE J.-D., HENRY N.: Task taxonomy for graph visualization. In *Proc. AVI BELIV* (2006), ACM, pp. 1–5. 2, 15

[MCMT14] MARTINS R., COIMBRA D., MINGHIM R., TELEA A.: Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics 41* (2014), 26–42. 1, 17, 18

[MD12] MCGEE F., DINGLIANA J.: An empirical study on the impact of edge bundling on user comprehension of graphs. In *Proc. ACM AVI* (2012), pp. 620–627. 19

[MHNW97] MILLER N., HETZLER B., NAKAMURA G., WHITNEY P.: The need for metrics in visual information analysis. In *Proc. ACM Workshop on New Paradigms in Information Visualization and Manipulation* (1997). 19

[MM08] MCDONNELL K., MUELLER K.: Illustrative parallel coordinates. *CGF 27*, 3 (2008), 1031–1038. 1, 4, 9, 12, 17, 18

[MMT15] MARTINS R. M., MINGHIM1 R., TELEA A.: Explaining neighborhood preservation for multidimensional projections. In *Proc. Computer Graphics & Visual Computing (CGVC)* (2015), Eurographics. 17

[Mou15] MOURA D.: 3D density histograms for criteria-driven edge bundling. *arXiv:1504.02687v1 [cs.GR]* (2015). 1, 4, 9, 11, 16, 19, 20

[MPG*14] MÜHLBACHER T., PIRINGER H., GRATZL S., SEDLMAIR M., STREIT M.: Opening the black box: Strategies for increased user involvement in existing algorithm implementations. *IEEE TVCG 20*, 12 (2014), 1643–1652. 18

[Mun14] MUNZNER T.: *Visualization Analysis and Design*. CRC Press, 2014. 4

[MVvW05] MOBERTS B., VILANOVA A., VAN WIJK J. J.: Evaluation of fiber clustering methods for diffusion tensor imaging. In *Proc. IEEE Visualization* (2005), p. 65âĂŞ72. 9

[MW02] MOUSTAFA R., WEGMAN E.: On some generalizations of parallel coordinate plots, seeing a million. In *Proc. Data Visualization Workshop* (2002). 9

[NB13] NOCAJ A., BRANDES U.: Stub bundling and confluent spirals for geographic networks. In *Proc. Graph Drawing* (2013), Springer, pp. 388–399. 4, 6

[NBD*15] NOCKE T., BUSCHMANN S., DONGES J. F., MARWAN N., SCHULZ H.-J., TOMINSKI C.: Review: visual analytics of climate networks. *Nonlin. Processes Geophys. 22* (2015), 545–570. 10

[NEH12] NGUYEN Q., EADES P., , HONG S.-H.: StreamEB: stream edge bundling. In *Proc. Graph Drawing* (2012), pp. 324–332. 1, 4, 10, 11, 12, 19

[NEH13] NGUYEN Q., EADES P., HONG S.-H.: On the faithfulness of graph visualizations. In *Proc. IEEE PacificVis* (2013). 2, 19

[NEH17] NGUYEN Q. H., EADES P., HONG S.-H.: Towards faithful graph visualizations. *arXiv:1701.00921v1 [cs.CG]* (2017). 19

[New89] NEWBERY F.: Edge concentration: A method for clustering directed graphs. *ACM SIGSOFT Software Engineering Notes 14*, 7 (1989), 76–85. 1, 3, 19

[NHE11] NGUYEN Q., HONG S.-H., EADES P.: TGI-EB: A new framework for edge bundling integrating topology, geometry and importance. In *Proc. Graph Drawing* (2011), Springer, pp. 123–135. 4, 5, 7, 10, 11, 12, 13, 19

[NPD16] NAGEL T., PIETSCH C., DÖRK M.: Staged analysis: From evocative to comparative visualizations of urban mobility. In *Proc. IEEE Visualizaton (Arts Program)* (2016), pp. 23–30. https://uclab.fh-potsdam.de/cf. 2, 7

[PBO*14] PALMAS G., BACHYNSKYI M., OULASVIRTA A., SEIDEL H.-P., WEINKAUF T.: An edge-bundling layout for interactive parallel coordinates. In *Proc. IEEE PacificVis* (2014), pp. 66–64. 1, 4, 9, 18

[PCJ95] PURCHASE H. C., COHEN R. F., JAMES M.: Validating graph drawing aesthetics. In *Proc. Graph Drawing* (1995), pp. 221–232. 19

[PHT15] PEYSAKHOVICH V., HURTER C., TELEA A.: Attribute-driven edge bundling for general graphs with applications in trail analysis. In *Proc. IEEE PacificVis* (2015), pp. 39–46. 1, 2, 4, 9, 10, 11, 12, 15, 16, 17

[PNK10] PUPYREV S., NACHMANSON L., KAUFMANN M.: Improving layered graph layouts with edge bundling. In *Proc. Graph Drawing* (2010), Springer, pp. 329–340. 4, 5, 11, 15, 19

[PT97] PIEGL L., TILLER W.: *The NURBS Book*. Springer, 1997. 2nd edition. 9, 10, 11

[PVH*03] POST F. H., VROLIJK B., HAUSER H., LARAMEE R. S., DOLEISCH H.: The state of the art in flow visualisation: Feature extraction and tracking. *CGF 22*, 4 (2003), 775–792. 18

[PW16] PALMAS G., WEINKAUF T.: Space bundling for continuous parallel coordinates. In *Proc. EuroVis (Short Papers)* (2016). 1, 4, 9

[PXY*05] PHAN D., XIAO L., YEH R., HANRAHAN P., WINOGRAD T.: Flow map layout. In *Proc. InfoVis* (2005), pp. 219–224. 1, 2, 3, 4, 6, 10, 11, 19

[QZW06] QU H., ZHOU H., WU Y.: Controllable and progressive edge clustering for large networks. In *Proc. Graph Drawing* (2006), Springer, pp. 399–404. 4, 11

[RDLC12] RICHE N. H., DWYER T., LEE B., CARPENDALE S.: Exploring the design space of interactive link curvature in network diagrams. In *Proc. ACM AVI* (2012), pp. 506–513. 14

[RFFT17] RAUBER P. E., FADEL S., FALCAO A., TELEA A.: Visualizing the hidden activity of artificial neural networks. *IEEE TVCG 23*, 1 (2017), 101–110. 1, 18

[RRAS08] ROYER L., REIMANN M., ANDREOPOULOS B., SCHROEDER M.: Unraveling protein networks with power graph analysis. *PLOS Computational Biology 4*, 7 (2008), 1–17. 6

[RVET14] RENIERS D., VOINEA L., ERSOY O., TELEA A.: The Solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product. *Science of Computer Programming 79* (2014), 224–240. 4, 5, 6, 7, 10, 12, 14, 15

[RVT11] RENIERS D., VOINEA L., TELEA A.: Visual exploration of program structure, dependencies and metrics with SolidSX. In *Proc. IEEE VISSOFT* (2011). 5, 12

[Sch07] SCHAEFFER S.: Survey: Graph clustering. *Computer Science Review 1*, 1 (2007), 27–64. 3, 5

[Set02] SETHIAN J.: *Level Set Methods and Fast Marching Methods*. Cambridge Univ. Press, 2002. 11

[SFPY07] SUN J., FALOUTSOS C., PAPADIMITRIOU S., YU P.: Graphscope: parameter-free mining of large time-evolving graphs. In *Proc. ACM KDD* (2007), pp. 687–696. 7

[SH13] SCHULZ H.-J., HURTER C.: Grooming the hairball-how to tidy up network visualizations? In *Proc. IEEE InfoVis (tutorials)* (2013). 1, 2

[SHH11] SELASSIE D., HELLER B., HEER J.: Divided edge bundling for directional network data. *IEEE TVCG 19*, 12 (2011), 754–763. 1, 4, 6, 10, 11, 12, 15, 16, 19

[Shn96] SHNEIDERMAN B.: The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. IEEE Symp. on Visual Languages* (1996), pp. 336–343. 14

[SHVDWVW16] SCHEEPENS R., HURTER C., VAN DE WETERING H., VAN WIJK J. J.: Visualization, selection, and analysis of traffic flows. *IEEE TVCG 22*, 1 (2016), 379–388. 9, 10, 16

[SL87] SIMON H. A., LARKIN J. H.: Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science 11* (1987), 65–100. 17, 19

[SP09] SIDDIQI K., PIZER S.: *Medial Representations: Mathematics, Algorithms and Applications*. Springer, 2009. 1, 9

[STT81] SUGIYAMA K., TAGAWA S., TODA M.: Methods for visual understanding of hierarchical system structures. *IEEE Trans Sys Man Cyber 11*, 2 (1981), 109–125. 3, 5

[SVPM14] SORZANO C., VARGAS J., PASCUAL-MONTANO A.: A survey of dimensionality reduction techniques, 2014. arxiv.org/pdf/1403.2877. 17

[SvW08] SHRINIVASAN Y. B., VAN WIJK J. J.: Supporting the analytical reasoning process in information visualization. In *Proc. ACM CHI* (2008), pp. 1237–1246. 20

[SvW09] SHRINIVASAN Y. B., VAN WIJK J. J.: Supporting exploration awareness in information visualization. *IEEE Computer Graphics and Applications 29*, 5 (2009), 34–43. 20

[SWvdW*11] SCHEEPENS R., WILLEMS N., VAN DE WETERING H., ANDRIENKO G., ANDRIENKO N., VAN WIJK J. J.: Composite density maps for multivariate trajectories. *IEEE TVCG 17*, 12 (2011), 2518–2527. 2, 9, 10, 12, 16

[SWvdWvW11] SCHEEPENS R., WILLEMS N., VAN DE WETERING H., VAN WIJK J. J.: Interactive visualization of multivariate trajectory data with density maps. In *Proc. IEEE PacificVis* (2011), pp. 147–154. 12, 16

[Sze10] SZELISKI R.: *Computer Vision: Algorithms and Applications*. Springer, 2010. 3

[T*17] TU S., ET AL.: The Protégé ontology editor, 2017. http://protege.stanford.edu. 19

[TA08] TELEA A., AUBER D.: Code flows: Visualizing structural evolution of source code. *CGF 3*, 27 (2008), 831–838. 4, 5, 9, 15

[TAvHS06] TOMINSKI C., ABELLO J., VAN HAM F., SCHUMANN H.: Fisheye tree views and lenses for graph visualization. In *Proc. Information Visualisation* (2006), pp. 202–210. 4, 14

[TBET99] TOLLIS I., BATTISTA G. D., EADES P., TAMASSIA R.: *Graph drawing: Algorithms for the visualization of graphs*. Prentice Hall, 1999. 2, 5

[TDT13] TRÜMPER J., DÖLLNER J., TELEA A.: Multiscale visual comparison of execution traces. In *Proc. IEEE ICPC* (2013). 4, 5, 10, 13, 15

[TE10] TELEA A., ERSOY O.: Image-based edge bundles: Simplified visualization of large graphs. *CGF 29*, 3 (2010), 543–551. 1, 2, 4, 5, 10, 12, 14, 17

[TEHR09] TELEA A., ERSOY O., HOOGENDORP H., RENIERS D.: Comparison of node-link and hierarchical edge bundling layouts: A user study. In *Dagstuhl Seminar Proceedings 09211* (2009). 15, 19

[Tel15] TELEA A.: *Data Visualization: Principles and Practice*. CRC Press, 2015. 2nd edition. 9, 15, 18

[Tob81] TOBLER W.: Depicting federal fiscal transfers. *The Professional Geographer 33*, 4 (1981), 419–422. 6

[TP15] THÖNY M., PAJAROLA R.: Vector map constrained path bundling in 3D environments. In *Proceedings of the 6th ACM SIGSPATIAL International Workshop on GeoStreaming* (2015), ACM, pp. 33–42. 9, 16, 17

[Tuf92] TUFTE E. R.: *The Visual Display of Quantitative Information*. Graphics Press, 1992. 1, 3, 6, 19, 20

[TvW99] TELEA A., VAN WIJK J. J.: Simplified representation of vector fields. In *Proc. IEEE Visualization* (1999), pp. 35–42. 18

[TvW02] TELEA A., VAN WIJK J. J.: An augmented fast marching method for computing skeletons and centerlines. In *Proc. VisSym* (2002), pp. 251–259. 11

[TWHW07] TSAI A., WESTIN C.-F., HERO A. O., WILLSKY A. S.: Fiber tract clustering on manifolds with dual rooted-graphs. In *Proc. IEEE CVPR* (2007). 9

[TWK*10] TATLER B., WADE N., KWAN H., FINDLAY J., VELICHOVSKY B.: Yarbus, eye movements, and vision. *i-Perception 1* (2010), 7–27. 17, 19

[VBS11] VERBEEK K., BUCHIN K., SPECKMANN B.: Flow map layout via spiral trees. *IEEE TVCG 17*, 12 (2011), 2536–2544. 4, 6, 10, 11

[vdMH08] VAN DER MAATEN L., HINTON G.: Visualizing data using t-SNE. *Journal of Machine Learning Research 9* (2008), 2579–2605. 18

[vdZCT16] VAN DER ZWAN M., CODREANU V., TELEA A.: CUBu: Universal real-time bundling for large graphs. *IEEE TVCG 22*, 12 (2016), 2550–2563. 1, 2, 4, 9, 11, 12, 13, 16, 17, 19, 20

[vHP09] VAN HAM F., PERER A.: Search, show context, expand on demand: Supporting large graph exploration with degree-of-interest. *IEEE TVCG 15*, 6 (2009), 953–960. 4

[vHW08] VAN HAM F., WATTENBERG M.: Centrality based visualization of small world graphs. *Computers & Graphics 27*, 3 (2008), 975–982. 10

[vLdL03] VAN LIERE R., DE LEEUW W.: GraphSplatting: Visualizing graphs as continuous fields. *IEEE TVCG 9*, 2 (2003), 206–212. 12

[VT14] VOINEA L., TELEA A.: Visual clone analysis with SolidSDD. In *Proc. IEEE VISSOFT* (2014). 15

[vW02] VAN WIJK J. J.: Image based flow visualization. *Proc. ACM TOG (SIGGRAPH) 21*, 3 (2002), 745–754. 13

[VWvdW99] VAN WIJK J. J., VAN DE WETERING H.: Cushion treemaps: Visualization of hierarchical information. In *Proc. IEEE InfoVis* (1999), pp. 73–82. 12

[WC07] WONG N., CARPENDALE S.: Supporting interactive graph exploration using edge plucking. In *Proc. SPIE* (2007), pp. 235–246. 4, 14

[WCG03] WONG N., CARPENDALE S., GREENBERG S.: EdgeLens: an interactive method for managing edge congestion in graphs. In *Proc. IEEE InfoVis* (2003), pp. 51–58. 2, 4

[WDS10] WOOD J., DYKES J., SLINGSBY A.: Visualisation of origins, destinations and flows with od maps. *The Cartographic Journal 47*, 2 (2010), 117–129. 6

[WF94] WASSERMAN S., FAUST K.: *Social network analysis: Methods and applications*. Cambridge University Press, 1994. 10

[WHS90] WISE J. A., HOPKIN V. D., SMITH M. L.: Automation and systems issues in air traffic control. In *Proc. NATO Advanced Study Institute on Automation and Systems* (1990), Springer. 16

[WL07] WETTEL R., LANZA M.: Program comprehension through software habitability. In *Proc. IEEE ICPC* (2007). 7, 11, 15

[Wol07] WOLFF A.: Drawing subway maps: A survey. *Computer Science – Research and Development 22*, 1 (2007), 23–44. 6

[WPCM02] WARE C., PURCHASE H., COLPOYS L., MCGILL M.: Cognitive measurements of graph aesthetics. *Information Visualization 1* (2002), 103–110. 19

[WYY15] WU J., YU L., YU H.: Texture-based edge bundling: A web-based approach for interactively visualizing large graphs. In *Proc. IEEE Big Data* (2015), pp. 1230–1237. 9

[YDGM17] YANG Y., DWYER T., GOODWIN S., MARRIOTT K.: Many-to-many geographically-embedded flow visualisation: An evaluation. *IEEE TVCG 23*, 1 (2017), 411–420. 6

[YMSJ06] YI J., MELTON R., STASKO J., JACKO J.: Dust & magnet: Multivariate information visualization using a magnet metaphor. *Information Visualization 4*, 4 (2006), 542–551. 13

[YWSC12] YU H., WANG C., SHENE C.-K., CHEN J. H.: Hierarchical streamline bundles. *IEEE TVCG 18*, 8 (2012), 1353–1365. 1, 3, 4, 9, 18

[ZWHK16] ZIELASKO D., WEYERS B., HENTSCHEL B., KUHLEN T. W.: Interactive 3D force-directed edge bundling. *CGF 35*, 3 (2016), 51–60. 7, 10, 11

[ZXYQ13] ZHOU H., XU P., YUAN X., QU H.: Edge bundling in information visualization. *Tsinghua Science and Technology 18*, 2 (2013), 145–156. 1, 2, 4

[ZYC*08] ZHOU H., YUAN X., CUI W., QU H., CHEN B.: Energy-based hierarchical edge clustering of graphs. In *Proc. IEEE PacificVis* (2008), pp. 55–61. 15

[ZYQ*08] ZHOU H., YUAN X., QU H., CUI W., CHEN B.: Visual clustering in parallel coordinates. *CGF 27*, 3 (2008), 1047–1054. 4, 9, 18