DOI: 10.1111/cgf.14789

COMPUTER GRAPHICS *forum*
Volume 42 (2023), number 6, e14789

Faster Edge-Path Bundling through Graph Spanners

Markus Wallinger,[1] Daniel Archambault,[2] David Auber,[3] Martin Nöllenburg[1] and Jaakko Peltonen[4]

[1]TU Wien, Vienna, Austria
{mwallinger, noellenburg}@ac.tuwien.ac.at
[2]Swansea University, Swansea, UK
d.w.archambault@swansea.ac.uk
[3]University of Bordeaux, Bordeaux, France
david.auber@u-bordeaux.fr
[4]Tampere University, Tampere, Finland
jaakko.peltonen@tuni.fi

Abstract

Edge-Path bundling is a recent edge bundling approach that does not incur ambiguities caused by bundling disconnected edges together. Although the approach produces less ambiguous bundlings, it suffers from high computational cost. In this paper, we present a new Edge-Path bundling approach that increases the computational speed of the algorithm without reducing the quality of the bundling. First, we demonstrate that biconnected components can be processed separately in an Edge-Path bundling of a graph without changing the result. Then, we present a new edge bundling algorithm that is based on observing and exploiting a strong relationship between Edge-Path bundling and graph spanners. Although the worst case complexity of the approach is the same as of the original Edge-Path bundling algorithm, we conduct experiments to demonstrate that the new approach is ***5–256*** *times faster than Edge-Path bundling depending on the dataset, which brings its practical running time more in line with traditional edge bundling algorithms.*

Keywords: computational geometry, information visualization, visualization

CCS Concepts: • Human-centred computing → Graph drawings; • Theory of computation → Sparsification and spanners

1. Introduction

Edge-Path bundling [WAA*22] is a recent edge bundling approach that is designed to avoid any *independent edge ambiguities*, which are known to occur in edge bundling if two edges from different connected components of a graph are grouped together into a common edge bundle. Such independent edge ambiguities create the false illusion of all-to-all connectivity between the two sets of endpoints on either side of an edge bundle, even if two edges in the bundle belong to different connected components. Edge-Path bundling is free of independent edge ambiguities by bundling long edges against existing alternative paths between their endpoints. Hence, by design, no two independent edges from different components can ever be bundled against the same path. Given a particular drawing of a graph, in the undirected case, this procedure is equivalent to bundling the cycles and in the directed case, it is equivalent to bundling a directed edge with a directed path that starts at the source vertex of the edge and ends at its target. Note that in an Edge-Path bundling, there can still be other visual ambiguities, *e.g.* when two bundles or edges cross at shallow angles. But even straight-line drawings with no bundling at all suffer from such visual edge ambiguities due to shallow angle crossings. While Edge-Path bundling does not incur independent edge ambiguities, it is computationally expensive, posing a barrier for its application to larger networks. This paper presents an approach to improve the performance of Edge-Path bundling, with two main contributions: a new Edge-Path bundling invariant, and a new algorithm for Edge-Path bundling that improves its runtime performance.

A new invariant. The original Edge-Path bundling was the first algorithm to compute a bundling using the Edge-Path primitive. All bundling algorithms turn an input drawing D_G into a bundled drawing Γ. In all previous algorithms, edges in a drawing were bundled with other edges. In the case of Edge-Path bundling, long edges are bundled along paths in a particular drawing—the Edge-Path primitive. Many other possible algorithms can be written that use the

Edge-Path primitive to produce what we define as a *valid Edge-Path bundling*: a bundled drawing without independent edge ambiguities. In this paper, we demonstrate an invariant of all Edge-Path bundlings that can be applied in conjunction with any algorithm that produces a valid Edge-Path bundling. The property is that *each biconnected component of the network can be processed separately, without influencing the resulting Edge-Path bundling*. Although a biconnected component decomposition will not change the worst-case complexity of Edge-Path bundling, it will limit the search of shortest path computations to within a biconnected component because primitives cannot cross biconnected components.

A new efficient algorithm. Second, we introduce a new Edge-Path bundling algorithm based on *graph spanners* that reduces the practical running time of computing Edge-Path bundlings making them competitive with previous work. A graph t-spanner is a connected sub-graph that preserves shortest path distances between any pair of vertices within a factor of $t \geq 1$ [PS89]. The research presented in our paper demonstrates an important connection between graph spanners and computing an Edge-Path bundling with bounded distortion: all the edges of a graph spanner will form part of a path primitive for Edge-Path bundling and conversely the set of locked edges in the original Edge-Path bundling algorithm [WAA*22] forms a graph spanner as long edges are only bundled against paths whose length is below a selected distortion factor.

Accordingly, there exists an Edge-Path bundling algorithm equivalent for every algorithm that computes a graph spanner. Here we propose an algorithm based on the greedy spanner algorithm [ADD*93]. For a graph $G = (V, E)$ and a particular drawing D_G of it to be bundled, our invariant and new algorithm (S-EPB) still have a worst-case time complexity of $O(|E|^2 \log |V|)$, but the practical runtime performance is **5**–**256** times faster than EPB [WAA*22] depending on the dataset. Moreover, its bundling quality in terms of the three originally proposed metrics *ink ratio*, *distortion* and *ambiguity* is comparable. Our spanner approach does not incur independent edge ambiguities and has performance similar to previous CPU-based edge bundling approaches [LBA10, HvW09]. However, all previous approaches, other than EPB and our new approach, incur independent edge ambiguities.

2. Related Work

Edge bundling [LHT17b, Hol06] and confluent drawings [DEGM05] have been the main approaches to reduce edge clutter in dense, non-planar drawings of graphs. Edge-bundling approaches turn an input drawing D_G into a bundled drawing Γ by clustering individual edges (or parts of them) together, either explicitly or implicitly, reducing the visual clutter. Edge bundling approaches suffer from *independent edge ambiguities* when they group edges from separate connected components into a single bundle that may create the (false) impression of a fully connected graph between the endpoints of the bundle (a biclique). Confluent drawings, as a theoretically motivated counterpart of edge bundling, consider only bicliques for bundling, meaning that by design they do not suffer from independent edge ambiguities. However, confluent drawings do not exist for all graphs, at least according to their original planar definition [DEGM05], and suffer from a low degree of bundling in real graphs. Edge-Path bundling [WAA*22], in contrast to both approaches, given a drawing D_G bundles a long edge $(u, v) \in E$ in the graph with an alternative path between u and v that is below a threshold deviation from the Euclidean distance of their straight-line connection. In the case of directed graphs, it bundles a directed edge (u, v) along a directed path from u to v. In the case of undirected graphs, the approach bundles cycles (usually with one long edge). This definition of the Edge-Path primitive is essentially the same as Equation (1) in Lhuillier *et al.* [LHT17b] for groups of edges, but we use distortion to play the role of δ and κ in their definition.

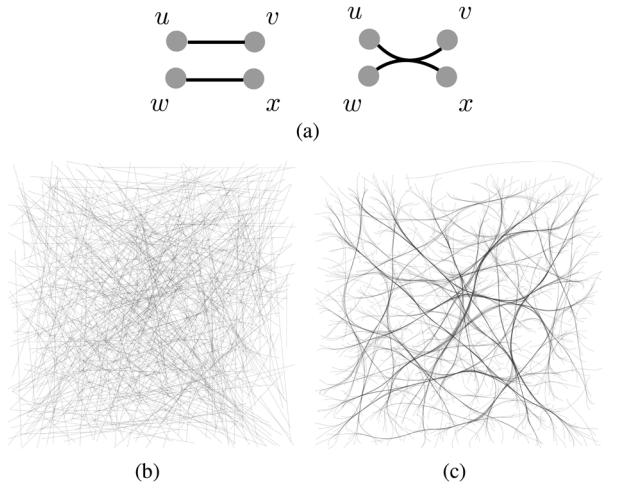


Figure 1: *Illustration of the effect of independent edge ambiguities. (a) Bundling two independent edges together incurring a false connection between u and x when there is no such connection. (b) A collection of random disconnected edges. As there is no signal, there should not be a pattern in the bundling. (c) A bundling of this graph with Winding Roads [LBA10]. The image clearly depicts a pattern where there is none. Edge-Path bundling algorithms, like the ones described in this paper, will not suffer from this drawback by definition and will not bundle this graph.*

Independent edge ambiguities, see Figure 1a, have been noted as an issue for over a decade [SHH11, LLCM12], leading to solutions such as the one we describe here. These ambiguities do cause concerns for visualizations when connectivity and graph topology is of interest. However, for certain tasks and data, this is not of interest. Consider *trail-sets* for which the location of vertices cannot be changed (*e.g.* vehicle positions, locations of cities) and we are more concerned about patterns in movement represented by the direct connections of the edges. The independent edge ambiguity is not a concern in this case and edges heading in the same direction should be grouped together, similar to a wire tie around computer cables. Furthermore, spatial trail-sets can be combined with a secondary structure that constrains the bundling. For example, origin–destination trails of car traffic and a road network. The additional information that is available can be used to infer which edges in the trail-set should and should not be bundled.

In this paper, we aim to bundle drawings where the independent edge ambiguities matter for the visualization task and where patterns in connectivity beyond direct connections are important.

Consider Figure 1b, which contains a collection of random disconnected edges. Seeing it as a graph, there is no pattern in this data. However, all edge bundling algorithms, with the exception of confluent drawing algorithms, EPB, and this work, would find misleading structure in this data (see Figure 1c).

We now describe related work in these areas and conclude with a discussion on graph spanners as we demonstrate a strong connection between Edge-Path bundling and graph spanners.

Edge bundling. Since its introduction [Hol06], many edge bundling algorithms have been proposed and evaluated. The first algorithm required a hierarchy [Hol06] where internal vertices of the hierarchy were used as control points. This requirement for a hierarchy was relaxed and a number of algorithms were written based on a variety of different methods: grids and quad trees [LBA10, LLCM12], triangular meshes [CZQ*08], force-directed approaches [HvW09, NHE11, SHH11], visibility spanners [PNBH16], multilevel approaches [GHNS11] and cluster hierarchies [TNI*17, LDB11, RVET14]. Image-based approaches [TE10, HET12, vCT16, EHP*11, WYY15, LHT17a], which do not require hierarchies either, greatly accelerated the bundling of edges by operating on the pixels of individual edges. A density or similarity map is created at the pixel level by summing up the contributions of all edges, after which the edges are independently advected upstream along gradients of the map.

Furthermore, in visualizations of origin–destination trail sets, constrained bundling approaches are commonly applied. Here, a secondary data structure is used to constrain the bundling. For example, vector maps are used to compute a bundling of the trail set along paths in the secondary graph [TP15] or road networks are used to estimate optimal parameters and adapt the kernel density estimation of KDEEB [ZSJT19]. One advantage of the approach in Zeng *et al.* [ZSJT19] is to morph [LLC*20] between a faithful representation of trails according to the road network and a visually simplified representation. In contrast, Edge-Path bundling does not need a secondary data structure and purely uses elements of the input graph for bundling. More precisely, our approach bundles long direct edges against shortest paths in the network, which typically pass through few nodes of high centrality.

Edge bundling has been a topic of active research for the past one and a half decades [LHT17b] with many approaches proposed. However, all of these edge bundling approaches suffer from the independent edge ambiguity which can bundle edges in different connected components together, causing false connections in the graph to appear. This issue can have significant consequences in networks where it creates signal where there is none (see Lhuillier *et al.* [LHT17b] fig. 19 and Figure 1). The approach presented in this paper, as in the work of Wallinger *et al.* [WAA*22], does not suffer from independent edge ambiguities, greatly reducing the ambiguity of bundled drawings.

Confluent drawings. Confluent drawings were introduced by Dickerson *et al.* [DEGM05] as a way of creating planar layouts of non-planar graphs using the existence of smooth paths in a crossing-free system of terminals (the vertices) and junctions, as well as arcs between pairs of terminals or junctions as the representation of the edges of the graph. Due to this property, confluent drawings are sometimes described as train track layouts, where two points u, v are connected if and only if a forward-moving train can reach v when starting in u [HPSŠ07]. A confluent edge bundle thus implies full connectivity (a biclique) between the vertices on one side with the vertices on the other side. A number of variations of confluent drawings have been studied in the literature, among them strong confluence [HPSŠ07], strict confluence [EHL*16] and Δ-confluence [EGM06]. From a theoretical point of view, characterization and recognition problems are of interest, and it is known that large classes of non-planar graphs admit planar confluent drawings, there are also graphs that do not have planar confluent drawings [DEGM05, FGKN19] and, for instance, recognizing graphs with strict confluent drawings is NP-complete [EHL*16]. More practical approaches [BRH*17, ZPG21] that relax some of the strict constraints of confluent drawings have also been investigated, but they assume full freedom to place or re-position vertices, which is usually not permitted for spatial graphs whose vertex positions encode meaningful information.

Edge-Path bundling. The original Edge-Path bundling algorithm [WAA*22] has the main advantage of combining a higher degree of bundling than confluent drawings without introducing independent edge ambiguities; the approach never bundles two edges from different connected components together and therefore will not create false connections between endpoints from different components. However, the resulting bundles can still suffer from ambiguities when edges or bundles cross at a flat angle close to 180°. The approach achieves this property by bundling edges along paths in the graph creating Edge-Path primitives. A path does not necessarily have a semantic meaning; only the positions of the vertices along the path are used as a structure to infer control points to bundle the edge. If sufficiently many Edge-Path primitives use similar paths in the graph, the visual complexity of the drawing is reduced. This approach works well on graphs of thousands of edges and is very easy to implement, with re-implementations available [epb, HMDMAGB22]. However, the approach is computationally not as scalable as other edge bundling approaches in the literature. This paper improves upon the computational performance and scalability of Edge-Path bundling approaches.

Graph spanners. Our new Edge-Path bundling approach heavily uses graph spanners to determine the paths that long edges will be bundled against. Graph spanners are sparsification methods for a graph to create spanning sub-graphs that do not affect shortest path distances between vertices too much, increasing the speed of distance approximation. A (multiplicative) t-spanner of a graph $G = (V, E)$ with a parameter $t \geq 1$ is a spanning sub-graph $H = (V, E')$ of G with $E' \subset E$ and the property that for any pair $u, v \in V$ their shortest distance in H is at most a factor of t (called the *stretch factor*) from their original distance in G, *i.e.* $d_H(u, v) \leq t d_G(u, v)$. The proposed algorithm in this paper uses graph spanners as a basis for computing an Edge-Path bundling. Representative sparsified sub-graphs have been used for visualizing and drawing graphs before [AAM06, vHW08, NOB15], whereas here we use graph spanners specifically for bundling. Graph spanners have been investigated for decades [Awe85, PS89] and a recently published survey [ABS*20] gives a detailed overview of different types of graph spanners and their respective theoretical bounds. We only consider multiplicative t-spanners in this paper and will continue to use the

term graph spanner to refer to this concept. Finding the sparsest (fewest edges) or lightest (minimum total edge length) spanner has been proven to be NP-hard [PS89, Cai94]. On the positive side, Integer Linear Programming (ILP) formulations exist that can provide exact solutions, but practically they only work on small graphs of up to 100 vertices [SZ04, AHJ*19]. However, as sparsification with spanners is important in practical applications [Cow01, CW04, SVZ07, SS10], heuristic approaches have been published. The first greedy algorithm was introduced by Althöfer *et al.* [ADD*93] which was later improved to be less computationally expensive [FG09, FG07, BCF*10], less memory intensive [ABtBB15] or sacrificing either sparseness, lightness or runtime to improve another property [RZ11, ADF*19, TZ05, ADF*19, ES16]. Approximation algorithms [KP94, DN97] started to appear shortly after the greedy algorithm and have been subsequently refined [EP01, BGJ*12, DK11, BBM*13, DZ16]. Similarly, probabilistic approaches [EN19, MPVX15, BS07] have been explored. Although both classes of algorithms are asymptotically faster, the guarantee for sparseness and lightness does not improve when compared to the greedy algorithm, and they do not generalize well to arbitrary graphs. Recently, an experimental evaluation [CS21] compared different spanner algorithms on runtime, sparsity, and lightness to clarify practical implications; they confirmed that the greedy algorithm [ADD*93] provides the best balance of these three criteria.

In this paper, we show a strong connection between computing a t-spanner and an Edge-Path bundling. Our algorithm is based on the greedy approach [ADD*93] and produces faster practical performance when compared to the original Edge-Path bundling algorithm [WAA*22].

3. Algorithm

In this paper, we describe a new approach to Edge-Path bundling, which includes two independent steps. The algorithm takes a graph $G = (V, E)$ and an input drawing D_G. Our first step decomposes the graph G into its biconnected components. Each component can be processed separately as an Edge-Path primitive will never span two biconnected components (Section 3.1). This decomposition can be applied in conjunction with any Edge-Path bundling algorithm. Our second step is a new algorithm for Edge-Path bundling based on multiplicative t-spanners (Section 3.2).

3.1. Process biconnected components independently

The first step divides the graph into smaller components, the biconnected components of the graph, that can be bundled independently without changing the Edge-Path bundling of the drawing. A biconnected component in a graph G is a maximal biconnected sub-graph H of G, *i.e.* the removal of any single vertex in H leaves H still connected. A connected graph can be decomposed in $O(|V| + |E|)$ time into its biconnected components, which are linked together at *cut vertices*, *i.e.* vertices whose removal from G increases the number of connected components. Figure 2 shows an example of a graph with its three biconnected components. Edge-Path bundling bundles cycles in the undirected case, and a directed path with an edge in the same direction in the directed case. In either case, the Edge-Path pair forms an underlying cycle (possibly ignoring the edge directions),

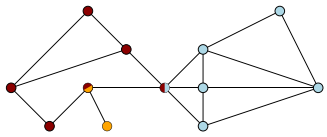


Figure 2: *Biconnected component decomposition of a graph (indicated using colour). A graph is split into three biconnected components at the respective cut vertices. Each component can be processed individually as there are no cycles between components.*

which is by definition biconnected. Hence any Edge-Path pair for bundling must be contained within a single biconnected component and we can process the biconnected components of G separately. Although this does not provide a guaranteed improvement in worst case complexity (the input graph may already be biconnected), it will restrict the search radius of the shortest path computations to be contained within the individual biconnected components improving the running time in many cases.

3.2. Sparsification with multiplicative t-spanner

The bottleneck, in terms of practical running time, of the Edge-Path bundling [WAA*22] algorithm is the computation of the shortest path in a dynamically updated graph using Dijkstra's shortest path algorithm, which is implemented in $O(|E| \log |V|)$ runtime. This step is repeated for each potentially bundled edge, which means that the worst case computational complexity of the algorithm is $O(|E|^2 \log |V|)$ as shortest path calculations are repeated $|E|$ times. For each vertex during the path computation, all outgoing edges are explored as they could potentially improve the length of the shortest path. Hence reducing the total number of edges in the graph will generally result in faster computation of paths. To obtain such a sparsified graph, we first calculate a corresponding t-spanner, which we later use to calculate shortest paths that will be used to bundle edges.

Recall that a (multiplicative) t-spanner of an edge-weighted graph $G = (V, E)$ is a sub-graph $H = (V, E')$ with the property that for any $u, v \in V$, the graph distance in H increases by at most a factor of t, *i.e.* $d_G(u, v) \leq t d_H(u, v)$. The graph distance $d_G(u, v)$ is defined as the length of a shortest path between u and v in G using the given edge weights as edge lengths. In a geometric graph, edge weights are usually considered as the Euclidean distances between the endpoints.

Informally, we may exclude an edge (u, v) when constructing a spanner if there is already an alternative path with stretch less than t between vertices u and v. Recall that in the Edge-Path bundling algorithm [WAA*22], an edge was only bundled against a path between its endpoints if the length of this path was less than the length the edge times a distortion factor of k. This definition and the definition of t-spanner are equivalent in the following sense: the graph induced by the unbundled edges in the result of Edge-Path bundling is a k-spanner of the input graph as we have an alternative path of stretch at most k for every bundled edge. Conversely, we can bundle all the edges that were excluded from a t-spanner along their shortest paths in the t-spanner and guarantee a bundling distortion of at most t.

Algorithm 1. Greedy spanner algorithm [ADD*93].

Input: Graph $G = (V, E)$, edge-weights $w : E \to \mathbb{R}^+$, stretch factor $t > 1$
Output: t-spanner $H = (V, E' \subseteq E)$
$E' \leftarrow \emptyset$
$H \leftarrow (V, E')$
sortedEdges $\leftarrow$ sortAscending (E, w)
for $e = (u, v) \in$ sortedEdges **do**
  $p \leftarrow$ shortestPath (H, u, v)
  if p.length() $> t * w(u, v)$ **then**
    $E' \leftarrow E' \cup \{e\}$
return $H = (V, E')$

Algorithm 2. Spanner-Edge-Path bundling algorithm.

Input: Graph $G = (V, E)$, input drawing D_G, maximum distortion $t > 1$, edge weight parameter κ.
Output: Control points for an Edge-Path bundled drawing Γ.
$H = (V, E') \leftarrow$ greedySpanner$(G, ||\cdot||, t)$
for $e \in E'$ **do**
  weight$(e) \leftarrow ||e||^\kappa$
for $e = (u, v) \in E \setminus E'$ **do**
  $p \leftarrow$ shortestPath $(H, u, v,$ weight$)$
  if p.length() $\leq t * ||e||$ **then**
    controlPoints$(e) \leftarrow p$.getVertexCoords()
return controlPoints

As mentioned previously, the sparseness of the spanner H reduces the overall runtime as less edges need to be explored when computing shortest paths; thus, sparser spanners having faster runtimes. However, finding the sparsest spanner—a minimum number of edges in E'—is NP-hard [PS89, Cai94]. Several works have been published that present algorithms for efficiently constructing (non-optimal) t-spanner for directed and undirected, weighted graphs. We mainly focus on the greedy approach by Althöfer *et al.* [ADD*93]; other approaches are asymptotically faster but practically less efficient [FS20], or have no guarantees on sparseness that are competitive with the greedy approach [CS21].

Greedy spanner algorithm. Arguably the most well-studied algorithm to construct a spanner is the greedy algorithm [ADD*93]. We first discuss the basic implementation, shown in Algorithm 1, before we introduce concepts from existing literature that can further optimize this approach.

Given an edge-weighted input graph $G = (V, E)$ and a stretch factor $t > 1$, we start by creating a graph $H = (V, E')$ with the same vertex set V and an initially empty edge set E'. Next, we sort the edges $e \in E$ in increasing order by their edge weights $w(e)$ (*e.g.* their Euclidean length) and process each edge individually in that order. In every iteration, we perform a shortest path computation for the endpoints of the next edge $e = (u, v)$ on the current graph H. If there exists no path between u and v or the length of the computed shortest path $d_H(u, v) > t \cdot w(e)$, we add the edge e to the edge set E'. At the end of the process, the graph H satisfies the t-spanner property by construction.

We observe that the greedy spanner algorithm is agnostic towards directed or undirected graphs and can handle both, meaning we can do both directed and undirected bundling in this way; thus, directed edge bundling can be computed without change to the algorithm.

Optimizing the greedy spanner. In the decades following the publication of the original greedy spanner algorithm, some work introduced techniques to improve the performance of this algorithm while constructing deterministicly the same spanner.

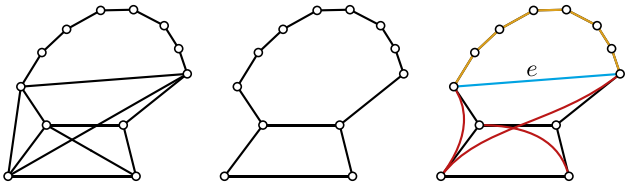
One approach, referred to as FG-greedy [FG07, FG09], decreases the computational cost by storing the known shortest path distances for vertex pairs. Whenever a shortest path between u and v is calculated, we update the distances from u to all vertices found during the path computation. But instead of performing a shortest path computation for every edge, we first check if there exists a stored path that fulfills the stretch constraint and safely dismiss the edge in the positive case. Otherwise, if there is no entry or the length of the path exceeds the stretch, we perform a shortest path computation. Due to the fact that only edges that are known to be shorter than an already explored t-spanner path are dismissed, the algorithm produces the same t-spanner as the original algorithm, but may reduce the required runtime.

Another observation is that the search radius for the shortest path between u and v can be bounded by $t \cdot w(u, v)$ as a longer path will immediately result in adding the edge to the spanner. So practically speaking, we can stop the shortest path search immediately after exceeding this threshold.

3.3. The spanner Edge-Path bundling algorithm

We will now describe our improved spanner-based Edge-Path bundling (S-EPB) algorithm, see Algorithm 2. This algorithm can be applied to an entire graph or, after computing a biconnected component decomposition of the graph, to those components sequentially or in parallel. Similar to Edge-Path bundling [WAA*22], the input is a graph $G = (V, E)$ with a drawing D_G and Euclidean distance $||\cdot||$ as the edge length, a maximum distortion threshold $t > 1$, and a bundling parameter $\kappa \geq 1$. Instead of immediately processing the edges of G, we first construct a t-spanner $H = (V, E')$ based on Euclidean distances as a sparse representation of G. Recall that it is guaranteed by construction of H that each edge uv is either contained in H or there is a u–v path p in H whose length exceeds $||uv||$ by at most the distortion factor t. Once we have computed the graph spanner, we assign to each spanner edge $e \in E'$ a new parameterized weight $w(e) = ||e||^\kappa$ (similar to the approach presented by Wallinger *et al.* [WAA*22]) and store it in a hash set.

Next, we iterate over all non-spanner edges in $E \setminus E'$ to determine their bundling paths in H. For each edge $e = (u, v)$ in this set, we calculate a shortest path p in H using the new edge weights. While for $\kappa = 1$, these edge weights remain the Euclidean edge lengths and p is a Euclidean shortest path in H, for $\kappa > 1$, we give adjustable preference to slightly longer paths with shorter edges over shorter paths with longer edges (from the perspective of the Euclidean metric). If the Euclidean length of that path p exceeds $t \cdot ||e||$, we do not bundle e; otherwise, we assign the vertices of p as control points for drawing the bundled edge e. This guarantees a distortion of at most t for any bundled edge even though we use adjusted edge weights



(a) graph G (b) 1.25-spanner H (c) bundled drawing Γ

Figure 3: *An example graph G (a) with a 1.25-spanner H (b). The computed Edge-Path bundling in (c) shows three bundled edges (red) and one unbundled edge e (blue) whose shortest path (orange) with edge weights $||\cdot||^\kappa$ for $\kappa = 2$ exceeds the maximum distortion of* $1.25||e||$.

to compute the shortest bundling path p. As we show in our experiments, the two parameters t and κ control both the maximum distortion and the bundling strength of the computed drawing. Figure 3 shows an example execution of Algorithm 2.

The worst case time complexity of Algorithm 2 is still $O(|E|^2 \log |V|)$, but it is practically faster than the original Edge-Path bundling algorithm as early stages of spanner computation have sparser graphs and the shortest paths for bundling edges are computed on the spanner H instead of the entire graph G. The graph spanner H can be computed in $O(|E|^2 \log |V|)$ time using Algorithm 1. Although the result of this algorithm is different when compared to the original Edge-Path bundling approach [WAA*22], it is still an Edge-Path bundling algorithm that creates a valid Edge-Path bundling. Cycles are turned into Edge-Path primitives in the undirected case and directed edges are turned into Edge-Path primitives with a directed path from source vertex to target vertex in the directed case.

Optimizing the shortest path computation. Instead of computing a shortest path for each edge $e \in E \setminus E'$, we can reuse the shortest path computation and process multiple edges with one computation. This variation computes paths for Edge-Path primitives by processing all vertices of V iteratively. In each iteration, we first compute a list of neighbours of a vertex u. We only consider vertices which are not connected by an edge in E' as neighbours. Then, we sort this list descending according to the distance between u and the respective neighbour. We perform a Dijkstra's shortest path computation from source vertex u to the first neighbour in the list. Once we found the shortest path, we iterate over the list and process and remove all neighbours where we also have a valid shortest path. If the list is empty, we continue with the next vertex in V. Otherwise, we compute a new shortest path between u and a new unprocessed target neighbour.

4. Experiments

Our experimental evaluation primarily compares Edge-Path bundling (EPB) [WAA*22] to the Spanner Edge-Path bundling (S-EPB) introduced in this paper. We hypothesize that S-EPB is faster and produces a bundling that is of commensurate quality. All datasets, images, algorithm implementation and code to reproduce the experiments can be found in the supplementary material on OSF (osf.io/t4h6j/).

Table 1: *The five datasets used in the experimental section ordered by increasing number of edges.*

Dataset	$\|V\|$	$\|E\|$ undir.	$\|E\|$ dir.	$\|C\|$	$C_{>2}$	$\|c_{\max}\|$
Airlines	235	1297	2101	40	6	191
Migrations	1702	6487	9726	643	14	1008
Airtraffic	1533	16,480	16,494	337	6	1187
Amazon200k	192,976	269,271	–	69,747	3838	104,368
PanamaPapers	743,253	1120,783	–	447,925	19,664	232,971

Additionally, information regarding the size of components $c \in C$ after the biconnected component decomposition is shown. $|C|$ is the number of components and $C_{>2}$ are components with at least three edges. $|c_{\max}|$ is the size of the largest component in terms of number of vertices.

4.1. Datasets and experimental procedure

We test our S-EPB approach on the same datasets used in Wallinger *et al.* [WAA*22] plus one new larger dataset (Panama Papers).

Statistics about the datasets can be seen in Table 1. The biconnected component decomposition decreases the input size for computing a valid Edge-Path bundling drastically. As a graph needs at least three edges to form a valid Edge-Path primitive all components with less than three edges can be ignored. Interestingly, all datasets have similar structural property of many small ($|c_i| < 100$) components and one large component ($c_{\max}$). As mentioned in Section 3.1, we can process biconnected components in parallel. However, due to the aforementioned structure of the graphs, the opportunity for load balanced parallelization is not present.

The experimental setting was the following: all runtime experiments were executed sequentially on an AMD Ryzen 5 5600x, with 3.7-GHz base clock 6-core CPU, but limited to one core. The L1 cache of 64KB and L2 cache of 512KB are available to each core exclusively while the L3 cache of 32 MB is shared among all cores. Thirty-two gigabytes of memory were available on the system; however, peak memory allocation never required that amount. The used operating system was Ubuntu 22.04LTS. Algorithms were implemented in C++ using the Open Graph algorithms and Data structures Framework (OGDF) [CGJ*13]. Graph datastructures, the biconnected component decomposition and Dijkstra's shortest path algorithm were used from OGDF. The code of the implementation was compiled with GCC 11.2.0 for optimized performance with compiler flag -O3.

We measured the wall clock time of the bundling algorithms, *i.e.* including the biconnected component decomposition and assignment of control points but excluding the loading of the graph, calculation of Bézier curves for the curved edges and rendering of the output. Our experiment procedure performed 100 runs of each individual bundling algorithm and we averaged the runtime over all results. Both Amazon200K and Panama Papers were drawn with FM^3 [HJ04] beforehand. As these datasets are large, we only ran the algorithm three times. For measuring memory allocation, we used Ubuntu's *time* command which summarizes system resource usage

of a programme. Here, we were mainly interested in the resident set size which includes heap, stack and shared library memory allocation.

For the ambiguity experiments, we rendered the output exactly the same as presented in Wallinger *et al.* [WAA*22]. For each edge, we exported a list of 50 control points representing a polyline approximation of a cublic Bézier curve.

4.2. Bundling quality metrics

For evaluating the bundling quality, we use the same metrics as Wallinger *et al.* [WAA*22], which we briefly describe below.

Ink reduction. The *ink reduction ratio* of a bundled graph layout is the proportion of its active pixels (which are coloured above some grey value threshold) compared to the active pixels of the unbundled layout. The smaller the ratio, the stronger the ink reduction. Consider an $m \times n$ greyscale bitmap image $I \in \{0, \ldots, 255\}^{m \times n}$ of a bundled graph layout Γ. We define its binarization $I^B \in \{0, 1\}^{m \times n}$ as

$$I^B(i, j) = \begin{cases} 1 & I(i, j) \geq \delta \\ 0 & I(i, j) < \delta \end{cases}, \tag{1}$$

where $\delta \in \{0, 1, \ldots, 255\}$ is a grey value above which we consider a pixel active. Analogously, let $J \in \{0, \ldots, 255\}^{m \times n}$ be the greyscale image of the unbundled layout and J^B its binarization. Then the ink-reduction $\text{ink}_J(I)$ of I with respect to J is defined as

$$\text{ink}_J(I) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} I^B(i, j)}{\sum_{i=1}^{m} \sum_{j=1}^{n} J^B(i, j)}. \tag{2}$$

Distortion. The next metric quantifies the average *distortion* of the edges in a bundled layout Γ compared to their straight-line renderings. For an edge $(u, v) \in E$, we define $||u - v||$ as its Euclidean length and $d_\Gamma(u, v)$ as its length in Γ. The distortion $\text{dist}(\Gamma)$ of layout Γ is calculated as the average distortion of its edges

$$\text{dist}(\Gamma) = \frac{1}{|E|} \sum_{(u,v) \in E} \frac{d_\Gamma(u, v)}{||u - v||}. \tag{3}$$

Ambiguity. The *ambiguity* of a bundled layout aims to quantify how many wrong adjacencies in the underlying graph can be derived or perceived from ambiguous renderings of edge or Edge-Path bundles, and also how wrong they are in terms of graph distance of false neighbours. We first define for each edge $e = (s, t) \in E$ and an endpoint s of e the (visually) reachable neighbour sets of s along e in Γ as $N_\Gamma(s, e) = \{v \in V \mid \exists \text{ ambiguous connection from } s \text{ to } v \text{ in } \Gamma\}$. We say that there is such an ambiguous connection if, for some point p on the curve of e, there is another curve of edge $e' = (u, v)$ that intersects a disk $U_\epsilon(p)$ of radius ϵ around p and the angle between e and e' within $U_\epsilon(p)$ is smaller than a threshold θ. In other words, the curves of edges come very close and form a very flat angle so that the human eye tracing e may inadvertently flip to e' instead.

Now the reachable neighbour sets $N_\Gamma(s, e)$ may contain some true and some false neighbours, where in case of false neighbours, we can classify the degree of being false by a graph distance threshold $\delta \geq 1$. We define the true neighbours as $N_\Gamma^t(s, e) = \{v \in N_\Gamma(s, e) \mid d_G(s, v) \leq \delta\}$ and the false neighbours as $N_\Gamma^f(s, e) = N_\Gamma(s, e) \setminus N_\Gamma^t(s, e)$. Here $d_G(s, v)$ is the hop distance between s and v in G, *i.e.* the length of the shortest unweighted path between s and v in G. For a value of $\delta = 1$, the true neighbours must be direct neighbours of s, whereas for $\delta > 1$, we accept vertices as true neighbours that are at most δ hops away from s in G. We finally define the ambiguity $\text{amb}(\Gamma)$ of bundled layout Γ as

$$\text{amb}(\Gamma) = \frac{\sum_{v \in V} \sum_{e=(v,w) \in E} |N_\Gamma^f(v, e)|}{\sum_{v \in V} \sum_{e=(v,w) \in E} |N_\Gamma(v, e)|}. \tag{4}$$

This value measures the proportion of false neighbours to all neighbours visually implied by Γ, with lower values corresponding to less ambiguous drawings.

4.3. Runtime and memory experiments

We evaluated several S-EPB variants against EPB to determine the runtime and memory behaviour. We also determined the effect of the biconnected component decomposition on the different variants. In the paper, we show images comparing EPB to S-EPB to demonstrate that the quality is maintained with a reduction in practical running time. For completeness, we provide all result images and metrics for all bundling algorithms in the supplementary material.

S-EPB variants. Section 3.2 presents a number of optimizations that can be made to the S-EPB algorithm. We tested two variants with different approaches to compute the spanner (Greedy and FG-Greedy) with the improved Dijkstra shortest path algorithm (V-Dijkstra), as well as a variant that uses FG-Greedy but performs a shortest path computation for each edge (E-Dijkstra). We also ran all experiments once with the biconnected component decomposition and once without biconnected component decomposition. These variations are all meant to improve performance.

Table 2 presents the runtime of these variants and EPB. The biconnected component decomposition drastically decreases the runtime of the Edge-Path bundling computation. The only exception here is the Air Traffic dataset as the largest component contains approximately 80% of all vertices. While all variants are still considerably faster than EPB, the overhead of computing the biconnected component decomposition and the respective sub-graphs neglects most of the speed-up when comparing S-EPB variants with and without biconnected component decomposition

For all datasets, the proposed approach to compute multiple shortest paths, instead of performing a shortest path computation for each edge, improves S-EPB. However, the two variants to compute the spanner are less clear and no variant clearly outperforms the other. This behaviour can be explained with the overhead of tracking the shortest distance between vertices and the correlation between density of the graph and succesful look-ups.

Large datasets. Table 2 shows the results of EPB and S-EPB on the large datasets. Here, we were mainly interested in the scalability of S-EPB. Generally, we see that S-EPB is **5–256** times faster depending on the dataset. S-EPB with and without biconnected component decomposition scales better with increased input data size.

Table 2: *Comparison of different EPB and S-EPB implementations.*

	Undirected			Directed			Large undirected	
Algorithm	Airlines	Migrations	Airtraffic	Airlines	Migrations	Airtraffic	Amazon200k	Panama Papers
Without Bicon. Decomp.								
EPB	34	603	2670	34	599	1543	47.01 min	11.47 h
S-EPB (FG-Greedy, V-Dijkstra)	5	109	195	8	174	231	1.18 min	39.70 min
S-EPB (Greedy, V-Dijkstra)	5	110	194	8	174	232	1.17 min	45.00 min
S-EPB (FG-Greedy, SSSP Dijkstra)	13	269	524	16	346	642	1.21 min	1.17 h
With Bicon. Decomp.								
EPB	32	443	2518	29	449	1522	3.96 min	1.8 h
S-EPB (FG-Greedy, V-Dijkstra)	5	71	176	7	121	220	10.98 s	9.48 min
S-EPB (Greedy, V-Dijkstra)	5	71	176	7	120	220	10.86 s	9.51 min
S-EPB (FG-Greedy, SSSP Dijkstra)	13	175	537	16	248	666	11.42 s	16.61 min

Undirected contains the undirected bundling versions of the small datasets. Directed contains the directed versions of the small datasets. Large undirected contains the Amazon200k and Panama Papers datasets. All runtimes are given in milliseconds except indicated otherwise.

Table 3: *Comparison on the small datasets of Wallinger* et al. *[WAA*22].*

Algorithm	Airlines	Migrations	Airtraffic	Implementation
CUBu	0.01	0.01	0.01	C++; CUDA
KDEEB	0.01	0.03	0.05	C++
S-EPB	0.01	0.11	0.19	C++
S-EPB Biconn. Dec.	0.01	0.07	0.17	C++
Winding Roads	0.35	3.27	7.60	C++
Force-directed	2.34	17.97	32.50	Javascript
EPB Biconn. Dec.	0.44	0.44	2.52	C++
EPB	0.03	0.60	2.67	C++

All runtimes are given in seconds.

Memory. For the panama dataset approximately 1.45 GB of memory are allocated for EPB and 1.55 GB for biconnected S-EPB. A table with the respective memory allocation for each dataset and variant can be found in the supplementary material. Memory consumption does not significantly increase when comparing EPB to any S-EPB variant. The reason here is that most memory is allocated to represent the input with OGDF's graph structure while both, S-EPB and the biconnected component decomposition, only allocate marginally more memory linear in the number of edges.

Comparison to other bundling algorithms. Table 3 compares the original edge path bundling algorithm to the graph spanner approach on the same machine as in the previous study [WAA*22]. Additionally, the original Edge-Path bundling algorithm was re-implemented in C++ using OGDF for increased performance. The S-EPB variant used in comparison computes the spanner with the FG-Greedy algorithm and uses V-Dijkstra to compute shortest paths. As seen in the table, S-EPB is on par with or outperforms all of the other bundling algorithms that are CPU-based. The image- and GPU-based approaches are still the fastest of all approaches, taking less than a 100 ms even for the panama dataset. However, all of these edge bundling algorithms have the independent edge ambiguity. For the approaches that do not have independent edge ambiguities, S-EPB is always faster than EPB by a factor of 5–21 times as it operates on a sparser graph.

Summary. The performance of S-EPB compared to EPB is between **5–256** times faster depending on the dataset. On the two larger datasets, S-EPB was at least **110** times faster and shows increased scalability compared to EPB. The speed-up of S-EPB compared to EPB can mainly be explained by the fact that shortest path calculations are performed on sparse sub-graphs of the input graph. The variance of speed-up between datasets is mainly due to structural differences between the graphs. Especially, sparsity and

Table 4: *Scores of the quality metrics for the undirected real-world datasets and a variety of undirected bundling algorithms.*

	US airlines					Migrations							Air traffic					
	ink_J	dist		amb^1	amb^2	ink_J	dist		amb^1	amb^2	amb^3	amb^4	amb^5	ink_J	dist		amb^1	amb^2	amb^3
Straight-line	1.00	1.00	1.00	0.66	0.03	1.00	1.00	1.00	0.67	0.39	0.13	0.06	0.04	1.00	1.00	1.00	0.54	0.10	0.00
EPB	0.57	1.07	1.03	0.79	0.05	0.58	1.06	1.02	0.68	0.34	0.10	0.05	0.04	0.60	1.10	1.06	0.58	0.14	0.01
S-EPB	0.58	1.08	1.04	0.80	0.05	0.59	1.07	1.03	0.69	0.34	0.10	0.05	0.03	0.63	1.11	1.07	0.58	0.14	0.01
CUBu	0.47	1.08	1.06	0.86	0.05	0.64	1.06	1.05	0.77	0.43	0.15	0.06	0.04	0.71	1.00	1.00	0.59	0.11	0.01
Winding Roads	0.47	1.08	1.06	0.86	0.05	0.58	1.06	1.04	0.75	0.42	0.15	0.07	0.05	0.56	1.06	1.04	0.58	0.12	0.01
Force-directed	0.77	1.04	1.02	0.73	0.04	0.77	1.12	1.06	0.76	0.44	0.15	0.07	0.05	0.79	1.06	1.02	0.54	0.10	0.00

S-EPB has parameters $t = 2, \kappa = 2$. Column dist gives mean and median. Columns amb^δ are only shown for $1 \leq \delta \leq 5$ if there are non-zero entries.

Table 5: *Scores of the quality metrics for the directed real-world datasets and a variety of directed bundling algorithms.*

	US airlines					Migrations						Air traffic						
	ink_J	dist		amb^1	amb^2	ink_J	dist		amb^1	amb^2	amb^3	amb^4	ink_J	dist		amb^1	amb^2	amb^3	amb^4
Straight-line	1.00	1.00	1.00	0.65	0.04	1.00	1.00	1.00	0.70	0.50	0.27	0.19	1.00	1.00	1.00	0.65	0.52	0.47	0.43
EPB	0.64	1.07	1.02	0.80	0.07	0.66	1.06	1.02	0.75	0.51	0.25	0.18	0.81	1.06	1.02	0.68	0.54	0.49	0.45
S-EPB	0.63	1.08	1.03	0.80	0.07	0.68	1.07	1.02	0.75	0.50	0.25	0.18	0.82	1.06	1.02	0.68	0.54	0.49	0.45
CUBu	0.55	1.05	1.03	0.85	0.10	0.64	1.07	1.02	0.79	0.56	0.30	0.21	0.56	1.02	1.01	0.68	0.56	0.50	0.46
Force-directed	0.88	1.11	1.05	0.86	0.10	0.83	1.14	1.07	0.86	0.62	0.33	0.22	0.84	1.60	1.39	0.72	0.57	0.51	0.46

S-EPB has parameters $t = 2, \kappa = 2$. Column dist gives mean and median. Columns amb^δ are only shown for $1 \leq \delta \leq 4$ if there are non-zero entries.

distribution of lengths of shortest paths is an indicator of the magnitude of speed-up. As Dijkstra's shortest path computation stops once the target vertex is found, a longer path usually correlates with a higher number vertices explored. Similarly, sparsity of the spanner correlates with fewer vertices explored during bundling in the S-EPB algorithm, thus, an overall speed-up. Biconnected component decomposition did help on certain datasets, but its performance was more variable and dataset dependent as expected. Especially, the Panama Papers and Amazon200k datasets profit from the biconnected component decomposition as an Edge-Path bundling is computed on a much smaller sub-graph; see Table 1 for details.

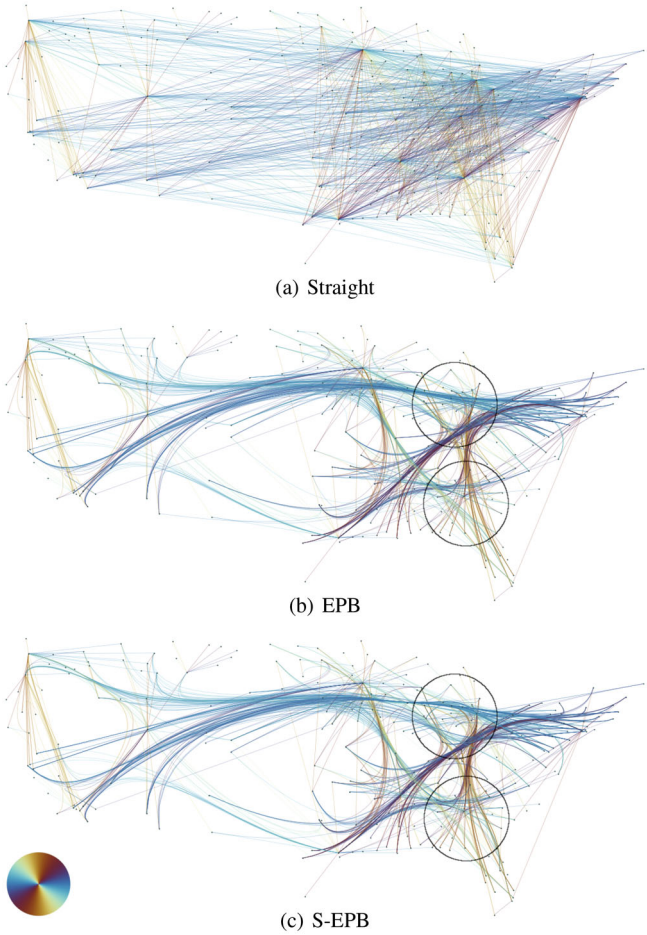
4.4. Quality metrics

Tables 4 and 5 show the results of the quality metric calculations on the three small real-world datasets. As presented in Wallinger *et al.* [WAA*22], the ambiguity metric is costly to compute and could not be computed on the larger datasets. Overall, all approaches are comparable, especially for higher values of delta. S-EPB without all pairs shortest path on the spanner tends to bundle less than the other approaches (seen by higher ink ratio and some lower ambiguity). Therefore, we can conclude that S-EPB has a similar performance to EPB on these quality metrics, but with an improvement in terms of runtime.

4.5. Comparison of image results

Figures 4–6 show the results of EPB and S-EPB on the Airlines and Migrations datasets, respectively. S-EPB is consistently faster when computing a bundling of these images. Even though EPB and S-EPB will compute different bundlings, the results look very similar as can be seen from the images. There are small differences between the two (burgundy bundles near Texas in Figure 4, thickness of bundle near the centre of the United States in Figure 5, and bundles across the Atlantic and Pacific in Figure 6), but the overall structure is similar with both bundlings free of independent edge ambiguities. Therefore, performance, in terms of speed, is greatly improved with the drawing quality remaining the same.

In Figure 7, we vary and compare the t and κ parameters to see how the stretch factor and bundling parameter influence the quality of the drawing. Both results images and quality metrics are available in this figure. The stretch factor t will determine which edges go into the graph spanner while the bundling parameter κ determines which



(a) Straight

(b) EPB

(c) S-EPB

Figure 4: *Airlines (undirected). (a) Straight line drawing. (b) The original Edge-Path bundling algorithm. (c) Spanner Edge-Path bundling. Visual quality is very similar for EPB and S-EPB with minor differences—specifically, the ocher and turquoise colour bundles around Atlanta and the bundles in the Great Lakes region (marked by circles).*

are bundled (higher values of κ favour short edges on shortest paths). When t and κ are both low, the drawing divides itself into many bundles in a similar way to Winding Roads [LBA10]. Lower values of t mean that more edges are present in the graph spanner which

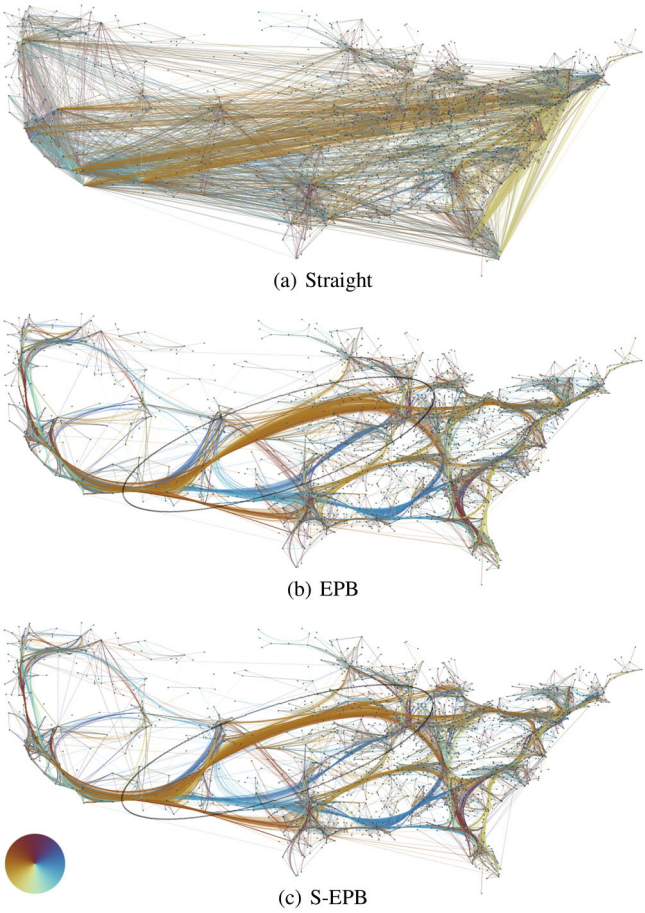


Figure 5: *Migrations (directed). (a) Straight line drawing. (b) The original Edge-Path bundling algorithm. (c) Spanner Edge-Path bundling. Visual quality is very similar for EPB and S-EPB with minor differences—specifically, the bundle at the centre of the United States (marked by an ellipse).*

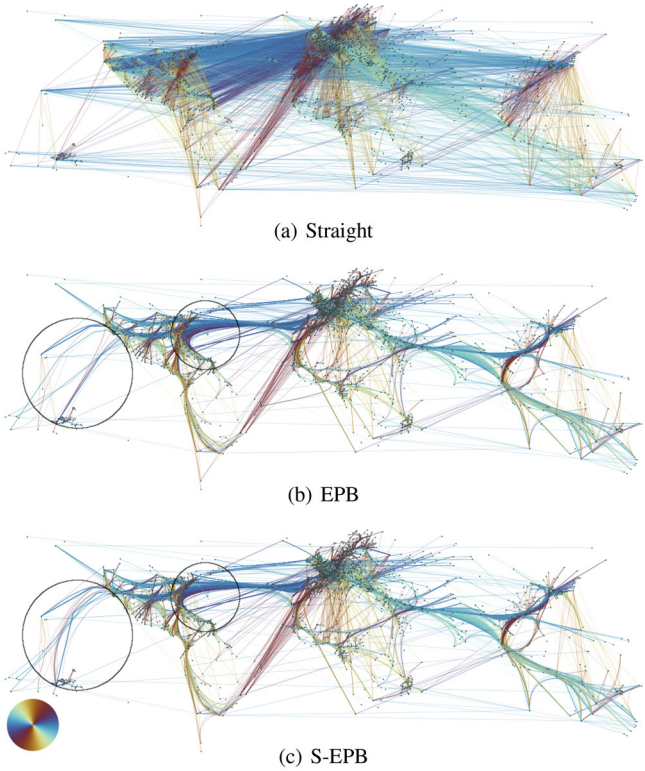


Figure 6: *Airtraffic (undirected). (a) Straight line drawing. (b) The original Edge-Path bundling algorithm. (c) Spanner Edge-Path bundling. Visual quality is very similar for EPB and S-EPB with minor differences—specifically, the bundles over the Atlantic and Pacific Oceans (marked by circles).*

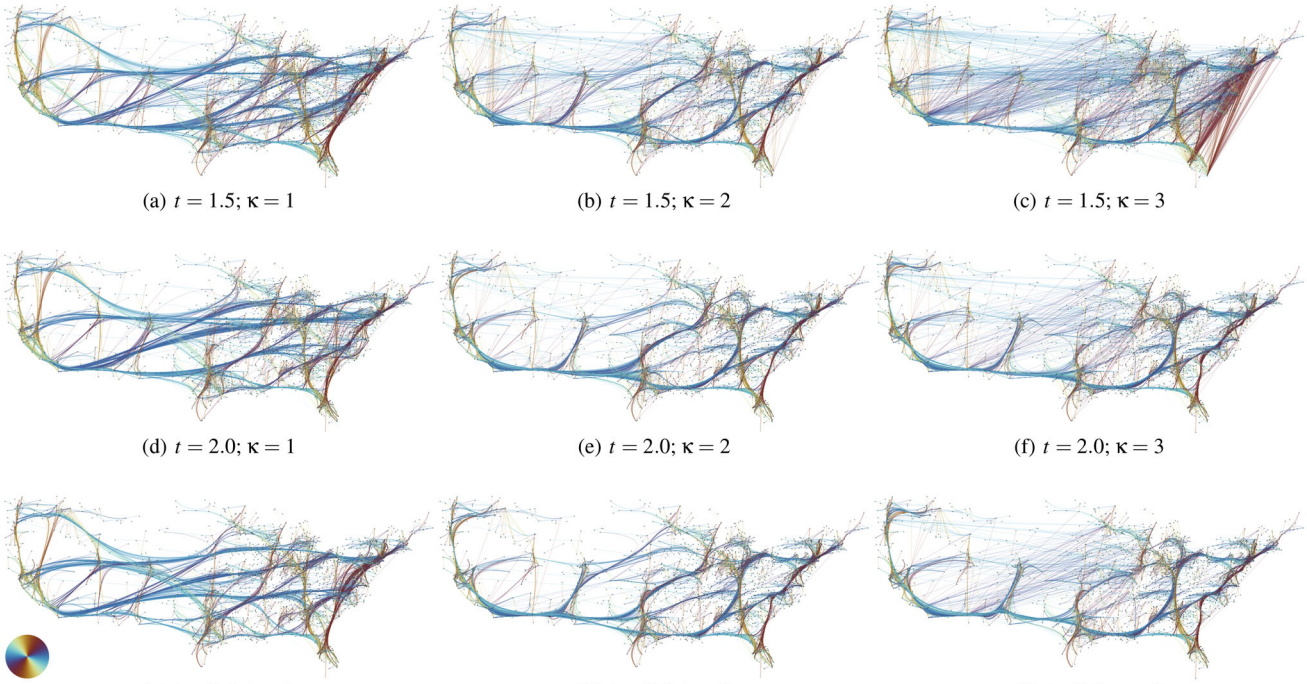
results more of them being used as parts of paths. Low versions of κ encourage bundling with low thresholds. Higher values of t and κ result in fewer bundles and more unbundled edges. The high value of t will cause a sparser graph spanner meaning there fewer edges can be used for paths, leading to fewer bundles. A high value of κ leads to more unbundled edges as the threshold is more easily exceeded. We can see these effects when we vary t (low values number of bundles increase) and κ (high values less bundling) independently through the table. As a reminder, our main experiments were run with S-EPB $t = 2, \kappa = 2$. This value of κ is the equivalent setting for the original Edge-Path bundling algorithm.

4.6. Unsuccessful optimizations

The following observations describe that some of the unsuccessful optimization ideas we discarded after the experimentation with the implementation showed that they did not have a positive effect.

One observation is that we have already computed a valid path to bundle against when it is decided an edge will not be added to the spanner. Therefore, we can reuse the shortest path computations from the spanner construction for bundling edges against paths. While this produces a valid bundling, both the runtime and quality of the bundling are reduced. The runtime degrades because we keep track of the predecessors during the shortest path computations. Furthermore, adapting this method to work with the FG-greedy algorithm means that it is necessary to additionally store shortest paths between vertices and update them, increasing memory requirements. The quality is mainly affected by the fact that edges are bundled against shortest paths in an incomplete spanner. Visually, this results in a low level of bundling. See the supplementary material on OSF for images.

Also, the above-mentioned issue of a low level of bundling arises from the fact that the bundling parameter κ is not used. We tested a variant of the above algorithm that keeps track of both $||\cdot||$ and $||\cdot||^\kappa$ during the greedy spanner construction. Vertices on the shortest path are explored in order of $||\cdot||^\kappa$ but only if $||\cdot||$ cost of the path p is also valid. While this has the desired effect of clustering bundled edges along paths with short edges, the resulting images are overbundled with bundles that are too tight. Furthermore, the overhead of storing additional information during path computations affects the runtime. See the supplementary material on OSF for images.



(a) $t = 1.5; \kappa = 1$ (b) $t = 1.5; \kappa = 2$ (c) $t = 1.5; \kappa = 3$

(d) $t = 2.0; \kappa = 1$ (e) $t = 2.0; \kappa = 2$ (f) $t = 2.0; \kappa = 3$

(g) $t = 2.5; \kappa = 1$ (h) $t = 2.5; \kappa = 2$ (i) $t = 2.5; \kappa = 3$

t \ κ	1			2			3		
	ink_J	dist	amb^1	ink_J	dist	amb^1	ink_J	dist	amb^1
1.5	0.65	1.03	0.74	0.72	1.04	0.74	0.87	1.03	0.75
2.0	0.60	1.05	0.77	0.55	1.07	0.72	0.64	1.07	0.75
2.5	0.58	1.08	0.78	0.50	1.10	0.76	0.62	1.10	0.76

(j) Quality Metrics

Figure 7: *Migrations (undirected). Different parameters are tested. (a)–(i) Values for t ranging from 1.5 to 2.5 and κ ranging from 1 to 3. With decreasing t, more edges are available in the spanner to be used as paths, splitting the drawing into more bundles. With increasing κ, fewer edges are bundled because the threshold is harder to exceed. Shorter edges are favoured for paths to be used in bundling. (j) Quality metrics for this range of parameter values. As expected, lower values of t produce higher ink ratios, low distortion and a bit less ambiguity. Higher values of κ can have higher ink ratio, similar distortion and similar ambiguity.*

As the graph spanner does not change after the construction, we tried to compute all shortest paths in one sweep by performing an all-pairs-shortest-path computation. The results of this computation was stored in a dictionary, which was used to query the paths between two end points when the remaining edges in $E \setminus E'$ were processed. This resulted in a minor runtime improvement for some experiments, but it did not generalize to all experiments. For the larger datasets (Amazon200k, Panama Papers), this memory overhead was too high causing the approach to run out of memory when constructing this additional data structure.

As we are mainly interested in bundling long edges, we tested introducing a threshold length on the edges for bundling, which could be used to instantly add shorter edges to the spanner. For example, we sorted the edges in increasing order and added the first 20% of edges to E' before proceeding with the greedy algorithm on the remaining edges. In practice, this approach added unnecessary edges to the spanner that increased the cost of the shortest path computation and resulted in slower total runtimes.

Finally, we experimented with replacing Dijkstra's shortest path algorithm with A* [HNR68] and the Euclidean distance between vertices as heuristic. While we noticed minor speed decrease for the smaller datasets, the additional cost of pre-computing and storing the pair-wise distances between vertices did increase the runtime for the larger datasets.

5. Conclusion

In this paper, we presented Spanner Edge-Path bundling. This approach uses graph spanners to accelerate the computation of edge path bundling while achieving commensurate results in terms of visual quality. The approach provides a 5–256 times speed increase

when compared to Edge-Path bundling, depending on the dataset used. Although the approach has the same worst case complexity of $O(|E|^2 \log |V|)$, as the graph spanner is sparser than the full graph G, the shortest path computations take less time. The bundling computed by S-EPB is not the same as that of EPB, but both are of the same class of algorithm that do not produce independent edge ambiguities.

We have improved the computational performance of Edge-Path bundling to bring it in line with other bundling approaches. However, the approach still cannot compete with image-based approaches, such as CUBu [vCT16], which can bundle large datasets in less than a second but suffers from independent edge ambiguities. In future work, it would be interesting to see if image-based techniques can be adapted to minimize the impact of independent edge ambiguities in bundlings of graphs or if the computational complexity, worst case or otherwise, of the approach can be reduced by other means.

As noted in the introduction, there is a strong connection between algorithms that produce Edge-Path bundlings and graph spanners. In this paper, we explore greedy spanner approaches, but other spanner construction algorithms exist which have not been explored. For example, probabilistic methods can produce spanners in linear time but without the sparseness guarantees of the greedy algorithm. Similarly, there is extensive work on increasing the efficiency of shortest path calculations. Especially, shortest path algorithms on dynamic data structures would be worth investigating.

As noted in the survey [LHT17b], drawings are bundled. Thus, bundlings will vary depending on the drawing of the graph. In future work, it would be interesting to see graph drawing algorithms that are able to optimize EPB algorithms, and possibly traditional edge bundling algorithms, to reduce visual clutter in drawings.

It is important to emphasize that not all the tasks edge bundling can support require the complete absence of independent edge ambiguities. Traditional edge bundling algorithms group edges headed in the same direction together, and as mentioned in related work, are more appropriate for the case of trail-sets. In the case of trajectories or trails, when the edges are a group of trajectories that start in one location and end up in another, Edge-Path bundling is not possible as no Edge-Path primitives exist and clustering groups of edges in the layout is sufficient. Future work should consider ways of determining when the extra constraints of Edge-Path bundling are needed to support the user tasks.

Conceptually, Edge-Path bundling removes all independent edge ambiguities, however, in cases where two bundles or edges cross at shallow angles there can still be visual ambiguities. Future research could focus on computing a different set of control points that resolves this issue.

Last, Edge-Path bundling uses the vertices of a path as a structure to infer control points of the curve representing an edge. However, depending on the context of the underlying data such a path might imply semantic meaning. Directions of future work could tackle this observation by decoupling the vertices of a path and the implied control points.

Acknowledgements

The authors have nothing to report.

References

[AAM06] Archambault D., Auber D., Munzner T.: Smashing peacocks further: Drawing quasi-trees from biconnected components. *IEEE Transactions on Visualization and Computer Graphics 12*, 05 (2006), 813–820. doi:10.1109/TVCG.2006.177

[ABS*20] Ahmed A. R., Bodwin G., Sahneh F. D., Hamm K., Jebelli M. J. L., Kobourov S. G., Spence R.: Graph spanners: A tutorial review. *Computer Science Review 37* (2020), 100253. doi:10.1016/j.cosrev.2020.100253

[ABtBB15] Alewijnse S. P. A., Bouts Q. W., ten Brink A. P., Buchin K.: Computing the greedy spanner in linear space. *Algorithmica 73*, 3 (2015), 589–606. doi:10.1007/s00453-015-0001-2

[ADD*93] Althöfer I., Das G., Dobkin D. P., Joseph D., Soares J.: On sparse spanners of weighted graphs. *Discret. Comput. Geom. 9* (1993), 81–100. doi:10.1007/BF02189308

[ADF*19] Alstrup S., Dahlgaard S., Filtser A., Stöckel M., Wulff-Nilsen C.: Constructing light spanners deterministically in near-linear time. In *Proceedings of the 27th Annual European Symposium on Algorithms, LIPIcs* (2019), vol. *144*, pp. 4:1–4:15. doi:10.4230/LIPIcs.ESA.2019.4

[AHJ*19] Ahmed A. R., Hamm K., Jebelli M. J. L., Kobourov S. G., Sahneh F. D., Spence R.: Approximation algorithms and an integer program for multi-level graph spanners. In *Analysis of Experimental Algorithms - Special Event, SEA^2 Revised Selected Papers*, *LNCS* (vol. *11544*). Springer, Cham (2019), pp. 541–562. doi:10.1007/978-3-030-34029-2_35

[Awe85] Awerbuch B.: Complexity of network synchronization. *Journal of the ACM 32*, 4 (1985), 804–823. doi:10.1145/4221.4227

[BBM*13] Berman P., Bhattacharyya A., Makarychev K., Raskhodnikova S., Yaroslavtsev G.: Approximation algorithms for spanner problems and Directed Steiner Forest. *Information and Computation 222* (2013), 93–107. doi:10.1016/j.ic.2012.10.007

[BCF*10] Bose P., Carmi P., Farshi M., Maheshwari A., Smid M. H. M.: Computing the greedy spanner in near-quadratic time. *Algorithmica 58*, 3 (2010), 711–729. doi:10.1007/s00453-009-9293-4

[BGJ*12] Bhattacharyya A., Grigorescu E., Jung K., Raskhodnikova S., Woodruff D. P.: Transitive-closure spanners. *SIAM Journal on Computing 41*, 6 (2012), 1380–1425. doi:10.1137/110826655

[BRH*17] Bach B., Riche N. H., Hurter C., Marriott K., Dwyer T.: Towards unambiguous edge bundling: Investigating confluent drawings for network visualization. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (2017), 541–550. doi:10.1109/TVCG.2016.2598958

[BS07] Baswana S., Sen S.: A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms 30*, 4 (2007), 532–563. doi:10.1002/rsa.20130

[Cai94] Cai L.: NP-completeness of minimum spanner problems. *Discrete Applied Mathematics 48*, 2 (1994), 187–194. doi:10.1016/0166-218X(94)90073-6

[CGJ*13] Chimani M., Gutwenger C., Jünger M., Klau G. W., Klein K., Mutzel P.: The open graph drawing framework (OGDF). In *Handbook on Graph Drawing and Visualization*. R. Tamassia (Ed.). Chapman and Hall/CRC, Boca Raton, FL (2013), pp . 543–569.

[Cow01] Cowen L.: Compact routing with minimum stretch. *Journal of Algorithms 38*, 1 (2001), 170–183. doi:10.1006/jagm.2000.1134

[CS21] Chimani M., Stutzenstein F.: Spanner approximations in practice. In *30th Annual European Symposium on Algorithms (ESA 2022)*. Leibniz International Proceedings in Informatics (LIPIcs) (2022). S. Chechik, G. Navarro, E. Rotenberg and G. Herman (Eds.), vol. *244*: pp. 37:1–37:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik: Dagstuhl, Germany. https://drops.dagstuhl.de/opus/volltexte/2022/16975

[CW04] Cowen L., Wagner C. G.: Compact roundtrip routing in directed networks. *Journal of Algorithms 50*, 1 (2004), 79–95. doi:10.1016/j.jalgor.2003.08.001

[CZQ*08] Cui W., Zhou H., Qu H., Wong P. C., Li X.: Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1277–1284. doi:10.1109/TVCG.2008.135

[DEGM05] Dickerson M., Eppstein D., Goodrich M. T., Meng J. Y.: Confluent drawings: Visualizing non-planar diagrams in a planar way. *Journal of Graph Algorithms and Applications 9*, 1 (2005), 31–52. doi:10.7155/jgaa.00099

[DK11] Dinitz M., Krauthgamer R.: Directed spanners via flow-based linear programs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)* (2011), pp. 323–332. doi:10.1145/1993636.1993680

[DN97] Das G., Narasimhan G.: A fast algorithm for constructing sparse euclidean spanners. *International Journal of Computational Geometry & Applications 7*, 4 (1997), 297–315. doi:10.1142/S0218195997000193

[DZ16] Dinitz M., Zhang Z.: Approximating low-stretch spanners. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA* (2016), pp. 821–840. doi:10.1137/1.9781611974331.ch59

[EGM06] Eppstein D., Goodrich M. T., Meng J. Y.: Delta-confluent drawings. In *Graph Drawing (GD'05)*, *LNCS* (2006), Springer, vol. *3843*, pp. 165–176. doi:10.1007/11618058_16

[EHL*16] Eppstein D., Holten D., Löffler M., Nöllenburg M., Speckmann B., Verbeek K.: Strict confluent drawing. *Journal of Computational Geometry 7*, 1 (2016), 22–46. doi:10.20382/jocg.v7i1a2

[EHP*11] Ersoy O., Hurter C., Paulovich F., Cantareiro G., Telea A.: Skeleton-based edge bundling for graph visualization. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2364–2373. doi:10.1109/TVCG.2011.233

[EN19] Elkin M., Neiman O.: Efficient algorithms for constructing very sparse spanners and emulators. *ACM Transactions on Algorithms 15*, 1 (2019), 4:1–4:29. doi:10.1145/3274651

[EP01] Elkin M., Peleg D.: Approximating k-spanner problems for k>2. In *Proceedings of the 8th International IPCO Conference on Integer Programming and Combinatorial Optimization* (2001), *LNCS*, Springer, vol. *2081*, pp. 90–104. doi:10.1007/3-540-45535-3_8

[epb] The Edgbundle R Package. https://github.com/schochastics/edgebundle. Accessed: October 30, 2021.

[ES16] Elkin M., Solomon S.: Fast constructions of lightweight spanners for general graphs. *ACM Transactions on Algorithms 12*, 3 (2016), 29:1–29:21. doi:10.1145/2836167

[FG07] Farshi M., Gudmundsson J.: Experimental study of geometric t-spanners: A running time comparison. In *Proceedings of the 6th International Workshop on Experimental Algorithms, WEA*, *LNCS* (2007), Springer, vol. *4525*, pp. 270–284. doi:10.1007/978-3-540-72845-0_21

[FG09] Farshi M., Gudmundsson J.: Experimental study of geometric *t*-spanners. *ACM Journal of Experimental Algorithmics 14* (2009). doi:10.1145/1498698.1564499

[FGKN19] Förster H., Ganian R., Klute F., Nöllenburg M.: On strict (outer-) confluent graphs. In *Graph Drawing and Network Visualization (GD'19), LNCS* (vol. *11904*). Springer, Cham (2019), pp. 147–161. doi:10.1007/978-3-030-35802-0_12

[FS20] Filtser A., Solomon S.: The greedy spanner is existentially optimal. *SIAM Journal on Computing 49*, 2 (2020), 429–447. doi:10.1137/18M1210678

[GHNS11] Gansner E. R., Hu Y., North S., Scheidegger C.: Multilevel agglomerative edge bundling for visualizing large graphs. In *Proceedings of the Pacific Visualization Symposium* (2011), IEEE, pp. 187–194. doi:10.1109/PACIFICVIS.2011.5742389

[HET12] Hurter C., Ersoy O., Telea A.: Graph bundling by kernel density estimation. *Computer Graphics Forum 31*, 3 (2012), 865–874. doi:10.1111/j.1467-8659.2012.03079.x

[HJ04] Hachul S., Jünger M.: Drawing large graphs with a potential-field-based multilevel algorithm. In *Proceedings of the International Symposium on Graph Drawing* (2004), pp . 285–295.

[HMDMAGB22] Hassan-Montero Y., De-Moya-Anegón F., Guerrero-Bote V. P.: SCImago graphica: A new tool for exploring and visually communicating data. *Profesional de la información 31*, 5 (Sep. 2022). doi:10.3145/epi.2022.sep.02

[HNR68] Hart P. E., Nilsson N. J., Raphael B.: A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science & Cybernetics 4*, 2 (1968), 100–107. doi:10.1109/TSSC.1968.300136

[Hol06] Holten D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 741–748. doi:10.1109/TVCG.2006.147

[HPSŠ07] Hui P., Pelsmajer M. J., Schaefer M., Štefankovič D.: Train tracks and confluent drawings. *Algorithmica 47* (2007), 465–479. doi:10.1007/s00453-006-0165-x

[HvW09] Holten D., van Wijk J. J.: Force-directed edge bundling for graph visualization. *Computer Graphics Forum 28*, 3 (2009), 983–990. doi:10.1111/j.1467-8659.2009.01450.x

[KP94] Kortsarz G., Peleg D.: Generating sparse 2-spanners. *Journal of Algorithms 17*, 2 (1994), 222–236. doi:10.1006/jagm.1994.1032

[LBA10] Lambert A., Bourqui R., Auber D.: Winding Roads: Routing edges into bundles. *Computer Graphics Forum 29*, 3 (2010), 853–862. doi:10.1111/j.1467-8659.2009.01700.x

[LDB11] Lambert A., Dubois J., Bourqui R.: Pathway preserving representation of metabolic networks. *Computer Graphics Forum 30*, 3 (2011), 1021–1030. doi:10.1111/j.1467-8659.2011.01951.x

[LHT17a] Lhuillier A., Hurter C., Telea A.: FFTEB: Edge bundling of huge graphs by the fast fourier transform. In *Proceedings of the IEEE Pacific Visualization Symposium* (2017), pp. 190–199. doi:10.1109/PACIFICVIS.2017.8031594

[LHT17b] Lhuillier A., Hurter C., Telea A.: State of the art in edge and trail bundling techniques. *Computer Graphics Forum 36*, 3 (2017), 619–645. doi:10.1111/cgf.13213

[LLC*20] Lyu Y., Liu X., Chen H., Mangal A., Liu K., Chen C., Lim B. Y.: OD morphing: Balancing simplicity with faithfulness for OD bundling. *IEEE Transactions on Visualization and Computer Graphics 26*, 1 (2020), 811–821. doi:10.1109/TVCG.2019.2934657

[LLCM12] Luo S., Liu C., Chen B., Ma K.: Ambiguity-free edge-bundling for interactive graph visualization. *IEEE Transactions on Visualization and Computer Graphics 18*, 5 (2012), 810–821. doi:10.1109/TVCG.2011.104

[MPVX15] Miller G. L., Peng R., Vladu A., Xu S. C.: Improved parallel algorithms for spanners and hopsets. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA* (2015), ACM, pp. 192–201. doi:10.1145/2755573.2755574

[NHE11] Nguyen Q. H., Hong S., Eades P.: TGI-EB: A new framework for edge bundling integrating topology, geometry and importance. In *Graph Drawing (GD'11), LNCS* (vol. *7034*). Springer, Berlin (2011), pp. 123–135. doi:10.1007/978-3-642-25878-7_13

[NOB15] Nocaj A., Ortmann M., Brandes U.: Untangling the hairballs of multi-centered, small-world online social media networks. *Journal of Graph Algorithms and Applications 19*, 2 (2015), 595–618. doi:10.7155/jgaa.00370

[PNBH16] Pupyrev S., Nachmanson L., Bereg S., Holroyd A. E.: Edge routing with ordered bundles. *Computational Geometry 52* (2016), 18–33. doi:10.1016/j.comgeo.2015.10.005

[PS89] Peleg D., Schäffer A. A.: Graph spanners. *Journal of Graph Theory 13*, 1 (1989), 99–116. doi:10.1002/jgt.3190130114

[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. doi:10.1016/j.scico.2012.05.002

[RZ11] Roditty L., Zwick U.: On dynamic shortest paths problems. *Algorithmica 61*, 2 (2011), 389–401. doi:10.1007/s00453-010-9401-5

[SHH11] Selassie D., Heller B., Heer J.: Divided edge bundling for directional network data. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2354–2363. doi:10.1109/TVCG.2011.190

[SS10] Shpungin H., Segal M.: Near-optimal multicriteria spanner constructions in wireless ad hoc networks. *IEEE/ACM Transactions on Networking 18*, 6 (2010), 1963–1976. doi:10.1109/TNET.2010.2053381

[SVZ07] Schindelhauer C., Volbert K., Ziegler M.: Geometric spanners with applications in wireless networks. *Computational Geometry 36*, 3 (2007), 197–214. doi:10.1016/j.comgeo.2006.02.001

[SZ04] Sigurd M., Zachariasen M.: Construction of minimum-weight spanners. In *Proceedings of the 12th Annual European Symposium on Algorithms - ESA 2004*, *LNCS* (2004), vol. *3221*, pp. 797–808. doi:10.1007/978-3-540-30140-0_70

[TE10] Telea A., Ersoy O.: Image-based edge bundles: Simplified visualization of large graphs. *Computer Graphics Forum 29*, 3 (2010), 843–852. doi:10.1111/j.1467-8659.2009.01680.x

[TNI*17] Toeda N., Nakazawa R., Itoh T., Saito T., Archambault D.: Convergent drawing for mutually connected directed graphs. *Journal of Visual Languages & Computing 43* (2017), 83–90. doi:10.1016/j.jvlc.2017.09.004

[TP15] Thöny M., Pajarola R.: Vector map constrained path bundling in 3D environments. In *Proceedings of the 6th ACM SIGSPATIAL International Workshop on GeoStreaming, IWGS 2015, November 3–6, 2015* (Bellevue, WA, USA, Nov. 2015), F. B. Kashani, C. Zhang and A. M. Hendawi (Eds.), ACM, pp. 33–42. doi:10.1145/2833165.2833168

[TZ05] Thorup M., Zwick U.: Approximate distance oracles. *Journal of the ACM 52*, 1 (2005), 1–24. doi:10.1145/1044731.1044732

[vCT16] van der Zwan M., Codreanu V., Telea A.: CUBu: universal real-time bundling for large graphs. *IEEE Transactions on Visualization and Computer Graphics 22*, 12 (2016), 2550–2563. doi:10.1109/TVCG.2016.2515611

[vHW08] van Ham F., Wattenberg M.: Centrality based visualization of small world graphs. *Computer Graphics Forum* (2008). doi:10.1111/j.1467-8659.2008.01232.x

[WAA*22] Wallinger M., Archambault D., Auber D., Nöllenburg M., Peltonen J.: Edge-path bundling: A less ambiguous edge bundling approach. *IEEE Transactions on Visualization and Computer Graphics* (2022), 313–323. doi:10.1109/TVCG.2021.3114795

[WYY15] Wu J., Yu L., Yu H.: Texture-based edge bundling: A web-based approach for interactively visualizing large graphs. In *Proceedings of the IEEE Big Data* (2015), pp. 2501–2508. doi:10.1109/BigData.2015.7364046

[ZPG21] Zheng J. X., Pawar S., Goodman D. F. M.: Further towards unambiguous edge bundling: Investigating power-confluent drawings for network visualization. *IEEE Transactions on Visualization and Computer Graphics 27*, 3 (2021), 2244–2249. doi:10.1109/TVCG.2019.2944619

[ZSJT19] Zeng W., Shen Q., Jiang Y., Telea A. C.: Route-aware edge bundling for visualizing origin-destination trails in urban traffic. *Computer Graphics Forum 38*, 3 (2019), 581–593. doi:10.1111/cgf.13712