Clustered edge routing

Citation for published version (APA):

Bouts, Q. W., & Speckmann, B. (2015). Clustered edge routing. In S. Liu, G. Scheuermann, & S. Takahashi (Eds.), *2015 IEEE Pacific Visualization Symposium (PacificVis2015, Hangzhou, China, April 14-17, 2015)* (pp. 55-62). Institute of Electrical and Electronics Engineers. <https://doi.org/10.1109/PACIFICVIS.2015.7156356>

DOI:

[10.1109/PACIFICVIS.2015.7156356](https://doi.org/10.1109/PACIFICVIS.2015.7156356)

Document status and date:

Published: 01/01/2015

Document Version:

Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Clustered Edge Routing

Quirijn W. Bouts* Bettina Speckmann*
Eindhoven University of Technology

Figure 1: Clustered edge routing based on a well-separated pair decomposition and a greedy sparsification of the visibility graph on a synthetic data set. One edge cluster is highlighted in red.

ABSTRACT

The classic method to depict graphs is a node-link diagram where vertices (nodes) are associated with each object and edges (links) connect related objects. However, node-link diagrams quickly appear cluttered and unclear, even for moderately sized graphs. If the positions of the nodes are fixed then suitable link routing is the only option to reduce clutter. We present a novel link clustering and routing algorithm which respects (and if desired refines) user-defined clusters on links. If no clusters are defined a priori we cluster based on geometric criteria, that is, based on a *well-separated pair decomposition* (WSPD). We route link clusters individually on a *sparse visibility spanner*. To completely avoid ambiguity we draw each individual link and ensure that clustered links follow the same path in the routing graph. We prove that the clusters induced by the WSPD consist of compatible links according to common similarity measures as formalized by Holten and van Wijk [17]. The greedy sparsification of the visibility graph allows us to easily route around obstacles. Our experimental results are visually appealing and convey a sense of abstraction and order.

Index Terms: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1 INTRODUCTION

Graphs are an important tool to express relational data. Common examples include airline and migration routes, biological networks, the interaction between classes in software systems, and human interactions in social networks. The classic method to depict graphs is a node-link diagram where vertices (nodes) are associated with each object and edges (links) connect related objects. Node-link diagrams present the information contained in a graph in the most direct way. However, node-link diagrams quickly appear cluttered and unclear, even for moderately sized graphs. If the positions of the nodes are fixed – because they represent geo-referenced data or are laid out according to functional requirements – then suitable link routing is the only option to reduce clutter. Hence recent years have seen a significant number of papers that address this issue.

Results and organization. We present a novel link clustering and routing algorithm which respects (and if desired refines) user-defined clusters on the links. Our input is a node-link diagram with fixed node positions and optionally a user-defined clustering on the links and/or a set of disjoint polygonal obstacles. Our clustering method is based on a *well-separated pair decomposition* (WSPD) and we route link clusters individually on a *sparse visibility spanner*. To completely avoid ambiguity we draw each individual link and guarantee that clustered links follow the same path in the routing graph (the routing does not “mix” links from different clusters). (To avoid confusion we refer to the edges of the routing graph as ‘edges’ and use the term ‘links’ to describe the input from the node-link diagram.) Our algorithm also ensures that clusters are not drawn close to nodes and do not cross obstacles. In contrast to previous work we separate the bundling and the routing step which allows either to be improved or changed independently of the other.

The well-separated pair decomposition (WSPD) was introduced by Callahan et al. [7]. Two point sets A and B are well-separated if they can be enclosed in two circles of equal diameter which are “far apart” relative to their diameter, where “far” is quantified by a separation constant. Holten and van Wijk [17] formalized four *edge compatibility measures* which indicate how similar links are. In Section 2 we argue that the clusters induced by a WSPD consist of compatible links according to the measures of Holten and van Wijk. The separation constant of the WSPD gives users a single parameter to vary to find a balance between few clusters of less compatible links and many clusters of very compatible links.

In Section 3 we show how to route the link clusters defined by the WSPD along a routing graph, namely the greedy sparsification of the visibility graph. The visibility graph can easily handle obstacles. By using a spanner as sparsification we maintain a bound on the detour of the shortest path between any two nodes in the graph. On the complete graph a greedy sparsification has several desirable properties, such as a provable angle constraint as well as low total weight. Our experiments in Section 5 show that these properties still hold in practice for the sparsification of the visibility graph.

In Section 4 we describe our complete drawing algorithm, including pre-processing, link ordering, and crossing minimization. Our method is designed to work best with small to medium sized graphs, but it can also deal with graphs of several thousand vertices. The experimental results in Section 5 are visually appealing and convey a sense of abstraction and order.

*e-mail: [q.w.bouts | b.speckmann]@tue.nl

Related work. Our work is closely related to the approach proposed by Pupyrev *et al.* [23]¹ who route links around obstacles using a routing graph based on a Yao-spanner of the visibility graph. The Yao-spanner is a cone-based spanner and as such exhibits the typical problems of its kind: edges can get arbitrarily close and angles can be arbitrarily small. Pupyrev *et al.* hence go through an extensive and computationally expensive iterative optimization step to improve this spanner (no provable quality guarantees can be given for the result). Dwyer and Nachmanson [12] also use visibility graphs, albeit approximate ones, to route edges. They describe two approaches. The first uses a spatial decomposition and requires node movement. It can hence not be used to draw graphs with fixed node positions. The second approach also uses the Yao-spanner and hence exhibits the same problems described above. Both papers route links by using shortest paths on the routing graph. Link clustering is hence simply induced by the routing graph and does not necessarily satisfy any similarity measures. Furthermore, neither of these methods supports user-defined link clusters. We show experimentally in Section 5 that our Greedy sparsification clearly outperforms the Yao sparsification.

Dwyer *et al.* [11] investigate how to integrate link routing techniques into a force-directed layout. Their method requires a feasible initial routing and moves vertices, which makes it not applicable if object locations are fixed.

Various techniques have been proposed to reduce link clutter by *bundling* links which are “close”, be it geometrically in a straight line drawing of the graph or conceptually in a hierarchical structure of the data. Gansner *et al.* [15] were among the first to use bundling. They consider graphs with a circular layout and route links either on the inside or the outside of the circle. Holten and van Wijk [17] describe a force directed approach and use the aforementioned compatibility measures to determine the strengths of forces. Cui *et al.* [8] propose a geometry-based approach which uses a control mesh to bundle links. Lambert *et al.* [20] use a combination of the Voronoi diagram and a quadtree as a multi-resolution grid for routing links. Colors and opacity are used in several techniques to enhance a bundling [8, 16]. Image-based techniques which use the medial axis or skeletons have also been deployed to cluster and bundle links [13, 26]. Hurter *et al.* [18] propose to use image sharpening techniques to merge local height maxima on a kernel density map to find a bundling. All these techniques use a straight line drawing of the graph as input.

Bundling methods generally draw bundled links on top of each other and hence it can become difficult, if not impossible, to decide unambiguously if two nodes are connected. Luo *et al.* [21] propose a bundling method which is ambiguity-free. This comes at a price: bundled links need to share a common node.

Some specifically fast techniques have been developed for large graphs. The method of Dwyer and Nachmanson mentioned earlier is one of these. A different technique for large graphs was proposed by Gansner *et al.* [14], they represent links as points in 4-dimensional space allowing for a simple distance metric in that space to define link similarity. Aside from techniques for general graphs, there are also algorithms specifically targeted at hierarchical data [16] and for layered graphs [24].

Confluent drawings, proposed by Dickerson *et al.* [10], draw certain non-planar graphs in a planar way by merging links. Similar to our approach the graphs are drawn in such a way that they have a one-to-one mapping with the original graph. That is, every link in the confluent drawing is also in the original graph. More recently Quercini *et al.* [25] used confluent drawings in combination with rectangular dualization to draw planar versions of general graphs.

¹ see “S. Bereg, A. E. Holroyd, L. Nachmanson, and S. Pupyrev. Edge Routing with Ordered Bundles.” arXiv:1209.4227 for a more extensive version.

2 CLUSTERING LINKS VIA A WELL-SEPARATED PAIR DECOMPOSITION

Our algorithm uses a well-separated pair decomposition (WSPD) to cluster links and to refine user-defined clusters. Here we first give the definition of a WSPD and then prove that the links clustered by a WSPD are compatible according to the compatibility measures of Holten and van Wijk [17].

Two point sets A and B are well-separated if they can be enclosed in two circles of equal diameter which are “far apart” relative to their diameter. More precisely, point sets A and B , with bounding boxes $R(A)$ and $R(B)$, are said to be s -well-separated for some separation constant $s > 0$ if $R(A)$ and $R(B)$ can be enclosed in two disjoint equal diameter circles C_A and C_B and the distance between C_A and C_B is at least s times the diameter of C_A . The WSPD of a set of points P with separation constant s is a sequence of m pairs $\{A_i, B_i\}$ of nonempty subsets of P such that

1. for each $1 \leq i \leq m$, A_i and B_i are well-separated with respect to s .
2. for any two distinct points p and q there is exactly one pair (A_i, B_i) such that p is in one set and q in the other.

The number of well-separated pairs m is also called the size of the WSPD. If the separation constant s is indeed constant we can compute a WSPD of size $O(n)$ on a point set in the plane in $O(n \log n)$ time, see [22] Chapter 9 for details. Every well-separated pair (A_i, B_i) induces a link cluster: each link with one endpoint in A_i and the other endpoint in B_i is part of the cluster.

Link compatibility measures. Holten and van Wijk introduced four measures which concern the angle, scale, and position of a pair of links, as well as the visibility between them. Consider the s -well-separated pair $\{A, B\}$. We examine the compatibility measures on any two links $e = (p_0, p_1)$ and $f = (q_0, q_1)$ with $p_0, q_0 \in A$ and $p_1, q_1 \in B$. We assume that the link e is at most as long as the link f . We use D to denote the diameter of the circles C_A and C_B .

Angle compatibility. Links in the same cluster should have a similar angle. We define the angle α between two non-parallel links as the smallest angle between the lines induced by the links. The angle of parallel links is 0.

Lemma 1. *The angle α between e and f is bounded by $\alpha \leq 2 \cdot \tan^{-1}(\frac{1}{s})$ for $s \geq 1$.*

Proof. The figure on the right shows a worst case configuration of e and f with respect to α . We have $|(p_0, t)| \leq 0.5D$ and $|(t, r)| \geq 0.5sD$ as rough bounds. We can now bound α by $2 \cdot \tan^{-1}(\frac{1}{s})$. $\square$

Scale compatibility. Links in the same cluster should have similar length.

Lemma 2. *The difference in length of e and f is at most $2 \cdot D$. The length ratio of f to e is bounded by $\frac{|f|}{|e|} \leq \frac{s+2}{s}$.*

Proof. The minimal length of e is by definition $s \cdot D$. Since the bounding boxes fit into circles of diameter D we have $|f| \leq |e| + 2D$ as a maximal length. This gives $\frac{|f|}{|e|} \leq \frac{s+2}{s}$. $\square$

Position compatibility. Links which are close to each other should be more likely to end up in the same cluster. Holten and van Wijk measure “close to each other” by considering the distance between the midpoints p_m and q_m of links e and f in relation to the average link length of e and f .

Lemma 3. The difference in position of links e and f with mid-points p_m and q_m is $|(p_m, q_m)| \leq D$. The ratio of the difference in position to the average length is bounded by $\frac{|(p_m, q_m)|}{(|e|+|f|)/2} \leq \frac{1}{s}$.

Proof. The minimal length of both e and f is by definition $s \cdot D$. Since the endpoints of e and f lie in the same circles of diameter D , $|(p_m, q_m)| \leq D$. We now have $\frac{|(p_m, q_m)|}{(|e|+|f|)/2} \leq \frac{D}{sD} = \frac{1}{s}$. $\square$

Visibility compatibility. Let q'_m be the point on the line induced by e that when projected onto f coincides with its midpoint q_m . The visibility compatibility of e with f is defined by the normalized distance between the midpoint of e (p_m) and q'_m . To normalize this distance we divide by the length of the segment $q'_0 q'_1$ which when projected onto the line induced by f coincides with f .

Lemma 4. Let q'_0, q'_1 and q'_m be the points on the line induced by e which, when projected onto f , coincide with q_0, q_1 , and q_m . The visibility compatibility is bounded by $\frac{|(p_m, q'_m)|}{|(q'_0, q'_1)|} \leq \frac{1}{s}$ for $s > 1$.

Proof. Let α be the angle between the lines induced by e and f which using Lemma 1 we can bound as $\alpha < 2 \tan^{-1}(\frac{1}{s})$. Let $\bar{p}_m$ be the projection of p_m onto the line induced by f . We have $|\bar{p}_m, q_m| \leq |(p_m, q_m)| \leq D$ by the triangle inequality. This implies $|(p_m, q'_m)| \leq \frac{D}{\cos(\alpha)}$. From the definition of s -well-separated we have $|(q_0, q_1)| \geq sD$, which implies that $|q'_0, q'_1| \geq \frac{sD}{\cos(\alpha)}$. We now have $\frac{|(p_m, q'_m)|}{|(q'_0, q'_1)|} \leq \frac{D}{\cos(\alpha)} \cdot \frac{\cos(\alpha)}{sD} = \frac{1}{s}$. $\square$

Increasing the separation constant s of the WSPD improves all four compatibility measures. Users can hence vary the single parameter s to find a balance between few clusters of less compatible links and many clusters of very compatible links. If the user has specified clusters we test if they are also spatially clustered and can hence be drawn nicely. If this is not the case, that is, if the endpoints of the clustered links are not well-separated, we can refine the user-specified cluster into compatible clusters using the WSPD. We so ensure that links marked as related by domain experts will never be “mixed” with unrelated links.

3 LINK ROUTING VIA SPARSE VISIBILITY SPANNERS

We now describe our routing graph, the greedy sparsification of the visibility graph. There are several desirable properties for a routing graph. It should be sparse to promote clustering and not have too many small angles to avoid clutter. It should contain many short edges, so links in the same cluster can merge quickly. The shortest path between two nodes should be relatively short, not much longer than their direct connection. Since we draw links individually there also needs to be sufficient clearance around the edges of the routing graph. And finally the routing graph should easily be able to accommodate obstacles (see Fig. 2).

The visibility graph. Let S be a set of disjoint polygonal obstacles in the plane with n obstacle vertices in total. The visibility graph of S contains an edge for every pair of vertices which are visible to each other, where vertex p is visible to a vertex q if and only if the line segment (p, q) does not intersect any of the polygonal obstacles in S . The visibility graph contains the shortest path around the obstacles between any pair of vertices. In a preprocessing step we add a small simple polygon around each node in the input to ensure

Figure 2: Our routing graph constructed on the TLR4 dataset [4]. The edge types are indicated: (black) visibility edge, (blue) obstacle edge, (green) internal edge. The sparsification has been computed using dilation $t = 1.8$. No straightening has been applied to this graph.

that the routes do not overlap nodes. Since the visibility graph is defined on disjoint polygons we merge *vertex obstacle polygons* if they overlap (see Section 4.1). The visibility graph can be calculated in $O(n^2 \log n)$ time (see [9] Chapter 15).

A greedy sparsification of the visibility graph. Let $d_G(p, q)$ denote the shortest path between two vertices p and q in a graph G . A *geometric t -spanner* ($t > 1$) of a graph $G = (V, E)$ is a graph $G' = (V, E' \subseteq E)$ such that for any two vertices $p, q \in V$, $d_{G'}(p, q) \leq t \cdot d_G(p, q)$. The so-called *greedy spanner* is constructed by considering all edges $e = (p, q) \in E$ in non-decreasing order and adding them to E' if and only if $d_{G'}(p, q) > t \cdot d_G(p, q)$. The greedy spanner can be computed in $O(n^2 \log n)$ time [5]. In our implementation we use the $O(n^2 \log^2 n)$ algorithm of Bouts *et al.* [6] which is faster in practice.

By using a spanner as a sparsification method we maintain a bound on the detour of the shortest path between any two vertices. On the complete graph a greedy sparsification has several desirable properties, such as a provable angle constraint as well as low total weight. Our experiments in Section 5 show that these properties still hold in practice for the sparsification of the visibility graph. We also show experimentally that our Greedy sparsification clearly outperforms the Yao sparsification used in previous work.

4 COMPUTING THE VISUALIZATION

This section describes our complete drawing algorithm, including pre-processing, link ordering, and crossing minimization. Our algorithm has two separate phases. In the first phase the link clusters are defined, either by user input, the well-separated pair decomposition, or a combination of both. In the second phase the link clusters are smoothly drawn as *ribbons* of links along the edges of the routing graph. The exact sequence of steps and pointers to their description in the paper are given in the algorithm below.

Algorithm Clustered Edge Routing

1. Compute or refine clusters via a WSPD (see Section 2)
2. Pre-process obstacles (see Section 4.1)
3. Compute visibility graph (see Section 3)
4. Greedy sparsification (see Section 3)
5. Route, draw, and order links (see Sections 4.2–4.4)

Figure 3: (a) Obstacles are too close. (b) Voronoi diagram within convex hull. (c) Snap Voronoi to convex hull vertices and add internal edges within each cell.

Our input consists of a node-link diagram with fixed node positions and optionally a list of link clusters. The user can also specify additional obstacle polygons if desired. As mentioned before we add a small polygon around each node. The size of this polygon is influenced by the user defined *clearance* parameter. The obstacles are pre-processed and merged in such a way that the resulting obstacles are all disjoint. Suitable *internal* edges are added to connect the nodes to their obstacle vertices. Next the visibility graph is calculated on the merged obstacles and sparsified using the greedy spanner algorithm. When routing a link only the internal edges of the obstacles in which its endpoints lie are used, i.e., a link may only enter an obstacle if its endpoints are inside that obstacle. The links in a cluster are drawn parallel to each other along the routing graph edges to form a ribbon of links. The space used for these ribbons is controlled by the clearance value. Finally, we use a heuristic to minimize crossings between ribbons and the links within a ribbon.

4.1 Preprocessing obstacles

We increase the size of input obstacle polygons based on the clearance so we can route ribbons over the obstacle edges without intersecting the original obstacles. We then merge all obstacles which are too close to each other (see Fig. 3(a)). What is considered “too close” can be specified by the user relative to the clearance. In particular, we enclose such obstacles within their joint convex hull which generally has fewer vertices than the input and allows flows to easily bend around (see Fig. 4). Merged obstacles contain multiple nodes. To connect these nodes to vertices of their joint convex hull, we calculate their Voronoi diagram (see Fig. 3(b)). The Voronoi edges give us maximal clearance inside the obstacles. Since the Voronoi edges may induce vertices on the obstacle boundary which are close to each other or to the obstacle vertices. We use a snapping heuristic to move the Voronoi vertices towards each other or towards the hull vertices (see Fig. 3(c)). We do not snap vertices if this puts multiple nodes into a single snapped Voronoi cell. Finally, we connect the nodes with the vertices of its (snapped) Voronoi cell, using a simple angle constraint to ensure that these edges are not too close to each other.

4.2 Routing links

The routing algorithm needs to respect the clusters. Specifically, links from the same cluster should share a common sub-path in the routing graph. For each link cluster we choose two routing

Figure 4: Non-merged vs. merged obstacles.

graph vertices as *merge points* for each set of the corresponding well-separated pair. We identify merge points via the closest pair of nodes a and b , one from each set of the well-separated pair. The merge points are the first vertices on the shortest path between a and b outside their obstacles. Since we merge obstacles it may happen that the closest pair of nodes resides inside the same obstacle. In this case we use the first vertex on the shortest path, which usually resides inside the obstacle as merge points. We then route the links to the closest vertices on the shortest path between the merge points. This heuristic results in natural looking ribbons of links and preserves the general direction of the links when routed. We use a simple straightening heuristic to remove some of the sharper bends in the routing graph. This heuristic is allowed to move vertices in the routing graph as long as edges are not lengthened.

4.3 Drawing ribbons of links

Inspired by the ordered bundles of Pupyrev *et al.* [23] we surround each vertex in our routing graph by a circle called a *hub*. Where the distance to adjacent vertices and neighboring obstacles is sufficiently large its radius is equal to the clearance value. For each vertex u and routing edge (u, v) we construct a *ribbon base*: a line segment perpendicular to (u, v) with both endpoints on the hub (drawn in green in Fig. 5(a)). When only a single link uses edge (u, v) the two ribbon bases of the edge collapse to points.

Figure 5: Hubs and ribbon bases.

The spacing between both the ribbons and links is influenced by the amount of space provided at the ribbon base. Usually the spacing is the same for all ribbon bases, unless a hub is smaller than the clearance value or two routing edges form a small angle. In such cases a smaller spacing is used (see Fig. 5(c)) to guarantee an unambiguous drawing. Inside a hub we connect the links at the ribbon bases with bi-arcs (see Fig. 5(b)).

4.4 Link ordering and crossing minimization

In a naive routing our links might cross more often than strictly necessary. Our algorithm has to find a two-level ordering. First we want to minimize the number of crossings among the ribbons. After that we minimize the crossings of the links within a ribbon. The problem of crossing minimization has been widely studied in the context of metro-line drawings [1] and is frequently NP-complete. In the special case where one is routing simple paths which may not pass through nodes Pupyrev *et al.* [23] have shown that the optimal ordering can be calculated in linear time. In our setting the same algorithm can be used to order the links within a ribbon. However, because of the tree-like structure of the ribbons, ordering them among each other is NP-complete. We prove this by reduction from the 1-sided crossing minimization problem [19]: Given a two-layered (bipartite) graph $G = (\{L_0, L_1\}, E)$ and an ordering x_0 of vertices on layer L_0 , is there an ordering x_1 of L_1 such that number of crossings is at most k ?

Definition 1 (Ribbon ordering problem). *Given a set of ribbons going through a hub. Is there an ordering of ribbons along their bases such that the number of crossings at the hub is at most k ?*

Lemma 5. *The ribbon ordering problem is NP-Complete.*

Figure 6: An instance of the 1-sided crossing minimization problem and the corresponding ribbon ordering instance. Routing the links to different edges on the left side makes them act as the fixed layer.

Proof. We give a polynomial time reduction of one-sided crossing minimization to ribbon ordering. Consider an arbitrary instance of the one-sided crossing minimization problem $G = (\{L_0, L_1\}, E)$ where L_0 is the fixed layer. We denote the vertices in the fixed layer by $vf_1 \dots vf_{|L_0|}$ and the vertices in L_1 by $v_1 \dots v_{|L_1|}$. We construct an instance of the ribbon ordering problem for a hub h as illustrated in Fig. 6. We add routing graph edges e_i , $1 \leq i \leq |L_0|$ on the left side corresponding to the vertices in L_0 and a single routing graph edge on the right side. We add $|L_1|$ ribbons $r_1 \dots r_{|L_1|}$ to the right routing edge corresponding to the vertices in L_1 . For each edge in $(vf_i, v_j) \in E$ we add a link from ribbon r_j to e_i .

We now have a 1 to 1 mapping to the original instance allowing us to solve it by ordering the ribbons on the right edge. Since the problem is also clearly in NP this proves NP-completeness. $\square$

Because of the close relation between 1-sided crossing minimization and ribbon ordering many heuristics for the former can be adapted to the latter. In our implementation we use the *Barycenter* heuristic [19]. It calculates the “average ranking” for each ribbon along the ribbon bases. We compared the number of ribbon crossings between the barycenter order and a random ordering. Even though a significant part of the crossings are unavoidable, since they are the result of the routing, the heuristic still improves significantly on a random ordering and usually more than halves the number of crossings. A disadvantage of adapting heuristics for the 1-sided crossing minimization problem is that these treat all ribbon crossings equal, independent of the number of links in these ribbons. Better results might be achieved with an algorithm which weighs ribbon crossings based on the number of links they contain.

5 EXPERIMENTAL EVALUATION

We implemented our algorithm in C++ and compiled it with gcc using standard optimization flags. All tests were run on a machine with a 2.76 GHz Intel i5 quadcore CPU running Debian Linux. Our algorithm was implemented to run on a single core. We used three real-world diagrams from [4] and [23] (Figures 8, 11 and 12). We also generated 100 random node-link diagrams to analyze the various properties of our algorithm. These node-link diagrams consist of 100 nodes and 200 links each, both generated uniformly at random. Square obstacles were used and a clearance was chosen manually to result in naturally looking drawings throughout the set.

Performance. For our analysis of the running time we assume a linear number of links and constant complexity of the obstacles. The WSPD-based clustering takes $O(n \log n)$ time. The preprocessing of obstacles requires the computation of convex hulls and Voronoi diagrams taking $O(n \log n)$ time over all obstacles. The visibility graph and its greedy sparsification can both be computed in $O(n^2 \log n)$ time using the standard sweep line algorithm and the greedy spanner algorithm of Bose *et al.* [5]. (We implemented the $O(n^2 \log^2 n)$ greedy spanner algorithm from Bouts *et al.* [6] because of its better performance in practice.) The sparsification ensures that the routing graph contains a linear number of edges and hence we can route the $O(n)$ links in $O(n^2 \log n)$ time. The straightening, ordering and drawing takes $O(n)$ time in total, for a final running time of $O(n^2 \log n)$.

Clustering	Merging	Internal E	VisGraph	Greedy
5.07	27.23	8.34	102.50	10.13
Routing	Straightening	Ordering	Drawing	Total
13.10	18.34	10.74	17.03	212.51

Table 1: Timing statistics in milliseconds on 100 random graphs.

We tested the performance in practice by measuring the running time of different parts of our algorithm on our set of random graphs. The results are shown in Table 1. As expected from the theoretical analysis the computation of the visibility graph takes up a big part of the running time. Surprisingly the sparsification which has a similar asymptotic running time is among the fastest steps in practice.

Our sparsification method. We use the greedy sparsification of the visibility graph because of its large angles and its general sparseness. Previous work [12, 23] used a Yao sparsification. To experimentally compare the performance of these two sparsification methods we have tested them on our set of 100 randomly generated graphs. For our statistics we counted only the result of the sparsification together with the original obstacle edges. We ignored the internal edges. We computed the Greedy sparsification for a dilation value $t = 2$. Similar to other cone-based spanners the Yao-sparsification does not allow us to directly specify the desired dilation. Instead one has to specify the number of cones used. The lowest number of cones for which a dilation of 2 has been proven is 11 [2], and hence we used 11 cones. Table 2 summarizes our results, Fig. 7 shows both sparsifications on the TLR4 graph. As expected from the theoretical properties of the Yao graph the sparsification is significantly less sparse, with more than three times as many edges and almost seven times the weight of the greedy sparsification. Very important for our visualization purposes is the smallest angle, which is only 0.25 degrees in the Yao sparsification. This explains the need for the extensive optimizations necessary in previous work to make a Yao-sparsified graph suitable for routing.

	$ V $	$ E $	Min angle(deg)	Total weight
Greedy	222.4	311.2	26.45	1047.66
Yao11	222.4	966.3	0.25	7133.98

Table 2: A comparison of the greedy and Yao sparsification methods. Results are averages over 100 random inputs.

Our clustering method. One of the main features of our algorithm are the provable guaranties on the quality of the clustering. Using the separation parameter of the WSPD the user can tune the compatibility of links sharing the same ribbon in the final drawing. We used our randomly generated graphs to test how representative the theoretical bounds are. The results of our experiments can be found in Table 3, many of the values are in fact much better than the theoretical bounds.

Figure 7: The greedy (left) and Yao (right) sparsifications.

Figure 8: The TLR4 graph from [4] (91 nodes, 124 links), rendered by Cerebral [4] (see inset) and by our algorithm.

s	$ C $	Angle (rad)	Scale	Position	Visibility
0.5	88	0.94 ($\pi/2$)	2.67 (5)	0.56 (2)	0.44 (-)
1	105	0.81 ($\pi/2$)	2.09 (3)	0.44 (1)	0.35 (-)
1.5	118	0.65 (≈ 1.18)	1.90 (7/3)	0.39 (2/3)	0.31 (2/3)
2	131	0.52 (≈ 0.93)	1.74 (2)	0.29 (1/2)	0.24 (1/2)
4	159	0.35 (≈ 0.49)	1.41 (2.5)	0.21 (1/4)	0.16 (1/4)

Table 3: Number of clusters $|C|$ and the compatibility measures from Section 2 for different separation values s on our generated graphs. The maximum observed values and the upper bounds are shown.

Real-world datasets. We illustrate the results of our algorithm on three graphs of different sizes, namely the TLR4 graph from Barsky *et al.* [4] (see Fig. 8, 9, and 10), the larger Airlines graph used in various previous work (see Fig. 11), and the small Tail graph from Pupyrev *et al.* [23] (see Fig. 12). The blue regions in Fig. 8 correspond to the green regions in the original rendering by the

Cerebral software (see inset) and represent additional information in the data not used by our algorithm.

For the small Tail graph we use $s = \infty$ which effectively disables clustering. For the bigger TLR4 graph we use $s = 1.5$ and for the Airlines graph we use $s = 1$. For the TLR4 graph input clusters were defined based on the function of the different biomolecules represented by the nodes, we refined these clusters.

We used a dilation of $t = 1.8$ in the sparsification step for the TLR4 and Airlines graph, so any detour caused by the sparsification is at most 1.8 times as long as the shortest path in the original graph. For the small Tail graph we used $t = 2$. Generally speaking a higher dilation results in a more abstract visualization whereas a lower dilation preserves the direction of links better. Fig. 9 shows the effect of different values of t on TLR4.

Comparison with other techniques. The TLR4 graph from Barsky *et al.* [4] represents interactions between biomolecules and link clusters have been defined by domain experts. Fig. 8 shows our method on the TLR4 graph using the node-layout calculated by

Figure 9: By varying the dilation one can choose between (a,b) a more direction preserving or (c) a more schematic and abstract layout

Figure 10: Detail of the TLR4 graph from Fig. 8 as drawn by (a) Cerebral [4] and (b) our algorithm.

Cerebral [4], the original rendering is shown as an inset.

Many bundling algorithms employ heuristics, like the force-directed approach of Holten *et al.* [17] or the proximity based splines of Barsky *et al.* [3]. Such methods often lack the control to ensure links, or bundles of links, are drawn unambiguously. See, for example, Fig. 10 which shows a detail of the TLR4 graph. The drawing by Cerebral has links which overlap each other and links which are overlapped by nodes. Cerebral’s drawing nicely preserves the direction of the links, highlighting the high-level structure of the network. However, in a setting where one is mostly interested in the connectivity of the nodes, we would like to argue that our approach gives a much cleaner and, most importantly, unambiguous visual summary of the network.

We now compare the results of our algorithm with the ordered bundles algorithm by Pupyrev *et al.* [23] on the larger Airlines graph (Fig. 11) and the small Tail graph (Fig. 12). Similar to our method the ordered bundles algorithm uses a routing graph and node obstacles to prevent links from overlapping nodes. The ordered bundles algorithm draws every link as a separate path and performs no bundling to allow the user to trace individual links. We use a hybrid approach where we draw ribbons of links. This drawing style is more explicit than the common solution of drawing clustered links on top of each other, it allows the user to follow

Figure 11: (a) The ordered bundles algorithm from [23] and (b) our method with two ribbons highlighted on the US Airlines graph (235 nodes, 2101 links).

Figure 12: The Tail graph from [23] (8 nodes, 28 links) as rendered by (a) the ordered bundles algorithm and (b) our method. In (a) the edges are drawn as bands using double lines, the single lines are part of the background image and correspond to our blue lines.

a link comparatively easily, and it is also compact enough for larger graphs. Since ribbons are clearly separated one can more easily follow groups of links from the same area. Our algorithm performs similar to the ordered bundles algorithm on the small Tail graph and gives an arguably cleaner picture on big US Airlines graph.

6 CONCLUSION AND FUTURE WORK

We presented a link clustering algorithm based on a well-separated pair decomposition. Our algorithm can be used directly on an input graph or it can be used to refine given clusters. It routes clusters as ribbons of links on a sparse visibility spanner. Our algorithm can easily be tuned by adjusting the separation constant of the WSPD (to influence the clustering) or the dilation of the routing graph (to find a balance between a more direction preserving and a more clean and abstract layout). All links in the resulting visualization can be traced individually and unambiguously (possibly assisted by interactive highlighting).

An interesting topic for future work is an improved routing graph which has better dilation bounds for paths used by ribbons consisting of many links. Such an *input sensitive spanner* would ideally still exhibit the sparsity of a higher dilation spanner, but it would also preserve the directionality of the more prominent features of the node-link diagram. In a setting where node positions are not specified it would be interesting to explore combinations of smart node placement and advanced routing and clustering techniques.

Acknowledgements. The authors would like to thank Tim Dwyer for fruitful discussions on the topic of this paper and Tamara Munzner for the Cerebral use case. Quirijn Bouts and Bettina Speckmann are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208.

REFERENCES

- [1] E. Argyriou, M. A. Bekos, M. Kaufmann, and A. Symvonis. On metro-line crossing minimization. *Journal of Graph Algorithms and Applications*, 14(1):75–96, 2010.
- [2] L. Barba, P. Bose, M. Damian, R. Fagerberg, W. L. Keng, J. O’Rourke, A. van Renssen, P. Taslakian, S. Verdonschot, and G. Xia. New and improved spanning ratios for yao graphs. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry, SOCG ’14*, pages 30–39, New York, NY, USA, 2014. ACM.
- [3] A. Barsky. Cerebral: Visualizing Multiple Experimental Conditions on a Graph with Biological Context. *Master’s Thesis, University of British Columbia*, 2008.
- [4] A. Barsky, T. Munzner, J. L. Gardy, and R. Kincaid. Cerebral: Visualizing Multiple Experimental Conditions on a Graph with Biological Context. *IEEE TVCG*, 14(6):1253–60, 2008.

- [5] P. Bose, P. Carmi, M. Farshi, A. Maheshwari, and M. Smid. Computing the greedy spanner in near-quadratic time. *Algorithmica*, 58(3):711–729, 2010.
- [6] Q. W. Bouts, A. P. ten Brink, and K. Buchin. A framework for computing the greedy spanner. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry, SOCG ’14*, pages 11–19, New York, NY, USA, 2014. ACM.
- [7] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42(1):67–90, 1995.
- [8] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE TVCG*, 14(6):1277–84, 2008.
- [9] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer-Verlag, 3rd edition, 2008.
- [10] M. Dickerson, D. Eppstein, M. Goodrich, and J. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. In *Proc. 11th International Symposium on Graph Drawing, LNCS 2912*, pages 1–12, 2004.
- [11] T. Dwyer, K. Marriott, and M. Wybrow. Integrating edge routing into force-directed layout. In *Proc. 14th International Symposium on Graph Drawing, LNCS 4372*, pages 8–19, 2007.
- [12] T. Dwyer and L. Nachmanson. Fast edge-routing for large graphs. In *Proc. 17th International Symposium on Graph Drawing, LNCS 5849*, pages 147–158, 2010.
- [13] O. Ersoy, C. Hurter, F. V. Paulovich, G. Cantareiro, and A. Telea. Skeleton-based edge bundling for graph visualization. *IEEE TVCG*, 17(12):2364–73, 2011.
- [14] E. R. Gansner, Y. Hu, S. North, and C. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. *IEEE Pacific Visualization Symposium*, pages 187–194, 2011.
- [15] E. R. Gansner and Y. Koren. Improved circular layouts. In *Proc. 14th International Symposium on Graph Drawing, LNCS 4372*, pages 386–398, 2007.
- [16] D. Holten. Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. *IEEE TVCG*, 12(5):741–8, 2006.
- [17] D. Holten and J. J. Van Wijk. Force-Directed Edge Bundling for Graph Visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.
- [18] C. Hurter, O. Ersoy, and A. Telea. Graph Bundling by Kernel Density Estimation. *Computer Graphics Forum*, 31(3):865–874, 2012.
- [19] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1(1):1–25, 1997.
- [20] A. Lambert, R. Bourqui, and D. Auber. Winding roads: Routing edges into bundles. *Computer Graphics Forum*, 29(3):853–862, 2010.
- [21] S.-J. Luo, C.-L. Liu, B.-Y. Chen, and K.-L. Ma. Ambiguity-free edge-bundling for interactive graph visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 18(5):810–821, May 2012.
- [22] G. Narasimhan and M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.
- [23] S. Pupyrev, L. Nachmanson, S. Bereg, and A. Holroyd. Edge routing with ordered bundles. In *Proc. 19th International Symposium on Graph Drawing, LNCS 7034*, pages 136–147, 2012.
- [24] S. Pupyrev, L. Nachmanson, and M. Kaufmann. Improving layered graph layouts with edge bundling. In *Proc. 18th International Symposium on Graph Drawing, LNCS 6502*, pages 329–340, 2011.
- [25] G. Quercini and M. Ancona. Confluent drawing algorithms using rectangular dualization. In *Proc. 18th International Symposium on Graph Drawing, LNCS 6502*, pages 341–352, 2011.
- [26] A. Telea and O. Ersoy. Image-based edge bundles: Simplified visualization of large graphs. *Computer Graphics Forum*, 29(3):843–852, 2010.