MASTER

Edge bundling and routing via well-separated pair decomposition and visibility spanners

Bouts, Q.W.

Award date:
2013

Link to publication
Master’s Thesis

Edge Bundling and Routing via Well-Separated Pair Decomposition and Visibility Spanners

by

Quirijn W. Bouts

Supervisor
prof.dr. Bettina Speckmann

Eindhoven, August 21, 2013
Abstract

We present a novel edge bundling algorithm which defines its bundles based on a well-separated pair decomposition and routes bundles individually on a sparse visibility spanner. We prove that the bundles induced by the well-separated pair decomposition consist of compatible edges according to the measures by Holten and Van Wijk [16]. The greedy sparsification of the visibility graph allows us to easily route around obstacles and guarantees a bound on the detour of the shortest paths between vertices. Our experimental results are visually appealing and convey a sense of abstraction and order. Edge clutter is significantly reduced while the individual routing of the bundles retains nearly all connectivity information.
Acknowledgements

I would like to thank my supervisor Bettina Speckmann for her guidance and valuable feedback throughout the project. In addition, I would like to thank Kevin Buchin and Jack van Wijk for reviewing this thesis and serving on the examination committee.

Finally, I would like to thank my friends who supported me not only during my graduation but throughout my academic education. Specifically, I would like to thank Sander Alewijnse and Alex ten Brink; together we worked on many interesting projects which helped me prepare for my graduation.

Quirijn W. Bouts

Eindhoven
August 21, 2013
Contents

Abstract iii

Acknowledgements v

1 Introduction 1

2 Edge bundling via well-separated pair decomposition 7
  2.1 Computing a well-separated pair decomposition ....................... 8
  2.2 Edge compatibility measures ...................................... 9

3 Edge routing via sparse visibility spanners 11
  3.1 The visibility graph ............................................. 11
  3.2 The greedy spanner ............................................. 11
    3.2.1 Properties of the greedy spanner ......................... 12
  3.3 A greedy sparsification of the visibility graph ..................... 13

4 Computing the visualization 15
  4.1 Preprocessing obstacles ....................................... 15
  4.2 Routing bundles ............................................... 16
    4.2.1 Drawing bundles ........................................ 16
    4.2.2 Calculating biarcs ...................................... 17
  4.3 Bundle ordering and crossing minimization ........................ 18
    4.3.1 A heuristic for bundle ordering ......................... 19

5 Experimental Evaluation 21
  5.1 Performance .................................................... 21
  5.2 Parameters ..................................................... 22
  5.3 Comparison with other techniques ............................... 23

6 Conclusion and future work 29
Chapter 1

Introduction

Graphs are an important tool to express relational data, that is, data which consists of objects and their interactions. Common examples include airline and migration routes, 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, when visualizing large graphs they can appear very cluttered. If the vertices cannot be moved (because the graph represents georeferenced data such as traffic networks) then the only option to reduce clutter is to change the shape and routing of the edges. Correspondingly recent years have seen a significant number of papers that address this issue. In particular, various techniques have been proposed to reduce edge clutter by bundling edges which are “close”, be it geometrically in a straight line drawing of the graph or conceptually in a hierarchical structure of the data.

Contribution. We present a novel edge bundling algorithm which defines its bundles based on a well-separated pair decomposition and routes bundles individually on a sparse visibility spanner. 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 input is a graph with fixed vertex positions and optionally a set of disjoint polygonal obstacles. Our algorithm ensures that bundles do not get close to vertices and do not cross obstacles.

The well-separated pair decomposition was first introduced by Callahan et al. [4]. 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 [16] introduced four edge compatibility measures which indicate how similar edges are and influence how likely it is that particular edges are bundled in their force-directed approach. They evaluate these measures based on a straight line drawing of the input graph and its iterative improvements. We argue that the bundles induced by a well-separated pair decomposition consist of compatible edges according to the measures of Holten and Van Wijk. Specifically, the separation constant of the well-separated pair decomposition gives users a single parameter to vary to find a balance between few bundles of less compatible edges and many bundles of very compatible edges. By definition, our bundling is based solely on the geometric locations of the input vertices. In addition, the properties of the well-separated pair decomposition allow us to handle very dense graphs.

We route the edge bundles defined by the well-separated pair decomposition along a routing graph. A similar approach was proposed by Pupyrev et al. [21]. Our algorithm differs from theirs in two aspects. First of all, Pupyrev et al. draw every edge separately in a metro line style layout. This quickly becomes infeasible if the number of edges is large. We use a hybrid approach where each bundle is drawn individually. This drawing style is more explicit than the common solution of drawing all bundles on top of each other. It allows the user to follow
a bundle comparatively easily and is also compact enough for large graphs. We use color to indicate the number of edges in each bundle (see Figure 1).

The second difference with Pupyrev et al. is the routing graph we use, namely the greedy sparsification of the visibility graph. The visibility graph can easily deal with obstacles. By using a spanner as a sparsification method we maintain a bound on the detour of the shortest path between any two vertices in the graph. That is, every edge is routed along a path that is only a little longer than the straight line connection between its endpoints. On the complete graph a greedy sparsification has several desirable properties, such as a provable angle constraint as well as a total weight of $O(MST)$ where $MST$ is the weight of the minimum spanning tree. Although we can prove that neither of these properties hold for a greedy sparsification of the visibility graph, our experiments show that our routing graph still exhibits most of them. Our routing graph can be easily computed in two steps: visibility graph and sparsification. In contrast, Pupyrev et al. start with a very dense routing graph of low quality and then go through an extensive and computationally expensive iterative optimization step which is not guaranteed to deliver results of any provable quality.

Our method is designed to work best with medium sized graphs, but it can also deal with graphs of several thousand vertices. The experimental results are visually appealing and convey a sense of abstraction and order. Edge clutter is significantly reduced but nevertheless the individual routing of the bundles still retains nearly all connectivity information. This allows the user to easily identify high-level edge patterns while still being able to trace individual routes.

**Organization.** In Section 2 we show how to compute the well-separated pair decomposition and prove that the bundles induced by it consist of compatible edges according to the measure introduced by Holten and Van Wijk. We then in Section 3 discuss the greedy spanner and the construction of our routing graph. In Section 4 we describe our complete drawing algorithm in detail, including pre-processing, bundle ordering, and crossing minimization. Finally, experimental results are presented in Section 5.

**Related work.** Gansner et al. [12] were among the first to use edge bundling. They presented an algorithm which places the vertices on the perimeter of a circle (see Figure 2). By finding a suitable placement of vertices they try to avoid long edges. To reduce clutter they route edges either on the inside or outside of the circle. The idea of *ink minimization* plays an important role in their final step where edges are drawn as bundled splines. Ink minimization has been used in many other bundling techniques.
Holten and Van Wijk [16] describe a force directed approach. To enable the straight line versions of edges to be bundled together they subdivide them into segments. For each edge a linear attracting spring force is used between each pair of consecutive subdivision points. Using a global spring constant they control the stiffness of edges and hence the amount of bundling. A second “electrostatic” attracting force is defined between interacting edges to pull them towards each other. This attracting force is determined by the four aforementioned compatibility measures.

The Divided edge bundling algorithm by Selassie et al. [24] incorporates edge directions into a force directed approach. Their method tries to better preserve the graph topology and highlight directional patterns not visible in when using an approach for undirected edge bundling. By drawing bundles with a different width corresponding to the aggregate weight of the their edges they can visualize the total bundle weight.

Dwyer et al. [8] investigate how to integrate edge routing techniques into a force-directed layout. By imposing separation constraints on the force directed model they ensure that vertices do not overlap edges or other vertices. Their method requires a feasible initial routing and moves vertices, which makes it not applicable if object locations are fixed.

As mentioned above, Pupyrev et al. [21] avoid obstacles by using a routing graph based on (a sparsification of) the visibility graph which is later optimized. Figure 5 shows their algorithm on the US migration graph. In Section 5.3 we thoroughly compare their routing graph to the graph used by our algorithm.

Cui et al. [5] propose a geometry-based approach which uses a control mesh to bundle edges. The graph is first divided into a grid. For each cell the primary direction of the edges in the cell is determined and clusters of cells with a similar direction are then merged together. Mesh edges are set perpendicular to the cluster directions and linked together to form the control mesh. At each control mesh edge, the edges crossing it are pulled together. Some local smoothing is used to further increase the quality of the bundling.

Lambert et al. [19] describe a related technique, the Winding roads algorithm. Similar to the geometry based approach of Cui et al. they also overlay the graph with a mesh to guide the bundling. They use a combination of the Voronoi diagram and a quadtree as a multi-resolution mesh and continue to route the edges along this mesh. They promote bundling by using lower weights for mesh edges which are used in many shortest paths.

Colors and opacity [5,15] as well as splatting and bump mapping [19,25] are used in several techniques to enhance a bundling.

Telea et al. [25] compute a hierarchical edge clustering of a given graph layout which groups similar edges into clusters. Using an image based technique which combines distance-based splatting and skeletonization they render the clusters.

A similar approach is used by Ersoy et al. [10], who iteratively bundle edges using clustering, distance fields and skeletonization. The result is a colorful organic looking bundling (see Figure 8).

Very recently Hurter et al. [17] proposed to use image sharpening techniques to merge local height maxima on a kernel density map to find a bundling. In several iterations they move edges towards the denser regions starting from a straight line drawing. By modifying the density map in each iteration they can create bundled layouts which avoid obstacles. If edge endpoints are located inside obstacles the algorithm can either move the endpoints to the obstacle boundary or leave them inside the obstacle depending on a user specified setting. An example of the technique with endpoint displacement enabled is shown in Figure 9.

In addition, some specifically fast techniques have been developed for large graphs. Dwyer et al. [9] explore two different techniques to approximate shortest path routing in a visibility graph without calculating the full visibility graph. Using a spacial decomposition imposed by a KD-tree they simplify the obstacles by grouping those further away in their joint convex hulls. The result is an $O(\log^2 n \log \log n)$ algorithm. A second approach which they explore is the use of a cone based spanner of the visibility graph which can be computed in $O(n \log n)$ time.

Gansner et al. [13] treat edges as points in four dimensional space, which allows for a simple distance metric in that space to define edge similarity. Using several levels of agglomerative bundling, based on a proximity graph in this four dimensional space, their approach is able to bundle hundreds of thousands of edges in a matter of seconds.

Aside from techniques for general graphs, there are also algorithms specifically targeted at hierarchical data and for layered graphs.
Holten [15] describes a technique for hierarchical data which bundles the adjacency, i.e., non-hierarchical edges towards the polylines defined by the path via the inclusion edges from one vertex to another. This emphasizes the parent child relation while reducing clutter.

Pupyrev et al. [22] give a method for bundling layered graphs using a metro-style drawing method. The algorithm is based on ink minimization. Figure 10 shows an example of the bundling of a layered graph.

Confluent drawings, proposed by Dickerson et al. [6], draw certain non-planar graphs in a planar way by merging edges. A confluent drawing of a graph has a one to one mapping with the original graph. That is, every edge in the drawing is also in the original graph. In a bundled drawing edges which do not form a complete bipartite graph may still be bundled together. Furthermore, in a bundling crossings are still allowed whereas a confluent drawing is planar. As a result not all graphs have a confluent drawing.

More recently Quercini et al. [23] used the idea of confluent drawing in combination with rectangular dualization to draw planar versions of confluent graphs.
Chapter 2

Edge bundling via well-separated pair decomposition

In this chapter we first give a more detailed definition of the well-separated pair decomposition and show how it can be calculated. We then prove that the edges bundled by the decomposition are compatible according to the compatibility measures of Holten and Van Wijk [16].

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, in two dimensional Euclidean space, point sets $A$ and $B$, with bounding boxes $R(A)$ and $R(B)$, are as 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 $C_A$ (see Figure 11).

Figure 11: Two pointsets that are $s$-well-separated. The circles have diameter $D$ and their distance is at least $sD$. 
More formally, the well-separated pair decomposition 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 decomposition. Every well-separated pair $(A_i, B_i)$ induces an edge bundle: each edge with one endpoint in $A_i$ and the other endpoint in $B_i$ is part of the bundle.

### 2.1 Computing a well-separated pair decomposition

In this section we give a sketch of the computation of the well-separated pair decomposition. More details on this calculation and a thorough running time analysis can be found in [20] Chapter 9 (note that they use the radius instead of the diameter in their definition of well-separated).

We construct the well-separated pair decomposition of a planar point set $S$ in two steps. First we calculated a binary tree, called a split tree, on $S$ which is not dependent on the separation constant. In the next step we use the split tree to construct the decomposition.

The split tree can be constructed by recursively “splitting” the bounding box of the point set in a node to define its two child nodes. The resulting rooted binary tree contains the point set $S$ in its leaves. By splitting in a specific order one can guarantee that the split tree is balanced in the sense that each node splits its point set evenly. As a result the height of such a balanced split-tree is $O(\log n)$ and it can be computed in $O(n \log n)$ time (see [20] Chapter 9.3.2).

For each internal (non-leaf) node $u$ with children $v$, $w$ in the split tree we run $\text{FindPairs}(v, w)$. This subroutine first check if $v$ and $w$ are well-separated. If they are it reports the pair, otherwise it looks at the child with the bigger longest bounding box edge. Say $v$ is this “bigger” node. It now recursively calls $\text{FindPairs}(u, v_l)$ and $\text{FindPairs}(u, v_r)$, where $v_l$ and $v_r$ are the children of $v$.

Since we use bounding boxes in our definition of well-separated we can store bounding box information in the split tree construction and use it to determine if two nodes are well-separated in constant time. Using that the split tree is a balanced binary tree with height $O(\log n)$ and $O(n)$ nodes we can construct a well-separated pair decomposition in $O(n \log n)$ time for any constant $s > 0$. 
2.2 Edge compatibility measures

Holten and Van Wijk introduced four compatibility measures which concern the angle, scale, and position of a pair of edges, as well as the visibility between them. Below we describe each measure and argue that two edges in the same bundle induced by the well-separated pair decomposition are compatible. Consider the \( s \)-well-separated pair \( \{A, B\} \). We examine the compatibility measures on any two edges \( e = (p_0, p_1) \) and \( f = (q_0, q_1) \) with \( p_0, q_0 \in A \) and \( p_1, q_1 \in B \). We assume that the edge \( e \) is at most as long as the edge \( f \). We use \( D \) to denote the diameter of the circles around the sets of the pair.

**Angle compatibility.** Edges in the same bundle should have a similar angle. We define the angle \( \alpha \) between two non-parallel edges as the smallest angle between the lines induced by the edges, this is illustrated in the figure on the right. The angle of parallel edges is defined as 0.

**Lemma 1** The angle \( \alpha \) between \( e \) and \( f \) is at most \( 2 \cdot \tan^{-1}(\frac{1}{s}) \) for \( s \geq 1 \).

**Proof.** The figure on the right shows a worst case configuration of \( e \) and \( f \) with respect to \( \alpha \). We have \(|(p_0, t)| \leq 0.5D\) and \(|(t, r)| \geq 0.5sD\) as rough bounds. By looking at the right-angled triangle \( \triangle p_0tr \) we can now bound \( \alpha \) by \( 2 \cdot \tan^{-1}(\frac{1}{s}) \). □

**Scale compatibility.** Edges in the same bundle should have similar length. We define the scale compatibility as the ratio between the length of the shorter and the length of the longer edge.

**Lemma 2** The difference in length of \( e \) and \( f \) is at most \( 2 \cdot D \). Furthermore the length ratio between \( e \) and \( f \) can be bounded by \( \frac{|e|}{|f|} \geq \frac{s}{s+2} \).

**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 \( \frac{|e|}{|f|} \geq \frac{s}{s+2} \). □

**Position compatibility.** Edges which are close to each other should be more likely to end up in the same bundle. Holten and Van Wijk measure “close to each other” by considering the distance between the midpoints \( p_m \) and \( q_m \) of edges \( e \) and \( f \) in relation to the average edge length of \( e \) and \( f \).

**Lemma 3** The difference in position of edges \( e \) and \( f \) with midpoints \( p_m \) and \( q_m \) is \(|(p_m, q_m)| \leq D\). Furthermore the ratio between the average length and the difference in position 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 \( \frac{|p_m q_m|}{(|e|+|f|)/2} \leq \frac{D}{sD} = \frac{1}{s} \). □
Visibility compatibility. The visibility compatibility of $e$ with $f$ is defined by the normalized distance between the midpoint of $e$ ($p_m$), and the point $q_m'$ on the line induced by $e$ which when projected onto $f$ coincides with its midpoint $q_m$. 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 $q_m$, respectively. We can bound the visibility compatibility by $|\langle p_m, q_m' \rangle| \leq \frac{1}{s}$ for $s > 1$.

**Proof.** We use Lemma 1 to bound the smallest angle $\alpha$ between the lines induced by $e$ and $f$ by $\alpha \leq 2 \cdot \tan^{-1}\left(\frac{1}{s}\right)$. Let $\overline{p_m}$ be the projection of $p_m$ onto the line induced by $f$. We have $|\langle \overline{p_m}, q_m \rangle| \leq |\langle p_m, q_m \rangle| \leq D$ by the triangle inequality. This implies $|\langle p_m, q_m' \rangle| \leq \frac{D}{\cos(\alpha)} = \frac{D}{\cos(2 \cdot \tan^{-1}\left(\frac{1}{s}\right))} = \frac{D(s^2+1)}{s^2-1}$. From the definition of $s$-well-separated we have $|\langle q_0, q_1 \rangle| \geq sD$, which implies that $|q_0', q_1'| \geq \frac{sD}{\cos(\alpha)}$ which simplifies to $\frac{sD(s^2+1)}{s^2-1}$. We now have $|\langle p_m, q_m' \rangle| \leq \frac{sD(s^2+1)}{s^2-1} \cdot \frac{1}{s} = \frac{s^2-1}{sD(s^2+1)}$.

Increasing the separation constant gives better values for all four compatibility measures, that is, the edges in a bundle become more compatible. Hence the separation constant of the well-separated pair decomposition gives users a single parameter to vary to find a balance between few bundles of less compatible edges and many bundles of very compatible edges.
Chapter 3

Edge routing via sparse visibility spanners

In this section we give a more detailed description of our routing graph; the greedy sparsification of the visibility graph. In general there are several desirable properties for a routing graph. It should be sparse to promote bundling. It should not have too many small angles to avoid clutter. It should contain many small edges, so edges in the same bundle can merge quickly. The shortest path between two points in the routing graph should be relatively short, not much longer than their direct connection. Since we intend to draw bundles 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.

3.1 The visibility graph

Let \(S\) be a set of disjoint polygonal obstacles in the plane with \(n\) obstacle vertices in total. The visibility graph \(G\) of \(S\) contains an edge for every pair of vertices which are visible to each other, where vertex \(v\) 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 \(G\) has the property that for any two vertices the shortest path around the obstacles is contained in \(G\).

In a preprocessing step we add a small simple polygon around each vertex in the input to ensure that the routes do not overlap the vertices. Since the visibility graph is defined on disjoint polygons we may need to merge vertex obstacle polygons if they overlap. Details can be found in Section 4.1. Using a simple sweepline approach the visibility graph can be calculated in \(O(n^2 \log n)\) time (see [2] Chapter 15). The optimal running time for computing the visibility graph is \(O(n \log n + k)\), where \(k\) indicates the number of edges in the visibility graph. The more complex algorithm of Gosh and Mount [14] achieves this running time.

3.2 The greedy spanner

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 can be constructed by considering all edges \(e = (p, q) \in E\) in non-decreasing order and adding them to \(E'\) if and only if \(d_{G'}(p, q) > t \cdot d_G(p, q)\). Since the greedy spanner has a linear number of edges \(d_{G'}(p, q)\) can be computed in \(O(n \log n)\) time using Dijkstra’s shortest path algorithm. The resulting running time for the original naive implementation of the greedy spanner is therefore \(O(n^3 \log n)\).

A lot of research has been done into the properties and more efficient calculation of the greedy spanner. Farshi and Gudmundsson [11] introduced an improvement on the naive greedy algorithm which was later called FG-greedy. The algorithm is similar to the naive algorithm
but uses a matrix to store shortest paths and updates the values in this matrix only when needed. They showed that the algorithm calculated the greedy spanner in $O(n^2 \log n)$ time in practice and conjectured that, with the use of this matrix, the number of shortest path queries could be bounded by $O(n)$ which would imply the $O(n^2 \log n)$ running time. This conjecture was later disproved in [3] where a slightly altered version of the FG-greedy algorithm was proposed which had a provable $O(n^2 \log n)$ running time. We use the original FG-greedy algorithm in our implementation because of its good performance in practice.

### 3.2.1 Properties of the greedy spanner

The greedy spanner as a sparsification of the complete graph has several interesting properties. We give a summary of the properties which are of interest in the routing graph construction here. Detailed proofs of these properties have been given in Chapter 14 of [20].

#### Definition 1 [Leapfrog property] Let $t > 1$ be a real number. A set $E$ of undirected edges in $\mathbb{R}^d$ is said to satisfy the $t$-leapfrog property, if for every $k \geq 2$, and for every sequence $\{p_1, q_1\}, \{p_2, q_2\}, \ldots, \{p_k, q_k\}$ of $k$ pairwise distinct edges of $E$,

$$t|p_1q_1| < \sum_{i=2}^{k} |p_iq_i| + t \left( |p_1p_2| + \sum_{i=2}^{k-1} |q_ip_{i+1}| + |q_kq_1| \right).$$

#### Lemma 5 Let $t > 1$ be a real number, and let $E$ be a set of undirected edges in $\mathbb{R}^d$ satisfying the $t$-leapfrog property. Let $\theta$ be a real number, such that $0 < \theta < \pi/4$ and $\cos\theta - \sin\theta \geq 1/t$. Let $(p, q)$ and $(r, s)$ be two distinct edges in $E$ with $\angle(pq, rs) \leq \theta$. Then, the directed edges $(p, q)$ and $(r, s)$ satisfy the strong $w$-gap property, for

$$w = \frac{1}{2}(\cos\theta - \sin\theta - 1/t).$$

Furthermore it can be shown that

#### Lemma 6 Let $t > 1$ be a real number, let $S$ be a set of $n$ points in $\mathbb{R}^d$ and let $G = (S, E)$ be an undirected graph, such that $E$ satisfies the $t$-leapfrog property.

1. The degree of every vertex of $G$ is $O(1/(t-1)^{d-1})$.

2. The weight $\text{wt}(E)$ of $E$ satisfies

$$O \left( \left( \frac{1}{(t-1)^d} \cdot \text{wt}(\text{MST}(S)) \log n \right) \right),$$

where $\text{MST}(S)$ denotes the minimum spanning tree of $S$

Using a stronger version of the leapfrog property, referred to as the leapfrog theorem in [20] an even stronger bound of $O(c_{dt} \cdot \text{wt}(\text{MST}(S))$ can be proven, where $c_{dt}$ is a constant depending only on $d$ and $t$. 

12
To summarize, on the complete graph the greedy sparsification has both an angle and a degree constraint which can be derived from the strong gap property. Furthermore the total weight can be bounded by a constant times the weight of the minimal spanning tree. Since the sparsification is a spanner the resulting graph also has a bound on the detour of the shortest path between vertices compared to the direct euclidean distance.

### 3.3 A greedy sparsification of the visibility graph

The greedy spanner as a sparsification of the complete graph has almost all good properties of a routing graph. However, it is not able to deal with obstacles. By using a greedy sparsification of the visibility graph we are able to deal with obstacles but lose the angle and weight bound.

![Figure 13: A comb-shaped obstacle with the minimal spanning tree on the point set (a) and on the visibility graph (b) highlighted in gray.](image)

Lemma 7 The total weight of the greedy sparsification of the visibility graph can be $O(n \cdot \text{wt}(\text{MST}))$, where $\text{wt}(\text{MST})$ denotes the weight of the minimal spanning tree.

*Proof.* Consider the visibility graph of a single comb shaped obstacle as depicted in Figure 13. As we make the teeth smaller, adding more without increasing the width of the comb the weight of the minimal spanning tree over the point set (highlighted in Figure 3.13(a)) remains roughly the same. Since the visibility graph does not contain edges crossing obstacles the total weight of the minimal spanning tree on the graph $G$ (highlighted in Figure 3.13(b)) grows linearly with the number of teeth. By construction the greedy spanner of a graph $G$ contains the minimal spanning tree on that graph. Hence the weight of the greedy spanner on the visibility graph can be $O(n \cdot \text{wt}(\text{MST}))$. □

Lemma 8 The minimum angle in the greedy sparsification of the visibility graph can be arbitrarily small.

*Proof.* Again we use that the greedy spanner of a graph $G$ contains the MST on that graph, $\text{MST}(G)$. Consider a visibility graph on a single diamond shaped obstacle. $\text{MST}(G)$ must contains three of the four diamond edges and hence also one degree two vertex with the smallest angle of the diamond. By flattening the diamond we can make this angle arbitrarily small. Hence the angle in the greedy sparsification of the visibility graph can also be arbitrarily small. □

As we have shown above, both the bound of $O(\text{wt}(\text{MST}))$ on the weight of the greedy spanner and the gap property and its associated angle constraint do not hold for the greedy sparsification of the visibility graph. However our experiments show that, on our well-behaved input, our routing graph still exhibits most of them.
Computing the visualization

We now describe our complete drawing algorithm in detail, including pre-processing, bundle ordering and crossing minimization. Our algorithm has two separate phases. In the first phase the bundles are defined according to the well-separated pair decomposition, in the second phase these bundles are routed and drawn. The exact sequence of steps and pointers to their description in this thesis are given in the algorithm below.

Algorithm
1. Compute WSPD (see Chapter 2)
2. Pre-processing obstacles (see Section 4.1)
3. Compute visibility graph (see Section 3.1)
4. Greedy sparsification (see Section 3.3)
5. Route, draw, and order bundles (see Sections 4.2–4.3)

Our input consists of a graph \( G = (V, E) \) with fixed vertex positions and optionally a list of obstacle polygons. We refer to the input vertices as terminals. As mentioned before we add a small simple polygon around each terminal and then pre-process and merge these terminal obstacle polygons in such a way that the resulting obstacles are all disjoint. Suitable internal edges are added to connect the terminals to their obstacle vertices. Next the visibility graph is calculated on the merged obstacles and sparsified using the greedy spanner algorithm. When routing a bundle only the internal edges of the obstacles in which the endpoints of the bundled edges lie are used, i.e., a bundle may only enter an obstacle if its endpoints are inside that obstacle. The bundles are drawn parallel to each other along the routing graph edges. The space around the routing graph edges allocated for drawing bundles is controlled by a clearance value \( c \). Finally, we use a heuristic to minimize crossings between bundles at routing graph vertices.

4.1 Preprocessing obstacles

We merge all (terminal) obstacle polygons which overlap or are too close to each other (see Figure 15 (a)). What is considered “too close” can be specified by the user relative to the clearance \( c \). 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 Figure 14). Merged obstacles contain multiple terminal vertices.

Figure 14: detail of Figure 25.
To correctly connect these terminals to vertices of their joint convex hull, we calculate the Voronoi diagram of the terminals inside the merged obstacle (see Figure 15 (b)). The Voronoi edges will give us the best possible clearance inside the obstacles. However, the Voronoi edges may induce vertices on the obstacle boundary which are close to each other or close to the obstacle vertices. We use a snapping heuristic to move the Voronoi vertices towards each other or towards the hull vertices (see Figure 15 (c)). We do not snap vertices if this would put multiple terminal vertices into a single snapped Voronoi cell. Finally, we connect the terminal vertices with the vertices of its (snapped) Voronoi cell, using a simple angle constraint to ensure that the edges connecting terminal vertices to Voronoi vertices are not too close.

\[\begin{array}{c}
\text{(a)} \\
\text{(b)} \\
\text{(c)}
\end{array}\]

Figure 15: (a) Obstacles are too close. (b) Voronoi diagram within convex hull. (c) Snapped Voronoi diagram with internal edges added.

4.2 Routing bundles

Clearly the routing algorithm needs to respect the bundles. Specifically, edges from the same bundle should share a common sub-path in the routing graph. For each bundle we choose two routing graph vertices as merge points for each set of the corresponding well-separated pair. We then use the shortest path via these merge points to route the edges into a single bundle. We identify merge points via the closest pair of points \(a\) and \(b\), one from each set of the well-separated pair. The merge points are the first points on the shortest path between \(a\) and \(b\) outside the terminal obstacles. Since we merge obstacles in rare cases it may happen that the closest pair of vertices reside inside the same obstacle. In this case we use the first non-terminal vertex, which usually resides inside the obstacle. This closest pair heuristic results in natural looking bundles which tend to preserve the general direction of the original edges when routed.

Assuming realistic obstacles we can bound the distance from a vertex to each merge point via the separation constant, and hence obtain a total bound on the detour an edge makes when it is bundled with other edges. Finally, we remove all vertices of degree 2 from the used part of the routing, if the corresponding shortcut does not intersect obstacles or creates a small angle with another edge.

4.2.1 Drawing bundles

Inspired by the Ordered bundles of Pupyrev et al. [21] we surround each vertex in our routing graph by a circle. We call these circles hubs and denote the hub of vertex \(v\) by \(h_v\) with radius \(r(h_v)\). Where there is enough space \(r(h_v)\) is equal to the clearance value \(c\). For each vertex \(u\) and edge \((u,v)\) we construct a bundle base: a line segment perpendicular to \((u,v)\) with both endpoints on \(h_v\) (drawn in red in Figure 4.16(a)). When only a single bundle uses edge \((u,v)\) the two bundle bases of the edge collapse to points.
Figure 16: Bundles are drawn using line segments and biarcs.

The part of the bundle which is drawn as a line segment along the edge and connects to the bundle bases is called a bundle segment. The spacing between the bundle segments is influenced by the amount of space provided at the bundle base. Usually this space is the same for all bundle bases, unless a hub is smaller than the clearance value or two routing edges form a small angle. In such cases a smaller spacing may be required (see Figure 16 (c)).

The effective space which can be used to draw bundles between two routing graph edges is reduced by the minimum separation angle ($\alpha$ in Figure 16 (c)). Using the ratio between the number of bundles of two neighboring edges we divide the remaining angle. Given the allotted angle $\beta$ for a routing edge and the hub radius $r$ the length of the bundle base is equal to the chord length and hence given by $2r \sin(0.5\beta)$. From this it easy to find the bundle spacing.

4.2.2 Calculating biarcs

Inside a hub we connect the bundles at the bundle bases with a smooth curve called a hub segment. To draw our hub segments we use biarcs (see Figure 16 (b)). As the name suggests biarcs are defined by two circle arcs with a common endpoint. By using two circle arcs, possibly with different radii, we are able to correctly connect endpoints with the same tangent vector while maintaining the geometric continuity. A circle arc can be uniquely defined by the endpoints and the external point of the isosceles control triangle of the circle.

Given the vector representation of a startpoint ($p_s$) and endpoint ($p_e$), as well as their unit length tangent vectors $t_s$ and $t_e$ respectively, we will show how to find the control points of two arcs which connect them in $G_1$ continuity. We denote the vector representation of the three control points of the arcs by ($p_s, p_1, p_2$) and ($p_2, p_3, p_e$) (see Figure 17). Since $t_s$ and $t_e$ are of unit length we have:

\[
\begin{align*}
    p_1 &= p_s + \alpha t_s \\
    p_3 &= p_e - \alpha t_e \\
    p_2 &= (p_1 + p_3)/2.
\end{align*}
\]

Because the control triangle of a circle is an isosceles triangle we have $|p_1 - p_2| = |p_2 - p_3| = \alpha$, which is equivalent to:

\[
(p_1 - p_2) \cdot (p_1 - p_2) = \alpha^2. \tag{4.1}
\]

By substituting $p_1$ and $p_3$ into the equation for $p_2$ we can express the difference vector $p_1 - p_2$ as

\[
p_1 - p_2 = (p_s - p_e + \alpha(t_s + t_e))/2.
\]
By computing the dot product of Equation 4.1 and simplifying we get:

\[ v \cdot v + 2\alpha v \cdot (t_s + t_e) + 2\alpha^2 (t_s \cdot t_e - 1) = 0. \]  

(4.2)

where \( v = p_s + p_e \). In general Equation 4.2 always has a positive root which uniquely defines the control points of the biarcs. There are a few special cases which are of interest. If the endpoint tangents are parallel or if the vector sum of the tangents is perpendicular to segment \((p_s, p_e)\) the control points can be calculated without solving the quadratic equation. If the tangents are parallel and \( p_s + ct_s = p_e \), for some constant \( c \), the endpoints should be connected by a line segment.

### 4.3 Bundle ordering and crossing minimization

In a naive routing our bundles might cross more often than strictly necessary. 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 terminal vertices Pupyrev et al. [21] have shown that the optimal ordering can be calculated in linear time. In our setting bundles are also not allowed to pass through terminal vertices, however, since we are dealing with tree-structured bundles (see Figure 18) instead of simple paths the problem remains NP-complete. We will prove this using a reduction to the 1-sided crossing minimization problem.

The 1-sided crossing minimization problem considers a two layer graph where the vertices on one layer have a fixed position. The problem is to find a positioning of the vertices on the other layer which minimizes the number of crossings. The decision variant of this problem, whether there is a positioning of vertices which only causes \( c \) crossings is NP-complete [7]. We will give a polynomial time reduction of an arbitrary instance of the one layer crossing minimization problem to our edge bundling problem, hence proving the NP-completeness of (the decision variant of) bundle ordering.

Note that finding positions for vertices which minimize the number of crossings is equivalent to finding an ordering of these vertices. A formal statement of the one layer crossing minimization follows.

Given a two-layered (bipartite) graph \( G = (\{L_0, L_1\}, E) \) and an ordering \( x_0 \) of vertices on layer \( L_0 \), find an ordering \( x_1 \) of \( L_1 \) such that number of crossings is minimized.

**Definition 2** [Bundle ordering problem] Given a set of bundles along a routing graph edge and a hub of that edge with its other outgoing edges. Is there an ordering of bundles along the routing graph edge such that the number of crossings at the hub is at most \( c \)?

**Lemma 9** The bundle ordering problem is NP-Complete.

**Proof:** Consider an arbitrary instance of the 1-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 \( f_1 \ldots f_{|L_0|} \) and the vertices in \( L_1 \) by \( v_1 \ldots v_{|L_1|} \).

We construct an instance of the edge bundling problem for a hub \( h \) as illustrated in Figure 19. We add a routing graph edge \( e_i \), \( 1 \leq i \leq |L_0| \) on the left side for each of the vertices in \( L_0 \). We add a single routing graph edge on the right side, representing \( L_1 \).
We add $|L_1|$ bundles $b_1, \ldots, b_{|L_1|}$ to the right routing graph edge corresponding to the vertices in $L_1$. For each edge in $(f_i, v_j) \in E$ we add bundle $b_j$ to $e_i$.

The crossings in $h$ now have a one-to-one correspondence with the crossings in the 1-sided minimization problem. Hence solving the bundle ordering problem by ordering the bundles along the right edge provides an optimal ordering for the vertices in $L_1$. The reduction is polynomial which implies the NP-completeness of bundle ordering. □

4.3.1 A heuristic for bundle ordering

Since the bundle ordering problem is closely related to the 1-sided crossing minimization problem many heuristics for the former can be adapted to the latter. In our implementation we use the Barycenter heuristic [18] which calculates the “average” position for each bundle along the bundle base which determines the rank of the bundle. Figure 20 illustrates this for the normal 1-sided crossing minimization problem.

As illustrated in Figure 19 and detailed in the proof of Lemma 9 there is a one-to-one correspondence between the bundle ordering problem and the 1-sided crossing minimization. To determine the order of the bundles along a bundle base we simply treat the other bundle bases of the hub as the fixed layer. By changing the fixed layer we can determine an order for all bundle bases. After a few iterations of changing the fixed layer the change in the number of crossings is minimal and we stop the heuristic. This simple heuristic performs surprisingly well and usually more than halves the number of crossings compared to a random ordering.
Chapter 5

Experimental Evaluation

We implemented our algorithm in C++ and compiled it with gcc using standard optimization flags. The vectorized images in this thesis were generated by outputting Ipe\(^1\) formatted xml files and converting them to pdf. All tests were run on a machine with a 2.76 GHz Intel i5 quadcore CPU running Debian Linux. The algorithm was implemented to run on a single core.

5.1 Performance

The asymptotic running time is dominated by the calculation of the visibility graph and the greedy spanner sparsification. As mentioned in section 3, \(O(n^2 \log n)\) algorithms exist for both of these computations. The well-separated pair decomposition can be calculated in \(O(n \log n)\) time and hence it can also be used as a basis for bundling algorithms on very large graphs. Obstacle merging takes linear time in practice. In a worst case scenario where all obstacles are merged into a single obstacle the Convex hull and Voronoi algorithms give an \(O(n \log n)\) worst case running time assuming obstacles of constant size. The Barycenter ordering heuristic is linear in time and in practice very fast.

The running time in seconds of the different steps of the algorithm are shown in Table 1. The number of obstacle vertices is also mentioned in the table. For the airlines and migration graphs (Figure 25 and 22) we use simple squares around the vertices as obstacles. The small Tail graph ((see Figure 26)) also uses non-vertex obstacles. As obstacles we use larger versions of the labels, each with 6 vertices (corners and two extra to split the long side). Note that the edges in the routing graph include both the sparsified visibility edges and the internal obstacle edges.

| Graph   | \(|V|\) | \(|E|\) | \(|E_{routing}|\) | bundling | Routing | Ordering | Overall |
|---------|--------|--------|-------------------|----------|---------|----------|---------|
| Tail    | 8 (90) | 28     | 210               | 0.01     | 0.03    | 0.00     | 0.04    |
| Airlines| 235 (940) | 2101   | 2206              | 0.03     | 2.78    | 0.03     | 2.84    |
| Migration| 1715 (6860) | 9780   | 16022             | 0.99     | 161.50  | 0.31     | 162.80  |

Table 1: Statistics and performance in seconds on various graphs.

The Barycenter heuristic performs surprisingly well for such a fast algorithm. It usually more than halves the number of crossings when compared to a random ordering. Table 2 shows the number of crossings on different graphs.

\(^1\)The Ipe extensible drawing editor: http://ipe7.sourceforge.net/
Table 2: A comparison of the number of crossings.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Random ordering</th>
<th>Barycenter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tail</td>
<td>141</td>
<td>48</td>
</tr>
<tr>
<td>Airlines</td>
<td>35264</td>
<td>17937</td>
</tr>
<tr>
<td>Migration</td>
<td>145393</td>
<td>66767</td>
</tr>
</tbody>
</table>

5.2 Parameters

The two most important parameters for our algorithm are the separation constant $s$ and the spanner dilation $t$. A high value for $s$ means that bundled edges are more compatible, but also that the number of bundles increases. High values of $s$ are hence more suitable for smaller graphs, whereas for larger graphs a lower value of $s$ should be used to decrease the number of bundles and hence improve the clarity of the visualization.

We illustrate the results of our algorithm using graphs of different sizes. For the small tail graph (see Figure 26) we used $s = \infty$ which effectively disables bundling. For the bigger Airlines graph we used $s = 1.5$ (see Figure 22 (e) and Figure 23). Our largest graph is the US migration graph, for which we use $s = 0.5$ (see Figure 25). The effect of different values for $s$ on the number of bundles in the airlines graph and on the compatibility measures is detailed in Table 3.

| $s$ | $|B|$ | Angle (rad) | $|e|$ | $|\langle p_m, q_m \rangle|$ | $\frac{|e| + |f|}{2}$ | $|\langle q'_0, q'_1 \rangle|$ |
|-----|-----|------------|-----|-----------------|-----------------|-----------------|
| 0.5 | 266 | $\pi/2$    | 1/5 | 2               | -               | -               |
| 1.0 | 339 | $\pi/2$    | 1/3 | 1               | -               | -               |
| 1.5 | 406 | $\approx 1.18$ | 3/7 | 2/3             | 2/3             | 2/3             |
| 2.0 | 485 | $\approx 0.93$ | 1/2 | 1/2             | 1/2             | 1/2             |
| 4.0 | 691 | $\approx 0.49$ | 2/3 | 1/4             | 1/4             | 1/4             |
| $\infty$ | 2101 | 0 | 1 | 0 | 0 |

Table 3: By varying $s$ we can influence the number of bundles in the airlines graph and the compatibility of edges in the same bundle (see Section 2).

The spanner dilation $t$ influences the routing graph. Using a higher dilation means that bigger detours are allowed and hence will result in a sparser routing graph. We used a dilation of $t = 2$, meaning that any detour caused by the sparsification is at most twice as long as the shortest path on the original graph. See Figure 21 for two examples of a routing using different values for $t$.

![Figure 21](image.png)

(a) $t = 1.5$  
(b) $t = 4$

Figure 21: A routing with different values for $t$. 

22
5.3 Comparison with other techniques

Both the force-directed edge bundling (FDEB) of Holten and Van Wijk [16] (see Figure 22 (b)) and the geometry based edge bundling (GBEB) of Cui et al. [5] (see Figure 22 (c)) do not consider obstacles. Hence bundles can overlap vertices and it becomes nearly impossible to determine which edges and vertices are connected to a bundle. In particular GBEB makes some rather arbitrary routing choices and as a result the essentially empty middle of the US seems to be crisscrossed with airlines activity. Both methods do convey some rather high-level structure of the network, but we would like to argue that our approach gives a much cleaner and complete visual representation.
Figure 23: Large version of Figure 22 (e): Our algorithm on the US airline graph (235 vertices, 2102 edges).
The best algorithm to compare our contribution with is the Ordered bundles algorithm by Pupyrev et al. [21]. Similar to our algorithm it uses a routing graph and uses vertex obstacles to prevent bundles from overlapping vertices. The Ordered bundles algorithm draws every edge as a separate path and performs no bundling to allow the user to trace individual edges. This quickly becomes infeasible if the number of edges is large. In their rendering of the airlines graph (see Figure 22 (d)) Puyrev et al. used a sparser version of the input containing only 1297 edges.

We use a hybrid approach where each bundle is drawn individually. This drawing style is more explicit than the common solution of drawing all bundles on top of each other, it allows the user to follow a bundle comparatively easily and is also compact enough for large graphs. Another point at which the algorithms differ is the routing graph. The Ordered bundles algorithm uses a two phase approach of first constructing an initial routing graph and later optimizing this graph when an initial routing is found. Their initial routing graph is based on the sparse visibility graph used in [9] which is essentially a cone based spanner of the visibility graph. While very fast (\(O(n \log n)\)) the resulting spanner is much denser than the greedy spanner which we use. Since it is a cone based spanner it has a degree constraint on the vertices but it does not have the angle constraint and weight constraints which the greedy spanner has. Seeing as our algorithm targets medium sized graphs the near quadratic running time of our routing graph construction is not problematic.

To overcome the problems of the low quality initial routing graph, the Ordered bundles algorithm uses an extensive optimization step. All edges of the routing graph which are not used in the routing are removed and the remaining vertices are moved to better accommodate the routing. This is done in an iterative manner where vertices are moved away from obstacles which are too close to the vertex to draw the bundles properly. An important part of this optimization is keeping the routing cost low. By iteratively adjusting the position of the vertices the heuristic tries to decrease the routing cost and find suitable positions for the vertices which leave enough space to draw the bundles. Using an R-tree and obstacles of constant complexity the asymptotic running time of the optimization algorithm can be reduced to \(O(n \log n)\). However, experiments show that in practice the optimization step is still the most computationally expensive step of the algorithm. On larger graphs the step takes almost 90% of the total computation time.

By using the greedy spanner after eliminating most problematic vertices by merging obstacles our routing graph is of higher quality than the initial routing graph of the Ordered bundles algorithm. A direct result of this is that we do not require such an extensive optimization phase. Another benefit is the easier construction and a clearer relation between the initial input and the resulting graph. While asymptotically more expensive the low constant factors in our algorithm allow for similar running time on medium sized graphs.

Figure 24 shows various details from the large US airlines graph. Our routing shows clear connections even to vertices which are close to each other. By using a bundling with provable properties and a structured routing graph with bounded detours our algorithm can cleanly display much bigger graphs while maintaining almost all underlying information.
Figure 25: Our algorithm on the migration graph (1715 vertices, 9780 edges).
Figure 26: The Ordered bundles algorithm (a) and our algorithm (b) on the Tail graph from [21].

Figure 26 shows our algorithm on a small graph with additional obstacles. The graph shows a part of a software model. The components which cooperate are connected with a line (single line in Figure 26 (a) and bold black line in Figure 26 (b)). The routed edges are drawn as double lines in Figure 26 (a) whereas we draw them in blue in Figure 26 (b).

One factor for improvement in small graphs where edges can be routed separately is the bundle ordering. We improved the ordering in the small graph of Figure 26 by using a second heuristic which greedily tries to move single edges. The Ordered bundles algorithm calculates an optimal ordering which is possible when routing only simple paths. While our algorithm was not developed with small graphs in mind it still performs similar to the Ordered bundles algorithm.
We presented an edge bundling algorithm which defines its bundles based on a well-separated pair decomposition and routes bundles individually on a greedy sparsification of the visibility graph. We have proven that the bundles induced by the decomposition consist of compatible edges according to the measures introduced by Holten and Van Wijk [16]. By preprocessing our obstacles our routing graph shows clear connections, even to vertices which are close to each other. The high quality of the greedy sparsification gives a clean and structured look to our bundling, resulting a clean and complete visual representation of the underlying data.

**Future work.** Several interesting questions remain as a topic of future work. The well-separated pair decomposition gives us provable bounds on the compatibility of edges in a pair. By definition the decomposition is solely based on the point locations. In dense graphs one might argue that a decomposition which takes the edges of the underlying graph into account will give similar results. If however the graph is less dense and has clear flows, a lot of potentially useful information is ignored. By modifying the construction of the split tree one might steer the resulting well-separated pair decomposition to give more useful pairs.

Since our algorithm separates the bundling and the routing phase it allows us to improve them individually. While the well-separated pair decomposition has many good properties, it might be interesting to look at other decompositions of the data. A bundle appears to imply a complete bipartite graph. However, a bundle induced by a well-separated pair may not be complete. Hence a certain amount of ambiguity is introduced by the bundling. Different decomposition methods which take the edges into account and not just the points could be used to produce a set of bundles which does not have this problem.

Another point of improvement is the obstacle merging step. We use a joint convex hull to merge obstacles which are too close to each other. If the clearance value is high many obstacles may be merged together while some might be properly separated. By using a geodesic convex hull and using (larger versions of) obstacles, which are close to those which are supposed to be merged, as constraining polygons we can prevent this unnecessary merging while retaining a proper clearance around merged obstacles.

We decided to use the closest pair heuristic to determine merge points because it gave natural looking bundles which tend to preserve the general direction of the original edges when routed. There are many other strategies one can consider which give different types of bundles. One can for example use the point closest to the centroid of the set. Another interesting option is the use of Steiner points as merge points. Although this may allow for a nicer bundling it also introduces several new problems such as connecting the terminal vertices to the Steiner points and those Steiner points back to the routing graph vertices without creating too much clutter.

Currently we use colors to indicate the amount of edges in a bundle. Another approach is to actually draw bundles with more edges thicker. To use the limited drawing space efficiently one could use logarithmic or similar scaling of the bundle thickness.
Finally, the current algorithm has virtually no post-processing. Some extra bundle straightening or similar optimizations may be used to further increase the quality of the resulting bundling.


