

## Drawing with Fat Edges\*

Christian A. Duncan

*College of Engineering and Science, Louisiana Tech University  
Ruston, LA 71272, USA  
christian.duncan@acm.org*

Alon Efrat, Stephen Kobourov

*Department of Computer Science, University of Arizona  
Tucson, AZ 85721-0077, USA  
{alon, kobourov}@cs.arizona.edu*

Carola Wenk

*Department of Computer Science, University of Texas at San Antonio  
San Antonio, TX 78249-0667, USA  
carola@cs.utsa.edu*

Received (received date)

Revised (revised date)

Communicated by Editor's name

### ABSTRACT

Traditionally, graph drawing algorithms represent vertices as circles and edges as curves connecting the vertices. We introduce the problem of drawing with “fat” edges, i.e., with edges of variable thickness. The thickness of an edge is often used as a visualization cue, to indicate importance, or to convey some additional information. We present a model for drawing with fat edges and a corresponding polynomial time algorithm that uses the model.

We focus on a restricted class of graphs that occur in VLSI wire routing and show how to extend the algorithm to general planar graphs. We show how to convert an arbitrary wire routing into a homotopically equivalent routing that maximizes the distance between any two wires. Among such, we obtain the routing with minimum total wire length. A homotopically equivalent routing that maximizes the distance between any two wires yields a graph drawing which maximizes edge thickness. Finally, our algorithm also allows for different edge weights, that is, the requirement for unit wire thickness can be removed.

*Keywords:* Graph drawing, fat edges, VLSI wire routing, continuous homotopic routing

## 1. Introduction

In the area of graph drawing many algorithms have been developed for the classic problem of visualizing graphs in 2D and 3D. If the underlying graph is weighted, the edge weight information is typically displayed as a label near the edge. It seems

---

\*An extended abstract of this paper appeared in [5] and a related video contribution in [7].

natural to assign to the edges a width (or thickness) proportional to their weights. The problem we are interested in is the following: Given a plane (not necessarily straight-line) drawing of an edge-weighted graph, re-draw the graph so that homotopy is preserved, edges are as thick as possible with thickness proportional to their weights, and there are no edge crossings; see Figure 1. Surprisingly, there does not seem to be any previous work on drawing graphs with edges of varying thickness.

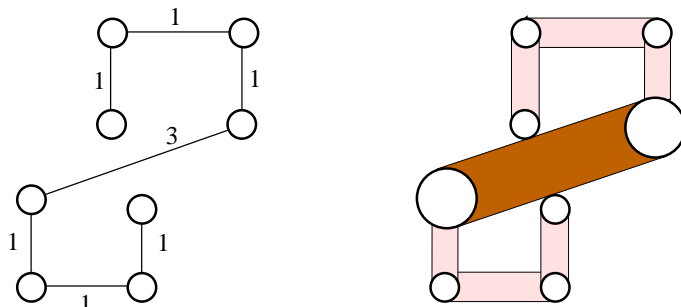


Fig. 1. An illustration of the fat edges problem. Given the plane drawing of a graph with edge weights (on the left), find the drawing with edges as thick as possible with thickness proportional to their weights (on the right).

Some related work has been done in addressing a classic VLSI problem, the continuous homotopic routing (CHR) problem [3, 11]. In the CHR problem an initial wiring layout for a set of wires with fixed terminals and fixed obstacles, and some constant  $\varepsilon > 0$  are given, and the task is to continuously transform the initial wire layout to a new wiring in which the wire separation is at least  $\varepsilon$ , if that is possible. If the initial wiring, which specifies a homotopy class, is not given or the terminals are not fixed, the problem is NP-hard [12, 16, 17]. In this setting the graph is given a fixed planar embedding and the maximum degree is 1 since wires do not share terminals. It is easy to see that the CHR problem can be rephrased as the following graph drawing problem: Given an embedding, does there exist a planar drawing that respects the embedding and in which all the wires are drawn with thickness  $\varepsilon$ ?

In this paper we address a more general optimization problem which we call the Fat Edge Drawing (FED) problem: given a planar weighted graph  $G$  with maximum degree 1 and an embedding for  $G$ , find a planar drawing such that all the edges are drawn as thick as possible and proportional to the corresponding edge weights. The FED problem is a generalization of the CHR problem since it allows for edges of different weights and solves the maximization problem, rather than the decision problem. The General Fat Edge Drawing Problem (GFED) is the FED problem without the maximum degree condition. We present an algorithm for the FED problem, which easily generalizes to an algorithm for the GFED problem.

### 1.1. Previous Work

Cole and Siegel [3] and Leiserson and Maley [11] show, in early work on continuous homotopic routing, that in the  $L_\infty$  norm a solution can be found in  $O(k^3 \log n)$

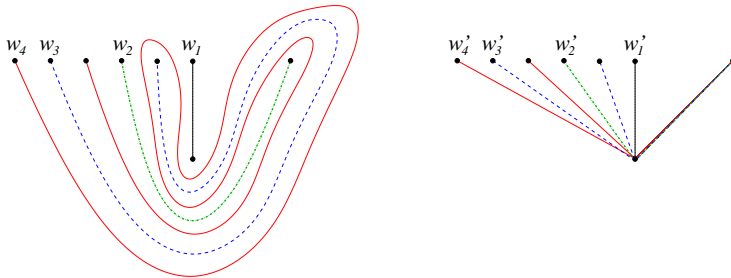


Fig. 2. An example with exponential complexity:  $k$  is  $\Omega(2^n)$ : On the left is the initial set  $P$ , and on the right is  $P'$  after the shortest paths have been computed. The number of edge segments in the shortest paths  $w'_1, w'_2, w'_3, w'_4$  is 1, 2, 4, 8. In general, path  $w'_i$  has  $2^{i-1}$  edge segments. Note that on the right many edge segments look coincident, but are actually adjacent and parallel.

time and  $O(k^3)$  space, where  $n$  is the number of wires and  $k$  is the maximum of the input and output complexities of the wiring. Note that the complexity of the wiring is measured in the number of line or curve segments necessary to describe the wiring. Maley [13] shows how to extend the distance metric to arbitrary polygonal distance functions (as well as the Euclidean distance) and presents a  $O(k^4 \log n)$  time and  $O(k^4)$  space algorithm. Note that  $k$  can be arbitrarily larger than  $n$ : for example a wire could wrap around a vertex an arbitrary number of times. Even after shortest paths have been computed for each wire,  $k$  can be  $\Omega(2^n)$ ; see Figure 2. Gao *et al.* [8] present a  $O(kn^2 \log(kn))$  time and  $O(kn^2)$  space algorithm for solving the decision problem when the wires have unit thickness.

Similar work has been done for a special class of grid graphs: finite subgraphs of the planar rectangular grid. The first algorithms for such restricted versions of the problem are presented by Mehlhorn and Kaufmann [10, 9], and by Schrijver [18, 19]. Work in this area is related to the river routing problem in VLSI chips [3, 4, 14, 16].

Pach and Wenger [15] consider a related problem, laying out a planar graph at predefined locations in the plane. Given a planar graph on  $n$  vertices and a point set with  $n$  points, they want to draw the graph subject to the condition that each vertex  $v_i$  is mapped to a point  $p_i$ . This can be done using  $O(n)$  bends per edge using the algorithm of Pach and Wenger [15], who also show this bound to be tight in the worst case even if the graph is merely a perfect matching. For the problem of determining minimum bend wire routings, this implies that  $k$  is  $\theta(n^2)$ . A recent result is that of Barequet *et al.* [1] who show how to draw a maximally planar graph with straight-line fat edges and fat vertices, both of given size, on a compact grid.

## 1.2. Our Results

We show how to solve the FED problem in  $O(n^3 + k)$  time and  $O(n + k)$  space, where  $n$  is the number of paths and  $k$  is the maximum of the initial and final complexities of all the paths (assuming, as is common, that  $k$  can be stored in constant space). Our algorithm assumes that the embedding of the graph is given as a set of homotopic shortest paths. If this is not the case then homotopic shortest paths can be computed in  $O(n\sqrt{n} + k')$  time and  $O(n + k')$  space as a preprocessing step

using the algorithm of [6], where  $k'$  is the maximum of the input and output complexities of the homotopic shortest paths algorithm. See Section 2 for the definition of homotopic shortest paths. From [6], we also know that the complexity of the shortest homotopic paths is always  $O(nk)$ , where  $k$  is the complexity of any equivalent set of homotopic paths. If we let  $k_{in}$  and  $k_{out}$  represent the input and output complexities of our algorithm and observe that both are homotopic equivalents of the intermediate shortest paths, then we see that  $k' \in O(k_{in} + \min(k_{in}, k_{out})n)$ . Therefore, our algorithm, when not given the shortest homotopic path, takes time  $O(n^3 + k + k')$  and requires  $O(n + k + k')$  space, for  $k$  and  $k'$  described above. For simplicity throughout the rest of this paper, unless otherwise noted, we shall assume that the input paths are given as a set of shortest homotopic paths.

We also show how to extend the FED algorithm to an algorithm for the GFED problem with the same time and space bounds. Our FED algorithm solves a more general problem than the CHR problem and is an improvement in both space and time complexity over the best known algorithm for the CHR problem. We refer the reader to the video contribution [7] that demonstrates our algorithm.

## 2. Continuous Homotopic Routing

A *path*  $p$  is a continuous injective curve  $p : [0, 1] \rightarrow \mathbb{R}^2$ . A set  $P = \{p_1, p_2, \dots, p_n\}$  of paths is called *collision-free* if the paths pairwise do not intersect. Let  $T = T(P) = \{p_i(0), p_i(1) \mid 1 \leq i \leq n\}$  be the set of *terminals* of  $P$ , and let  $V = T \cup O$  be the set of *vertices*, where  $O$  is a set of  $O(n)$  point *obstacles* disjoint from  $P$ .

Let  $p, q : [0, 1] \rightarrow \mathbb{R}^2$  be two paths. We say that  $p$  and  $q$  are *homotopic with respect to a set*  $V \subseteq \mathbb{R}^2$  if  $p(0) = q(0)$ ,  $p(1) = q(1)$ , and there exists a continuous function  $h : [0, 1] \times [0, 1] \rightarrow \mathbb{R}^2$  with the following three properties:

- (i)  $h(0, t) = p(t)$  and  $h(1, t) = q(t)$ , for all  $0 \leq t \leq 1$ ,
- (ii)  $h(\lambda, 0) = p(0) = q(0)$  and  $h(\lambda, 1) = p(1) = q(1)$  for all  $0 \leq \lambda \leq 1$ , and
- (iii)  $h(\lambda, t) \notin V$  for all  $0 \leq \lambda \leq 1, 0 < t < 1$ .

In other words, we continuously deform the path from  $p$  to  $q$ , without changing the terminals and without having the intermediate paths intersect any of the vertices. We call a set of paths  $P' = \{p'_1, p'_2, \dots, p'_n\}$  a *homotopic shift of (the collision-free set)  $P = \{p_1, p_2, \dots, p_n\}$  with respect to  $V$* , if  $p'_i$  is homotopic to  $p_i$  for all  $1 \leq i \leq n$ , and  $P'$  is collision-free. See [2], for related work on sets of homotopic paths.

For each path  $p$ , we also associate a *weight*  $\omega_p > 0$ . We would like to draw all paths with thicknesses proportional to their weight. We will define the *weighted distance* between two paths  $p, q$  such that when  $p, q$  have weighted distance  $d^*$ , then  $d^*$  is the largest  $d$  such that  $p$  can be drawn with thickness  $\omega_p d$  and  $q$  can be drawn with thickness  $\omega_q d$ , and the two thickened paths are interior-disjoint. We define the *weighted distance* between two paths  $p$  and  $q$  as the minimum weighted distance between all  $u \in p$  and  $v \in q$ , where the weighted distance between  $u$  and  $v$  is the Euclidean distance divided by  $\omega_p + \omega_q$ . Notice that if both weights are 1, the

weighted distance is half the Euclidean distance between the paths. The (*weighted*) *separation*,  $s(P)$ , of a set of paths  $P$  is defined to be the minimum weighted distance between any two paths  $p_i, p_j \in P$ .

We address the following **Fat Edge Drawing (FED)** problem:

Given a (shortest path) collision-free set  $P = \{p_1, p_2, \dots, p_n\}$  of weighted paths and a vertex set  $V$ , find a homotopic shift  $P' = \{p'_1, p'_2, \dots, p'_n\}$  of  $P$  with respect to  $V$  with maximum (weighted) separation, i.e.,  $s(P') \geq s(Q)$  for all homotopic shifts  $Q$  of  $P$ .

Note that if a collision-free set  $P$  of paths has separation  $\sigma$ , then we can draw each path  $p_i$  with thickness  $\omega_i\sigma$ , i.e., replace  $p_i$  with

$$q_i = p_i \oplus B(\omega_i\sigma) = \{a + b \mid a \in p_i, b \in B(\omega_i\sigma)\}$$

and guarantee that all pairs  $q_i, q_j$  (for  $i \neq j$ ) are interior disjoint. Here  $B(\omega_i\sigma)$  is a disk of radius  $\omega_i\sigma$  centered at the origin, and the  $\oplus$  operation is the Minkowski sum operation. Hence, the maximum separation for a set of paths  $P$  corresponds to the maximum thickness with which the paths in  $P$  can be drawn without intersections. Therefore, computing a homotopic shift of  $P$  with respect to  $V$  with maximum separation corresponds to drawing a planar graph, with vertex degrees at most 1, with edges as thick as possible.

We call  $q_i$  a *fat edge* of *width* (or *thickness*)  $\omega_i\sigma$ . We call  $p_i$  the *backbone* of  $q_i$ , and define the *length* of  $q_i$  to be the length of its backbone  $p_i$ . We call any positioning of the backbones a *routing*. The output of our algorithm will be the routing of the backbones of the set of fat edges with maximum separation.

### 3. Algorithm Overview

We are now ready to outline the general technique for solving the FED problem. We begin with the initial set of paths  $P$  with vertex set  $V$ , which we assume is a shortest homotopic shift. We consider  $P = P(0)$  as a set of fat edges, each with zero width.

From  $P(0)$  we compute a homotopic shift with maximum separation by applying the following kinetic approach: We let the fat edges simultaneously grow in thickness over time, at a speed proportional to the individual weight of each path. Throughout this growth process, the sum of the lengths of all paths (backbones) remains as short as possible, and the homotopy among paths is preserved. Let  $t \geq 0$  be the time parameter and  $P(t)$  be the set of *thickened paths* at time  $t$ , i.e., path  $p \in P(t)$  has thickness  $\omega_p t$ . The process terminates when it becomes impossible to increase their widths while keeping them interior disjoint. We next study the structure that these fat edges form at each arbitrary time  $t$ .

At time  $t$  each terminal vertex defines a disk of radius  $\omega_i t$ ; see Figure 3. A fat edge  $q_i(t)$  which touches this vertex must share a portion of its boundary with this disk (since its backbone is shortest), so a portion of  $\partial q_i(t)$  is a circular arc, forcing a portion of its backbone  $p_i(t)$  to form a circular arc as well. In turn, this forces

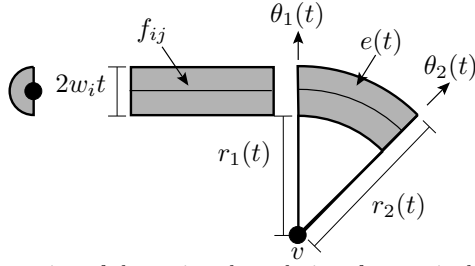


Fig. 3. An illustration of the various boundaries of a terminal elbow, straight segment, and non-terminal elbow. The segments are separated only for illustration.

another portion of  $q_i = q_i(t)$  to form a circular arc; see Figure 3. Altogether a region of  $q_i(t)$  which we call an *elbow*<sup>a</sup>, is a portion of an annulus between two rays emerging from the center of the annulus. Formally, an *elbow*  $e(t)$  associated with a vertex  $v \in V$ , the radii  $r_1(t), r_2(t) > 0$ , and angles  $\theta_1, \theta_2 \in [0, 2\pi)$  is defined by

$$e(t) = \{a \in \mathbb{R}^2 \mid r_1(t) \leq \|a - v\| \leq r_2(t) \text{ and the orientation of } \overrightarrow{av} \in [\theta_1(t), \theta_2(t)]\}.$$

Here the *orientation* of a directed segment  $\overrightarrow{av}$  is the counterclockwise angle between  $\overrightarrow{av}$  and a horizontal ray emerging from  $a$  in the positive direction. We call the portion of the elbow with smaller and larger radii the *inner* and *outer* circular arcs of the elbow. Note that if there is another fat edge that touches the outer circular arc, then using analogous arguments, its boundary must form a circular arc, defining another elbow, etc. As we leave the center of the vertex defining this elbow, we meet the elbows in a natural order. Each elbow “stores” the neighboring elbow, and the neighboring parts of the fat edges, to be described next. The vertex *associated* with the elbow is the one defining the elbow. In addition, we refer to the elbows at the endpoints of a fat edge as *terminal elbows*. For simplicity in our description, we also associate a terminal elbow with weight zero to each obstacle vertex, meaning these elbows do not grow but can otherwise be treated the same as other terminal elbows.

As  $t$  increases, only circular arcs force the backbone to diverge from a straight segment, since the backbone is always a shortest path. Hence the portions that are not circular arcs must be straight segments. Every straight-line segment  $f_{ij}$  of  $p_i$  corresponds to a rectangular portion of the fat edge  $q_i(t)$ , called a *straight*. This rectangle has one side (its *width*) of length  $2w_i t$  and the other side of length  $|f_{ij}|$ . Here  $|f_{ij}|$  is the length of the straight-line segment  $f_{ij}$ . Formally, a straight is the set of all the points in the plane of distance less than or equal to  $w_i t$  from  $f_{ij}$  that lie in the infinite strip of width  $|f_{ij}|$  orthogonal to  $f_{ij}$ . It is easy to note that  $q_i$  consists of an alternating sequence of straights and elbows.

Every straight  $s$  maintains pointers to the preceding and succeeding elbows of its own fat edge and to the neighboring straights that share part of the boundary with  $s$ . For simplicity, we refer to the *ends* of  $s$  as touching the elbows and to the *sides* of  $s$  as touching neighboring straights. We say that a straight is associated

<sup>a</sup>We borrow the names for these curves from plumbing jargon.

with vertices  $u, v$  if it connects two elbows, one associated with vertex  $u$  and the other with vertex  $v$ .

Note that under the general position assumption, a straight shares an edge with at most one other straight on each of its sides. Similarly, an elbow must have a unique elbow (possibly a terminal elbow) on its inner side, but it might have several elbows on its outer side.

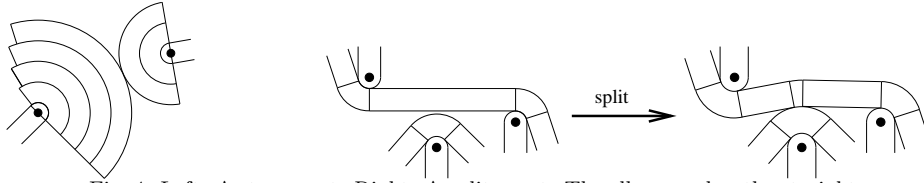


Fig. 4. Left: A stop event. Right: A split event. The elbow pushes the straight up such that it gets replaced by a straight-elbow-straight sequence.

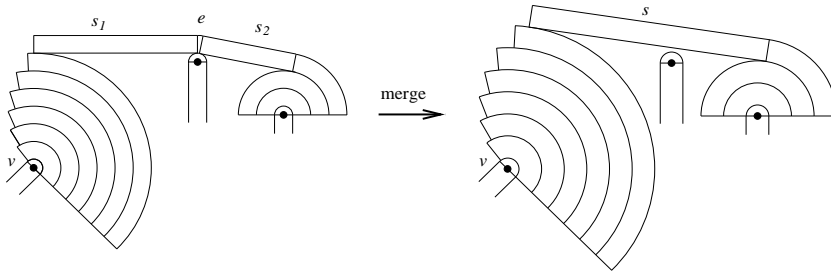


Fig. 5. A merge event. Vertex  $v$  pushes the left straight  $s_1$  up such that the elbow  $e$  in the middle disappears. The straight-elbow-straight sequence  $s_1 - e - s_2$  gets replaced by a single straight  $s$ .

During the “thickening” process as  $t$  increases from 0, various events occur that affect the combinatorial nature of the paths. In particular, the following events are of interest:

- **Split event:** a straight touches an elbow that it does not connect,
- **Merge event:** an elbow collapses into a line segment:  $\theta_1 = \theta_2$ ,
- **Stop event:** an elbow touches another elbow with a different associated vertex.

See Figures 4 and 5 for an illustration of the events.

In general, our thickening process does not go on indefinitely. In fact, as the following lemma explains, the process must terminate at the first stop event. For simplicity, when occurring at the same time frame, we perform all split and merge events *before* any stop events. We refer to this time  $t^*$  as the *stop time*.

**Lemma 1** *Let  $t^* > 0$  be the first time that a stop event occurs. All thickened paths of  $P(t)$  are interior disjoint, if and only if  $t \leq t^*$ .*

**Proof.** Let us examine the case where  $t > t^*$ . Since  $t^*$  is a stop event, let  $u$  and  $v$  be the two vertices associated with the two colliding elbows,  $e_u$  and  $e_v$ . Let  $\ell_u$  be

the sequence of elbows interior to  $e_u$ , and let  $\ell_v$  be the sequence of elbows interior to  $e_v$ . That is, start with  $e_u$  and repeatedly look at the unique elbow inside of it, until reaching the terminal elbow associated with  $u$ . Let  $S_u$  be the sequence of path labels containing each elbow in  $\ell_u$ , and let  $S_v$  be the sequence of path labels containing each elbow in  $\ell_v$ . Note that it is possible that a path label appears in  $S_u$  multiple times, due to the path looping around a vertex. Let  $S = S_u \cup S_v$  be the merge of the two sequences. Let  $w = \sum_{p \in S} w_p$  be the sum of the weights of each path (segment) in the sequence. Because of the radii of each elbow and the fact that  $e_u$  touches  $e_v$ , we know that  $wt^* = \|u - v\|$ . Since the paths are homotopic, for  $t > t^*$ , each of the path segments in  $S$  must still pass between  $u$  and  $v$ . This means that the width of the elbows associated with  $S$  must be  $wt > wt^* > \|u - v\|$ . Therefore, there must be at least two paths that are not interior disjoint.

Let us now prove the case for  $t \leq t^*$  by showing that an intersection among the thickened paths implies that  $t > t^*$ . Let  $t$  be the smallest  $t > 0$  such that the thickened path segments are not interior disjoint. Let  $s_1$  and  $s_2$  be two (thickened) path segments that intersect at time  $t$ . Let  $t'$  be a time frame infinitesimally smaller than  $t$  where  $s_1$  and  $s_2$  are tangent, and after all split and merge events have taken place. There are three possibilities for the two segments: both are straight segments, both are elbow segments, or one of each. If the two segments are both elbows, they must be associated with different vertices in order to intersect at time  $t$ . We therefore have a stop event at time  $t'$ , implying that  $t > t' \geq t^*$ . If  $s_1$  is an elbow and  $s_2$  is a straight segment,  $s_1$  must be tangent to an elbow  $e_2$  at one of the ends of  $s_2$ , for a split event would have separated  $s_2$  at the point of contact. Consequently, at time  $t$ , the two elbows  $s_1$  and  $e_2$  intersect, which again implies that  $t > t^*$ . If the two segments are both straight segments, then without loss of generality at least one end of  $s_1$ , which is an elbow  $e_1$ , must also be tangential to  $s_2$ . Therefore at time  $t$  we have the elbow  $e_1$  intersecting the straight segment  $s_2$ , and again  $t > t^*$ .  $\square$

#### 4. Compact Routing Structure

To represent the straights and elbows and maintain the state of the growth process efficiently, we introduce a compact routing data structure. From the beginning to the end of the growth process, the algorithm that maintains this data structure uses  $O(n^3)$  time and  $O(n + k)$  space, where  $k$  is the maximum of the input and the output complexities of the paths. In order to achieve this time bound independent of  $k$ , which can be arbitrarily large, we group straights and elbows together. Note that in  $P$  many of the paths may travel in parallel. We take advantage of this and group such parallel paths into *bundles* and maintain *straight bundles* and *elbow bundles* instead of single straights and elbows in our compact routing structure.

In Theorem 2 we prove that the number of bundles stored in this data structure at any fixed time is only  $O(n)$ , and in Theorem 3 we prove that the space required is  $O(n + k)$ . After the growth process stops we reconstruct the set of maximally separated paths consisting of straight line segments and circular arcs by unzipping the bundles.

#### 4.1. Bundling Elbows and Straights

**Definition 1** Let  $u, v \in V$ .

- A **straight bundle** associated with  $u$  and  $v$  is a maximal connected rectangular region formed by the union of straights that are associated with  $u$  and  $v$ .
- An **elbow bundle** associated with  $v$  is a maximal connected region formed by the union of elbows associated with  $v$  and sharing the same straight bundles on both ends. A terminal elbow forms its own bundle which is not merged with other bundles.

For any vertex  $v$  there may be several elbow bundles; see Figure 6 for an illustration of bundles.

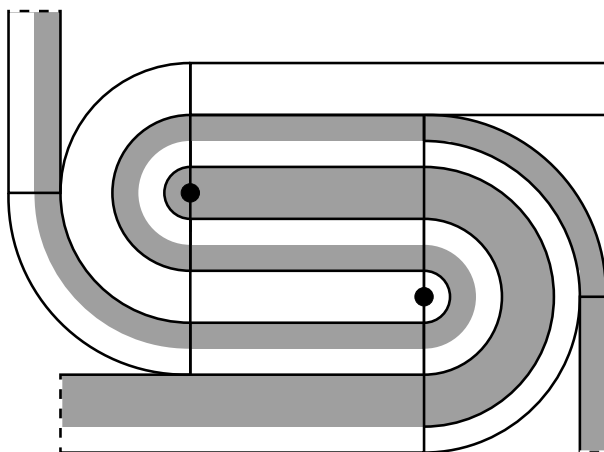


Fig. 6. An illustration of various possible bundles between paths passing (and originating) at two vertices. Bundle boundaries are drawn bold. The two terminal elbows are the half-disks at the two vertices. The left vertex has four elbow bundles associated with it, a total of six elbows. As the paths widths grow, the upper horizontal straight bundle will intersect an elbow region of the right vertex, forcing it to bend clockwise. This will cause this straight bundle to merge with the one below it. The same applies for the lower horizontal straight bundle.

**Lemma 2** For any two vertices  $u, v \in V$  there can be at most five straight bundles associated with  $u$  and  $v$ .

**Proof.** Without loss of generality assume that  $u$  and  $v$  lie on the  $x$ -axis. Then any non-terminal straight bundle associated with  $u$  and  $v$  can be of four possible types. It can have both  $u$  and  $v$  below, both  $u$  and  $v$  above,  $u$  below and  $v$  above, or  $u$  above and  $v$  below. There can be at most one straight bundle of either type, because two straight bundles of the same type would have to be adjacent, which is a contradiction of the maximality of bundles. Hence, there are at most four straight bundles associated with  $u$  and  $v$ . However, it is not possible to have both an above/below bundle and a below/above bundle since those two bundles would have

to intersect. Hence there are at most three such bundles. Observing that there are two additional bundles possible for the terminal vertices leads to the above result.  $\square$

#### 4.2. The Compact Routing Structure $\mathcal{S}$

We store  $P(t)$  in a compact routing structure  $\mathcal{S}(t)$  as follows:

- A straight bundle  $sb(t)$  stores:

*left/right* : references to the two elbow bundles at each end of  $sb(t)$  that are outermost among all these elbow bundles. Note these may not be on the same end of the bundle, consider an “S” shaped pattern. Here for the middle straight bundle, the left elbow would be the bottommost segment and the right elbow would be the uppermost.

*size* : number of segments associated with this bundle.

*weight* : sum of the weights of the segments in the bundle.

- An elbow bundle  $eb(t)$  stores:

*vertex* : reference to the associated vertex,

*left/right* : references to the two straight bundles that have  $eb(t)$  on one end (for terminal elbows,  $left = right$ ),

*inner* : reference to the elbow bundle just inside it (for terminal elbows,  $inner = \text{NIL}$ ),

*size* : number of segments associated with this bundle (for terminal elbows,  $size = 1$ ),

*weight* : sum of the weights of the segments in the bundle.

*layerW* : sum of the weights of each of the paths between  $eb(t)$  and  $v$ , not counting its own weight.

**Lemma 3** *Given a set of (homotopic shortest) paths  $P$  the compact routing structure  $\mathcal{S}$  can be initialized to represent  $P(0)$  in  $O(n + k)$  time and space, where  $k$  is the complexity of  $P$ .*

**Proof.** The algorithm of [6] provides, along with the shortest paths, the order among the edges that are associated with the same pair of vertices. Creating the structure is then just a matter of merging the straight segments between equal vertices and then creating the elbow bundles outward from each vertex. This will take constant work per segment merged or  $O(n + k)$  time.  $\square$

#### 4.3. Maintaining $\mathcal{S}$

The combinatorial structure of  $P(t)$  changes at split events and merge events. Since  $\mathcal{S}$  represents  $P(t)$  using bundles we perform split and merge operations (c.f. Figure 7) directly on the bundles.

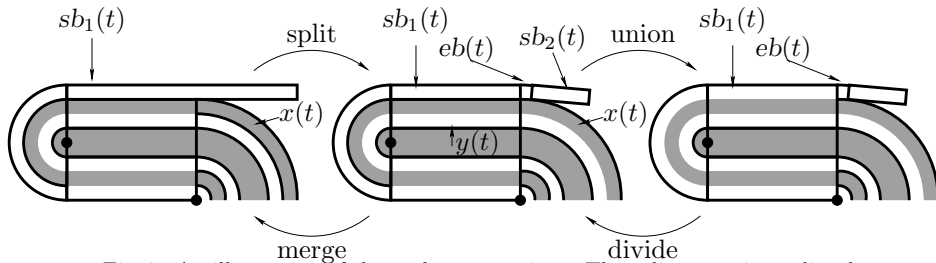


Fig. 7. An illustration of the update operations. The split operation splits the straight bundle  $sb(t)$  at the elbow bundle  $x(t)$  creating three bundles  $sb_1(t)$ ,  $eb(t)$ , and  $sb_2(t)$ . The union operation then combines  $sb_1(t)$  and  $y(t)$  into the merged  $sb_1(t)$  bundle. In addition, this union also triggers the union of the two leftmost elbow bundles. The divide and merge operations reverse this process.

**Definition 2** *During the thickening process, we perform the following update operations on  $\mathcal{S}$ ; see Figure 7:*

- **split**( $sb(t), x(t)$ ): *Splits a straight bundle  $sb(t)$  into three adjacent bundles  $sb_1(t)$ ,  $eb(t)$ , and  $sb_2(t)$ , where  $eb(t)$  is a degenerate elbow bundle that touches the elbow  $x(t)$ .*
- **merge**( $sb_1(t), eb(t), sb_2(t)$ ): *Merges the straight-elbow-straight bundle sequence  $sb_1(t), eb(t), sb_2(t)$  (where  $eb(t)$  is a degenerate elbow of zero length) into one straight bundle.*
- **union**( $x(t), y(t)$ ): *Merges two non-maximal straight (resp. elbow) bundles  $x(t)$  and  $y(t)$  that are associated with the same two vertices  $u$  and  $v$  and share a line segment (resp. circular arc) along their boundary into a single straight (resp. elbow) bundle  $z(t)$ .*
- **divide**( $sb(t), eb(t)$ ): *Splits a straight bundle  $sb(t)$  at the adjacent elbow bundle  $eb(t)$  into two interior-disjoint straight bundles  $sb_1(t)$  and  $sb_2(t)$  having the same associated vertices and that (temporarily) share a straight segment. This is used in preparation for removing the degenerate bundle  $eb(t)$  by merging  $sb_1(t)$  with the other straight bundle adjacent to  $eb(t)$ . The two divided straight bundles will then no longer be adjacent to the same vertices.*

The pseudocode in Figure 8 and Figure 9 describes the various operations in detail. For simplicity, we leave off the  $t$  term. In addition, we make general assumptions about left and right sides, namely that the “right” edge of an elbow bundle references the “left” edge of a straight bundle and vice versa. In fact, it may be either edge depending on the orientation of the various bundles. However, it is a constant time operation to determine which side was actually referenced.

**Lemma 4** *The update operations, split, merge, union, and divide, can be performed in  $O(n)$  time.*

**Proof.** A careful look at the code shows the total time taken per operation is exactly the number of bundles affected. The only loop updates are for the reference

```

split(sb, x)
  Postcondition: sb gets split into sb, eb, and sb2.
  1 sb2 ← copy of sb
  2 Create a new elbow bundle eb.
  3 sb.right ← eb, sb2.left ← eb
  4 eb.left ← sb, eb.right ← sb2, eb.inner ← x
  5 eb.size ← sb.size, eb.weight ← sb.weight
  6 eb.layerW ← x.layerW + x.weight
  ▷ Update the elbow bundles on the right to point to sb2
  7 ebRight ← sb2.right
  8 while ebRight.left = sb
  9   do ebRight.left ← sb2
  10   ebRight ← ebRight.inner
  ▷ Check if sb (resp. sb2) must unite with x's left (resp. right) straight bundle.
  11 if x.left.left.vertex = sb.left.vertex then union(sb, x.left)
  12 if x.right.right.vertex = sb2.right.vertex then union(sb2, x.right)

merge(sb1, eb, sb2)
  Postcondition: sb1 becomes the new single straight bundle
  1 if sb1.size > eb.size
  2   then divide(sb1, eb)
  3 if sb2.size > eb.size
  4   then divide(sb2, eb)
  5 sb1.right ← sb2.right
  ▷ Update the elbow bundles on the right to point to sb1.
  6 See split:7–10.
  7 Delete eb, sb2

```

Fig. 8. Pseudocode for the **split** and **merge** operations.

pointers from elbows to straights. In Theorem 2, we prove that this size is  $O(n)$ , completing the proof.  $\square$

Since the compact routing data structure does not maintain the path segments explicitly, we must recreate the paths by unzipping the bundles. We take each straight bundle and “tear” the top segment off, separating it from the rest of the bundle. We repeat this for every bundle until every straight bundle consists of exactly one segment. The definition of elbow bundles implies that every elbow bundle also has one segment. Consequently, we are left with only segments. The process of tearing destroys the weight information; however, this information is irrelevant in the unzipping process as we do not physically move the paths. After unzipping, we simply start at the terminals for each path and follow the data structure pointers to identify each segment’s (singleton bundle’s) associated path and hence thickness. The procedure **TEAR**, is derived from the **divide** function; see Figure 10.

Using the code from **UNZIP**, we arrive at the following lemma.

**Lemma 5** *The bundles of  $\mathcal{S}(t)$  can be uncompressed into straights and elbows representing  $P'(t)$  in time  $O(k_{out})$ , where  $k_{out}$  is the final complexity of all paths.*

**Proof.** From the code for **TEAR**, one can see that the procedure correctly removes one segment from the top of a straight bundle. Each of the elements of the compact routing data structure described in Section 4.2 is updated except for the weights. Each call to **TEAR** takes  $O(1)$  time and decreases a bundle’s size

```

union(x, y)
  Postcondition: x will become the united bundle
  1 x.size ← x.size + y.size, x.weight ← x.weight + y.weight
  2 if the bundles are straight bundles
  3   then if y is farther from x.left.vertex than x
  4     then x.left ← y.left
  5     if y is farther from x.right.vertex than x
  6       then x.right ← y.right
  7     ▷ Update the elbow bundles referencing y to reference x.
  8     See split:7–10.
  9     ▷ Check to see if two elbows (on the left) must merge
 10    for each adjacent pair (e1, e2) of elbow bundles to the left of x
 11    if e1.left = e2.left
 12      then union(e1, e2)
 13    Repeat for the right side
 14  else ▷ the bundles are elbow bundles
 15    if x is farther from x.vertex than y is
 16      then x.inner ← y.inner
 17      x.layerW ← y.layerW
 18    else y.left.right ← x
 19           y.right.left ← x
 20  Delete y

divide(sb, eb)
  Precondition: eb, the top elbow bundle of sb, is to be merged.
  Postcondition: sb2 is the new upper straight bundle separated from sb.
  1 if sb.right = eb
  2   then sb2 ← copy of sb
  3     sb.right ← eb.inner, eb.left ← sb2
  4     sb2.size ← eb.size, sb2.weight ← eb.weight
  5     sb.size ← sb.size – sb2.size, sb.weight ← sb.weight – sb2.weight
  6     ▷ Need to update references on left of straight.
  7     ▷ May also need to divide one elbow.
  8     ▷ Here we assume that sb is closer than sb2 to their left associated vertex.
  9     ▷ The code is similar in the other case.
 10    ebLeft ← sb2.left
 11    size ← ebLeft.size, weight ← ebLeft.weight
 12    while size < sb2.size
 13      do ebLeft.right ← sb2
 14          ebLeft ← ebLeft.inner
 15          size ← size + ebLeft.size, weight ← weight + ebLeft.weight
 16    if size > sb2.size ▷ The current elbow bundle must be split
 17      then ebNew ← copy of ebLeft
 18          ebLeft.inner ← ebNew
 19          ebNew.size ← size – sb2.size
 20          ebNew.weight ← weight – sb2.weight
 21          ebLeft.size ← ebLeft.size – ebNew.size
 22          ebLeft.weight ← ebLeft.weight – ebNew.weight
 23          ebLeft.right ← sb2
 24          ebLeft.layerW ← ebNew.layerW + ebNew.weight
 25    else ▷ sb.left = eb
 26      Repeat for this symmetrical case.

```

Fig. 9. Pseudocode for the union and divide operations.

```

TEAR( $b$ )
  Postcondition:  $\triangleright$  Remove the top segment from the current straight bundle
  1 if  $b.size = 1$ 
  2   then return
  3  $c \leftarrow$  a new straight bundle
  4  $c.size \leftarrow 1$ , decrement  $b.size$ 
  5  $\triangleright$  Remove the top segment from the left elbow bundle and attach to  $c$ 
  6  $eb \leftarrow b.left$ 
  7  $c.left \leftarrow eb$ ,  $eb.right \leftarrow c$ 
  8 if  $eb.size = 1$ 
  9   then  $b.left \leftarrow eb.inner$ 
 10  else  $ebInner \leftarrow$  copy of  $eb$ 
 11      $eb.size \leftarrow 1$ , decrement  $ebInner.size$ 
 12      $eb.inner \leftarrow ebInner$ 
 13      $b.left \leftarrow ebInner$ ,  $ebInner.right \leftarrow b$ 
 14  $\triangleright$  Remove the top segment from the right elbow bundle (same as previous)

UNZIP()
  1 while any straight bundle  $b$  has size greater than 1
  2   do TEAR( $b$ )
  3 for each path  $p$ 
  4   do Follow the left/right adjacency values from  $p$ 's start to end terminals
  5     filling in the path's label and weight into the singleton bundles along the way.
  6     Each segment has just two adjacent segments so the traversal is trivial.

```

Fig. 10. Pseudocode for the TEAR and UNZIP operations.

by one creating at most three new singleton bundle segments, each representing a segment in the final output. The process terminates when there are no more non-singleton straight bundles. Since each elbow bundle contains segments that connect to the same straight bundles, each elbow bundle can have at most one segment as well. Consequently, every bundle in the now uncompressed structure is a singleton having exactly one associated segment, a size of 1. The final loop of UNZIP simply traces a path of segments filling in the associated path for each segment (singleton bundle), which is done exactly once per segment. The total running time is therefore proportional to the number of new singleton bundles and so is  $O(k_{out})$ .  $\square$

#### 4.4. Bounding the Number of Bundles in $\mathcal{S}$

We prove that  $\mathcal{S}$  contains  $O(n)$  bundles in total by showing that a constant fraction of  $\mathcal{S}$  can be converted into a planar graph.

**Definition 3** A **planar fat embedding** of a graph  $G = (V, E)$  is an embedding of  $G$  with the following properties:

- Every vertex  $v \in V$  is represented by a simply connected closed region  $P_v$ .
- Every edge  $f = (u, v) \in E$  is represented by a simply connected closed region  $P_f = P_{uv}$ .
- For any pair of vertices,  $u, v \in V$ , the two associated regions are disjoint,  $P_u \cap P_v = \emptyset$ .
- For any pair of non-incident edges,  $f, g \in E$ , the two associated regions are disjoint,  $P_f \cap P_g = \emptyset$ .

- For any pair of incident edges,  $f, g \in E$ , the interiors of the two associated regions are disjoint, while their boundaries might intersect:  $P_f \cap P_g \subseteq \partial P_f$ .
- For any edge  $f \in E$  and vertex  $v \in V$ ,  $P_f \cap P_v \neq \emptyset$  if and only if  $v$  is incident to  $f$ , i.e.  $f$  is an edge between  $v$  and some other vertex.

In other words, all vertices and edges are represented by simply connected regions. All such regions are disjoint except for between edges and vertices that share endpoints.

We now show that a planar fat embedding corresponds to a planar embedding.

**Theorem 1** *A graph  $G = (V, E)$  is planar if and only if it can be represented by a planar fat embedding.*

**Proof.** One direction is straightforward: if  $G$  is planar then there exists a regular planar embedding, which is a planar fat embedding.

To show the other direction, we need to show how to map a planar fat embedding  $\mathcal{E}$  into a regular planar embedding  $\mathcal{E}'$ . We do this by showing how to embed the graph using points for vertices and paths for edges. First, for any vertex  $v \in V$ , we place the vertex at any point  $p_v \in P_v$ . Next for any edge  $f = (u, v) \in E$ , we route the edge along the shortest path from  $p_u$  to  $p_v$  that lies completely within  $P_u \cup P_f \cup P_v$ . Notice since  $P_f$  must intersect both  $P_u$  and  $P_v$  the region is connected and a path exists.

The new embedding is planar since no two vertices can share the same point, and if two edge paths intersect the two edges must share a vertex endpoint in common. In fact, since the paths used are the shortest possible, if two edges intersect they touch and remain touching until they reach the shared endpoint. This implies that the embedding has no crossings and so  $G$  must be planar.  $\square$

To show that the number of bundles in  $\mathcal{S}$  is  $O(n)$ , we describe how it represents a planar fat embedding which implies that it also represents a planar graph that has  $O(n)$  size, where  $n$  is the number of vertices. First, we must show the relation between the number of elbow bundles and straight bundles.

**Lemma 6** *If the number of straight bundles in  $\mathcal{S}$  is  $m$  then there are at most  $O(m)$  elbow bundles.*

**Proof.** We begin by charging each elbow bundle to a straight bundle. Let  $eb_1(t)$  be a bundle and let  $eb_2(t)$  be the unique adjacent inner bundle to  $eb_1(t)$ . If  $eb_1(t)$  is the innermost bundle then we charge either of the two straight bundles that have  $eb_1(t)$  on one end. Otherwise, look at the straight bundles that have  $eb_1(t)$  on one end and  $eb_2(t)$  on the other end. Since the elbows are maximized, we know that at most one straight bundle can be common to both elbow bundles, which in turn implies that  $eb_1(t)$  has at least one straight bundle,  $sb_1(t)$ , that is different than  $eb_2(t)$ . We charge  $sb_1(t)$  for the presence of  $eb_1(t)$ .

We must now bound the total number of charges made to each straight bundle. Let  $sb(t)$  be any straight bundle. It has two associated vertices  $u$  and  $v$  each with possibly multiple elbow bundles at one or the other end of  $sb(t)$ . Let us look at the set  $E_u$  of elbow bundles that are associated with vertex  $u$  and are on one end of the straight bundle  $sb(t)$ . First, clearly the elbow bundles in  $E_u$  are adjacent

consecutively to each other, when traversing from the outermost bundle to the innermost bundle. We claim that only the innermost of these bundles can charge  $sb(t)$ . Let  $eb_1(t) \in E_u$  be any elbow bundle that charges  $sb(t)$ . Let  $eb_2(t)$  be its adjacent inner neighbor. If  $eb_2(t)$  does not exist, then  $eb_1(t)$  is the innermost of all bundles associated with  $u$  and so we are done. Otherwise, since  $eb_1(t)$  charges  $sb(t)$ , we know that  $eb_2(t)$  cannot be on any end of  $sb(t)$ , our criterion for charging a bundle. Therefore,  $eb_2(t) \notin E_u$ . Since the elbow bundles in  $E_u$  are adjacent consecutively, this implies that  $eb_1(t)$  is the innermost bundle. As a result, the straight bundle  $sb(t)$  can be charged at most twice. Therefore, there are at most  $2m$  elbow bundles.  $\square$

**Theorem 2** *Let  $\mathcal{S}$  be a compact routing structure for a set of paths  $P = \{p_1, p_2, \dots, p_n\}$  with  $O(n)$  terminal and obstacle vertices. The number of bundles stored in  $\mathcal{S}$  is  $O(n)$ . The total storage required for  $\mathcal{S}$  is  $O(n)$ .*

**Proof.** We show how the representation in  $\mathcal{S}$  can be mapped to a planar fat embedding of a graph  $G = (V, E)$ . Let  $V$  be the set of terminals and obstacles. For each vertex  $v \in V$  we let  $P_v$  be the simply connected region formed by the union of the elbow bundles associated with  $v$ . For each pair of vertices  $u, v \in V$  let  $S_{uv}$  be the set of straight bundles associated with both  $u$  and  $v$ . From Lemma 2 we know that  $|S_{uv}|$  is  $O(1)$ . If  $|S_{uv}| > 0$  we define  $f := (u, v) \in E$  and let  $P_{uv} = P_f$  be the region associated with one of these straight bundles.

These regions form a planar fat embedding for  $G = (V, E)$ . It follows from Theorem 1 that  $G$  is planar. Therefore,  $|E|$  is  $O(|V|)$  which implies that the number of straight bundles is  $O(n)$ . From Lemma 6 the total number of elbow bundles in  $\mathcal{S}$  is  $O(n)$  as well. Since  $\mathcal{S}$  stores only a constant amount of information for each bundle, the total storage space for  $\mathcal{S}$  is  $O(n)$ .  $\square$

## 5. Algorithm

The algorithm proceeds in the following steps:

1. (If necessary) compute the set of shortest paths  $P$  from the given set of paths  $P'$  with respect to vertex set  $V$ , and initialize the compact routing data structure  $\mathcal{S}$  with  $P$ .
2. Thicken the fat edges in  $P$  (maintaining  $P(t)$  in  $\mathcal{S}$ ) until maximum possible thickness.
3. Extract paths from the bundles in  $\mathcal{S}$ .

From Lemma 3 we know that the compact routing data structure can be initialized in  $O(n + k)$  time and  $O(n + k)$  space. From Lemma 5, the final extraction of the paths from the elbow and straight bundles in the compact routing structure for the maximum possible thickness can be done in time  $O(k)$ . In the remainder of this section we concentrate on step 2, the path thickening process and show that it can be done in  $O(n^3)$  time and  $O(n + k)$  space.

Let  $t \geq 0$  be a general thickness parameter, also referred to as *time*. At time frame  $t$  we assign to each path  $p \in P(t)$  with weight  $\omega$  the thickness  $\omega t$ . Starting at

$t = 0$  with all paths in  $P$  having thickness 0, we let  $t$  monotonically grow, such that the thicknesses of the paths also grow monotonically, and we maintain the invariant that the sum of the lengths of all paths is as short as possible. As the paths grow, three types of events can happen: *split events*, *merge events*, and *stop events*; see Figures 4 and 5 and Lemma 1. We construct a priority queue of events, with the key being the time at which the events occur. For increasing time we update the compact routing data structure and the event queue successively for each event. We obtain the events that we store in the queue by considering for each bundle the next event it will cause, independent of other bundles:

- For each straight bundle we store the time at which it will hit the next elbow bundle (not taking any other straight bundles into account).
- For each elbow bundle we store the time at which it will hit the next straight bundle or elbow bundle (not taking any other bundles into account).
- For each elbow bundle we store the time when it gets straightened (only taking the two incident straight bundles into account).

The first item corresponds to a split event, the second to a split or a stop event, and the third to a merge event. Note that the time at which an elbow bundle gets straightened as well as the time at which two bundles hit can be computed in constant time. Thus, for a fixed bundle the bundle it will hit next can be found in  $O(n)$  time. We initialize the event queue by inserting the next event for each bundle, which takes  $O(n + \log n)$  time per bundle, thus  $O(n^2)$  in total.

**Lemma 7** *A merge or a split event can be processed in  $O(n)$  time.*

**Proof.** For two given bundles that do not intersect at time  $t$  it takes constant time to compute, by algebraic manipulation, the smallest  $t' > t$  at which the two bundles will intersect, i.e., touch.

In a merge event a straight-elbow-straight bundle sequence is replaced by a single straight bundle  $b$  by calling the `merge` operation. Since there are  $O(n)$  bundles in total at any point in time we can compute those bundles that will intersect this new bundle in the future, together with their touching times, in  $O(n)$  time. Amongst those bundles we find the elbow bundle with the smallest touching time, which is a bundle that could possibly cause a split event with  $b$ , and insert this split event in the event queue, in total  $O(n)$  time.

In a split event one straight bundle gets replaced by a straight-elbow-straight bundle sequence by calling the `split` operation. For each of these three new bundles we compute in  $O(n)$  time, just as we did when handling the merge event, the next split events that each of those bundles might cause and insert those three events into the event queue. We also compute the time at which the new elbow bundle gets straightened, and insert this new merge event into the event queue.

During the handling of split and merge events bundles are removed from the compact routing data structure, which might still be referred to in future events in the event queue. Hence, the event queue might contain *invalid* events that refer to non-existing bundles. When such an invalid event that still contains one existing

bundle has to be processed, it is deleted from the queue, and the next event for this bundle is found and inserted in the queue in  $O(n)$  time. If the invalid event only refers to non-existing bundles it is simply removed.

Note that a split event might cause two straight bundles to become parallel such that they have to be unified. Similarly a merge event might cause a straight bundle to be divided into two straight bundles. This can be checked easily and is processed by calling the operations `union` or `divide`. Altogether there is a constant number of calls to update operations that each take  $O(n)$  time, see Lemma 4.  $\square$

**Lemma 8** *Let  $sb(t)$  be a straight bundle defined by the vertices  $v_1(t)$  and  $v_2(t)$ , where  $t$  is some time frame and  $v(t)$  denotes the union of elbow bundles associated with  $v$  at time  $t$ . Let  $v_3$  be another vertex and assume that  $v_3(t_0)$  is disjoint from  $sb(t_0)$  at time  $t_0$ .*

*Then either  $v_3(t)$  never causes a split event with  $sb(t)$  for all  $t \geq t_0$ , or there exists a time  $t_1$  such that for all  $t \geq t_1$   $sb(t)$  is never associated with  $v_1$  and  $v_2$ , i.e.,  $v_3(t)$  keeps splitting  $sb(t)$  for all  $t \geq t_1$ .*

**Proof.** It suffices to model the situation as follows: Assume  $v_1(t)$ ,  $v_2(t)$ , and  $v_3(t)$  are disks growing in a rate proportional to  $t$  and consider  $sb(t)$  to be a line tangent to  $v_1(t)$ ,  $v_2(t)$ . Then  $v_3(t)$  causes a split event with  $sb(t)$  if and only if  $v_3(t)$  is tangent to  $sb(t)$ .

In order to find the time  $t$  and the corresponding line  $sb(t)$  at which this event happens, we need to solve a system of three equations and a single variable. It is easy to see that this system has a unique solution.  $\square$

Although there are examples with  $\Omega(n)$  vertices that are each involved in  $\Omega(n)$  events with a single path, we can show that the total number of events that occur during the growth process is bounded by  $O(n^2)$ .

**Lemma 9** *The total number of events that the structure goes through is  $O(n^2)$ .*

**Proof.** Consider the case of a split event between an elbow bundle around a vertex  $v$  and a straight bundle  $sb$  defined by two vertices  $u$  and  $w$ . Without loss of generality assume  $sb$  lies above both  $u$  and  $w$ . Once  $sb$  gets split there will never again be a straight bundle between  $u$  and  $w$  touching both  $u$  and  $v$  from above, because the bundles grow thicker monotonically in time. This follows from Lemma 8. Since there are only  $O(n^2)$  possible straight bundles this proves the claim for split events.

For merge events the argument is similar: Let the two straight bundles about to merge be  $sb_1$  and  $sb_2$ . Let  $sb_1$  be defined by  $(u, v)$ , and let  $sb_2$  be defined by  $(v, w)$ . Let  $u$  be the vertex with many elbows around it that pushes the straight bundle  $sb_1$  between  $u$  and  $v$  up. Then, either  $sb_1$  or  $sb_2$  can never occur again. Indeed, once  $u$  has lifted  $sb_1$  up above  $u$  the only way to make  $sb_2$  touch  $v$  again is for a vertex to push  $sb_1$  down, but this again destroys  $sb_1$  as a straight bundle forever. Similarly  $sb_2$  could also be destroyed in order to make  $sb_1$  appear again. In any case, either  $sb_1$  or  $sb_2$  will never appear again. And since there are only  $O(n^2)$  possible straight bundles this proves the claim for merge events.

Since there are  $O(n)$  different elbow bundles the number of stop events is clearly  $O(n^2)$ . Since the processing of each split or merge event introduces only a constant

number of new events, the number of invalid events in the event queue, i.e., events that refer to non-existing straight or elbow bundles, is also  $O(n^2)$ . Thus the total number of events in the queue is  $O(n^2)$ .  $\square$

We are now ready to state our main result:

**Theorem 3** *A homotopic shift of  $P$  with respect to vertex set  $V$  with maximum separation can be computed in  $O(n^3 + k)$  time and  $O(n + k)$  space.*

**Proof.** By Lemma 3 the initialization of  $\mathcal{S}$  needs  $O(n+k)$  time and space. The growth process stops at the first stop event, see Lemma 1. From Lemma 9 we know that this will happen after at most  $O(n^2)$  events. Each event can be processed and the data structure maintained in  $O(n)$  time according to Lemma 7, which yields a total runtime of  $O(n^3)$  for the thickening process. The compact routing data structure can be uncompressed in time and space  $O(k)$ , see Lemma 5. However, we need  $O(n^2)$  space to store the events in the event queue, for a total of  $O(n^2 + k)$  space.

We can reduce the space to  $O(n+k)$  without increasing the asymptotic running time as follows: Instead of keeping all future events, we divide the execution of the algorithm into  $O(n)$  phases, where in each phase  $O(n)$  events happen. The priority queue then contains only the next  $O(n)$  events, so only  $O(n)$  space is needed. After each phase, we find the next  $O(n)$  events by checking each bundle against every other bundle (which is done in  $O(n^2)$  time), so the total time spent for finding future events remains  $O(n^3)$ , as before.  $\square$

## 6. General Planar Graphs

We now extend the Fat Edge Drawing (FED) problem to general graphs by dropping the maximum degree condition. Recall the General Fat Edge Drawing (GFED) problem: given a weighted planar graph (not necessarily of degree 1) and an embedding for it, find a planar drawing with the edges drawn as thick as possible with thicknesses proportional to the edge weights.

It is easy to extend our algorithm to an algorithm for general planar graphs. We can modify our algorithm as follows: Let each vertex grow at a rate proportional to its degree. In this setting our modified algorithm will find the optimum solution. However, the solution may not be optimal in the sense that some vertices may occupy more space than they need, thus causing the algorithm to terminate earlier; see Figure 11.

This problem can be addressed by allowing vertices to have a variable rate of growth as follows. Each vertex is the smallest circle such that its adjacent edges do not overlap outside that circle. Note that the largest diameter circle needed for a vertex of degree  $i$  is  $i$ . The problem with this approach is that the angles of the adjacent edges change dynamically throughout the algorithm and hence would require updates at every event for every elbow.

## 7. Open Problems

Some of the open problems related to the FED and GFED problems include:

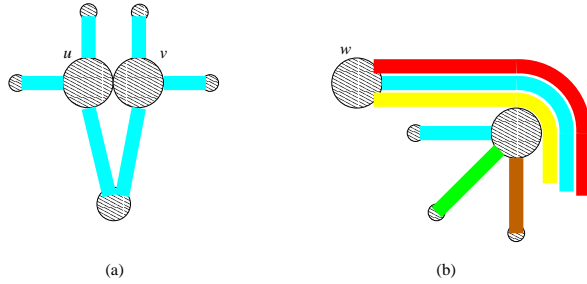


Fig. 11. The FED algorithm applied to general planar graphs. In both graphs shown the max degree is 3 and all vertices have uniform weights. Vertices of different degree have different sizes, more precisely, a vertex of degree  $i$  has diameter  $i$ . (a) The algorithm stopped because vertices  $u$  and  $v$  touched. However, vertices  $u$  and  $v$  need not be that large since their edges “fan out.” (b) In this case vertex  $w$  has to be large since its edges do not fan out.

- Is there a data structure that allows to find the next event for a bundle in polylogarithmic time (instead of linear time)?
- Can the FED algorithm be extended to non-point obstacles?
- What is a “good” model for the GFED problem?
- Can the FED algorithm be modified to an algorithm for the GFED problem with vertices growing at varying rates?

## Acknowledgements

We would like to thank Jack Snoeyink for introducing us to the homotopic routing problem. We also thank him and Cesim Erten for many fruitful discussions.

## References

1. G. Barequet, M. Goodrich, and C. Riley. Drawing planar graphs with large vertices and thick edges. *J. of Graph Algorithms and Applications (JGAA)*, 8(1):2–20, 2004.
2. S. Cabello, Y. Liu, A. Mantler, and J. Snoeyink. Testing homotopy for paths in the plane. *Discrete & Computational Geometry*, 31(1):61–81, 2004.
3. R. Cole and A. Siegel. River routing every which way, but loose. In *25th Annual Symposium on Foundations of Computer Science*, pages 65–73, Los Angeles, Ca., USA, Oct. 1984. IEEE Computer Society Press.
4. D. Dolev, K. Karplus, A. Siegel, A. Strong, and J. D. Ullman. Optimal wiring between rectangles. In *Conference Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computation*, pages 312–317, Milwaukee, Wisconsin, 11–13 May 1981.
5. C. Duncan, A. Efrat, S. Kobourov, and C. Wenk. Drawing graphs with fat edges. In *9th Symp. on Graph Drawing (GD)*, pages 162–177, 2001.
6. A. Efrat, S. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. *Computational Geometry: Theory and Applications*, to appear.

7. A. Efrat, S. Kobourov, M. Stepp, and C. Wenk. Growing fat graphs. In *18th Annual Symposium on Computational Geometry (SoCG)*, pages 277–278, 2002. Video contribution. <http://www.cs.arizona.edu/people/kobourov/PROJECTS/fatedges.avi>.
8. S. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, W. Rülling, and C. Storb. On continuous homotopic one layer routing. In *Proceedings of the Fourth Annual Symposium on Computational Geometry (Urbana-Champaign, IL, June 6–8, 1988)*, pages 392–402, New York, 1988. ACM, ACM Press.
9. M. Kaufmann and K. Mehlhorn. Routing through a generalized switchbox. *Journal of Algorithms*, 7(4):510–531, Dec. 1986.
10. M. Kaufmann and K. Mehlhorn. On local routing of two-terminal nets. *JCTB: Journal of Combinatorial Theory, Series B*, 55, 1992.
11. C. E. Leiserson and F. M. Maley. Algorithms for routing and testing routability of planar VLSI layouts. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 69–78, Providence, Rhode Island, 6–8 May 1985.
12. C. E. Leiserson and R. Y. Pinter. Optimal placement for river routing. *SIAM Journal on Computing*, 12(3):447–462, Aug. 1983.
13. F. M. Maley. *Single-Layer Wire Routing*. PhD thesis, Massachusetts Institute of Technology, 1987.
14. A. Mirzaian. River routing in VLSI. *Journal of Computer and System Sciences*, 34(1):43–54, Feb. 1987.
15. J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17(4):717–728, 2001.
16. R. Pinter. River-routing: Methodology and analysis. In *Third Caltech Conference on VLSI*, pages 141–163, 1983.
17. D. Richards. Complexity of single layer routing. *IEEE Transactions on Computers*, 33:286–288, 1984.
18. A. Schrijver. Disjoint homotopic paths and trees in a planar graph. *Discrete & Computational Geometry*, 6:527–574, 1991.
19. A. Schrijver. Edge-disjoint homotopic paths in straight-line planar graphs. *SIAM Journal on Discrete Mathematics*, 4(1):130–138, Feb. 1991.