Drawing with Fat Edges[*]

Christian A. Duncan[1], Alon Efrat[2], Stephen G. Kobourov[2], and
Carola Wenk[3][**]

[1] Department of Computer Science
University of Miami
Coral Gables, FL 33124
`duncan@cs.miami.edu`
[2] Department of Computer Science
University of Arizona
Tucson, AZ 85721
`{alon,kobourov}@cs.arizona.edu`
[3] Institut für Informatik
Freie Universität Berlin
Berlin, Germany
`wenk@inf.fu-berlin.de`

Abstract. In this paper, we introduce the problem of drawing with "fat" edges. Traditionally, graph drawing algorithms represent vertices as circles and edges as closed curves connecting the vertices. In this paper we consider the problem of drawing graphs 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 take an arbitrary wire routing and convert it into a homotopic equivalent routing such that the distance between any two wires is maximized. Moreover, the routing uses the minimum length wires. Maximizing the distance between wires is equivalent to finding the drawing in which the edges are drawn as thick as possible. To the best of our knowledge this is the first algorithm that finds the maximal distance between any two wires and allows for wires of variable thickness. The previous best known result for the corresponding decision problem with unit wire thickness is the algorithm of Gao *et al.*, which runs in $O(kn^2 \log(kn))$ time and uses $O(kn^2)$ space, where n is the number of wires and k is the maximum of the input and output complexities. The running time of our algorithm is $O(kn + n^3)$ and the space required is $O(k + n)$. The algorithm generalizes naturally to general planar graphs as well.

---

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 natural to assign to the edges a width (or thickness) proportional to their weights. If the weights are in a large range then a logarithmic scale may be used. Surprisingly, there does not seem to be any previous work on drawing graphs with edges of varying thickness.

Some related work has been done in addressing a classic VLSI problem, the continuous homotopic routing problem (CHRP) [2, 9]. For the CHRP problem, we need to route wires with fixed terminals among fixed obstacles when a sketch of the wires is given, i.e., each wire is given a specified homotopy class. If the wiring sketch is not given or the terminals are not fixed, the problem is NP-hard [10, 14, 15]. In the CHRP problem we are given a wiring layout and some constant ϵ and we want to find if this wiring can be continuously transformed to a new wiring in which the wire separation is at least ϵ. In this setting the graph is given a fixed planar embedding and the maximum degree is 1. It is easy to see that the CHRP problem can be rephrased as the following graph drawing problem: does there exist a planar drawing in which all the wires can be drawn with thickness ϵ?

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 CHRP 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

Some of the early work on continuous homotopic routing was done by Cole and Siegel [2] and Leiserson and Maley [9]. They show that in L_∞ norm a solution can be found in $O(k^3 \log n)$ 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. Maley [11] shows how to extend the distance metric to arbitrary polygonal distance functions (including Euclidean distance) and presents a $O(k^4 \log n)$ time and $O(k^4)$ space algorithm. Note that k cab be arbitrarily larger than n. In fact, it is easy to construct examples in which k is arbitrarily large. More surprisingly, even after shortest paths have been computed for each wire, k can be as large as $k = \Omega(2^n)$, see Figure 1. The best result so far is due to Gao *et al.* [4] who present a $O(kn^2 \log(kn))$ time and $O(kn^2)$ space algorithm.
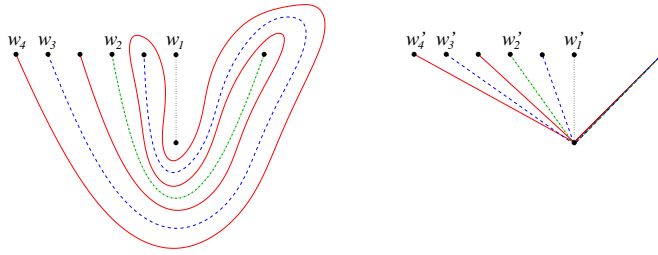
Fig. 1. An example with exponential complexity: $k = \Omega(2^n)$: On the left is the initial wiring sketch, and on the right is the wiring after the shortest paths have been computed. The number of edge segments in the shortest paths w_1', w_2', w_3', w_4' is $1, 2, 4, 8$. In general, wire w_i' has 2^{i-1} edge segments. Note that on the right many edge segments are parallel.

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 [6, 7], and by Schrijver [16, 17]. Work in this area is related to the river routing problem in VLSI chips [2, 3, 12, 14].

A related problem was considered by Pach and Wenger [13]. They address the problem of 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, we 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 with at most $O(n)$ bends per edge using the algorithm of Pach and Wenger [13]. In the wire routing setting, this implies that $k = \theta(n^2)$.

As a part of our algorithm we use a geometric shortest path algorithm. For triangulated polygons, the Euclidean shortest path between two points can be computed in linear time using the algorithms of Chazelle [1] or Lee and Preparata [8]. The latter algorithm is known as the funnel algorithm and it can be extended to river routing [2, 4, 9]. In our setting, the shortest paths can be found in optimal $O(nk)$ time using the algorithm of Hershberger and Snoeyink [5].

1.2   Our Results

We show how to solve the FED problem in $O(nk + n^3)$ time and $O(n + k)$ space. We describe the algorithm in the tradition of the homotopic wire routing where n is the number of wires and k is the maximum of the initial and final complexities of all the 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 CHRP problem and is an improvement in both space and time complexity over the best known algorithm for the CHRP problem.

2 Continuous Homotopic Routing

In this paper we use some basic definitions from Gao et al. [4]. Let $W = \{w_1, w_2, \ldots, w_n\}$ be a set of *wires* (also called *paths*). *Paths* are non-intersecting planar continuous curves. Such a collection of wires is called *collisionfree*. Let $T = \{w_i(0), w_i(1) : 1 \le i \le n\}$ be the set of *terminals* (endpoints of the wires) of W, $|T| = 2n$. Let $V = T \cup O$ be the set of *vertices*, where O is a set of at most $O(n)$ point *obstacles* disjoint from W.

Let $p, q : [0, 1] \longrightarrow \mathbb{R}^2$ be two continuous curves parameterized by arc-length. Then p and q are homotopic with respect to a set $V \subseteq \mathbb{R}^2$ if there exists a continuous function $h : [0, 1] \times [0, 1] \to \mathbb{R}^2$ with the following three properties:

1. $h(0, t) = p(t)$ and $h(1, t) = q(t)$, for $0 \le t \le 1$
2. $h(\lambda, 0) = p(0) = q(0)$ and $h(\lambda, 1) = p(1) = q(1)$ for $0 \le \lambda \le 1$
3. $h(\lambda, t) \notin V$ for $0 \le \lambda \le 1, 0 < t < 1$

We call a set of paths $P = \{p_1, p_2, \ldots, p_n\}$ a *homotopic shift* of a set of wires $W = \{w_1, w_2, \ldots, w_n\}$ if the following two conditions are met:

1. p_i is homotopic to w_i with respect to $V, 1 \le i \le n$
2. P is collisionfree, i.e., no two paths in P intersect

Let for each path p a weight $\omega_p > 0$ be given. Let the weighted distance between two paths p and q be the minimum weighted distance between all point pairs u and v with $u \in p$ and $v \in q$, where the weighted distance between u and v is the Euclidean distance multiplied by $2/(\omega_p + \omega_q)$. Note that if all weights have unit weight this yields the Euclidean distance. Then the separation, $s(P)$, of a set of paths P is defined to be the minimum weighted distance between any two paths $p_i, p_j \in P$.

In this paper we address the following Fat Edge Drawing (FED) problem: Given a set $W = \{w_1, w_2, \ldots, w_n\}$ of wires with terminal set T and obstacle set O. Furthermore, for each wire w_i let an associated weight $\omega_i > 0$ be given. Find a homotopic shift $P = \{p_1, p_2, \ldots, p_n\}$ of W with maximum separation, i.e., $s(P) \ge s(Q)$ for all homotopic shifts Q of W.

Note that the maximum separation for a set of paths P directly yields the maximum thickness with which the paths in P can be drawn without intersections. Indeed, if the weighted distance between two paths p and q is $2\epsilon/(\omega_p + \omega_q)$, then p can be drawn with thickness $\omega_p \epsilon/(\omega_p + \omega_q)$, q can be drawn with thickness $\omega_q \epsilon/(\omega_p + \omega_q)$, and p and q are disjoint. Computing a homotopic shift for W, T, and O with maximum separation thus corresponds to drawing a planar graph, with vertex degrees at most 1, with edges as thick as possible.

3 Algorithm Overview

We are now ready to outline the general technique for finding the continuous homotopic routing of maximum separation. We begin with the initial set of wires

W and compute for each $w \in W$ the shortest path w' homotopic to w, which yields a set of shortest paths W' that is a homotopic shift of W. This can be done in $O(nk)$ time using a result from Hershberger and Snoeyink [5]. The idea of the shortest path computation is to triangulate the region using the $O(n)$ terminals and obstacles, yielding a triangulation of size $O(n)$. A shortest path w' homotopic to a given path w can be computed in time $O(C_w + \Delta_w)$, where C_w is the complexity of w and Δ_w is the number of times w intersects a triangulation edge which is $O(nC_w)$. Since $\sum_{w \in W} C_w = k$, the total time taken to compute all shortest paths becomes $O(k + nk) = O(nk)$. The storage required by this algorithm is $O(n + k)$.

From W' we compute a wire routing with maximum separation by applying the following kinetic approach: We let the wires simultaneously grow in thickness over time, in a speed proportional to the individual weight of each wire. Throughout this growth process, the wires remain as short as possible, and the homotopy between wires is preserved. As a result, the line segments of the original polygonal paths are deformed into two types of curves. We borrow the names for these curves from plumbing jargon: *straights* and *elbows*. *Straights* are rectangular regions and *elbows* are formed by the arc of an annulus with two given radii, Figure 2. We examine elbows and straights in depth in Section 4.

We introduce a compact routing data structure which allows us to represent the thick wires, i.e., the straights and elbows, and to maintain the growth process efficiently. From the beginning to the end of the growth process, the algorithm that maintains the data structure requires $O(n^3)$ time and $O(n^2 + k)$ space. In order to achieve this time bound independent of the complexity of the wires k (which can be as large as $\Omega(2^n)$), we group wires together. Note that in W' many of the wires may travel in parallel, see Figure 1 for an example. We take advantage of this and group such parallel wires 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 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 wires which consist of straight line segments and circular arcs.

4 Compact Routing Structure

4.1 Straights, Elbows, and Bundles

Definition 1. The wires in the routing are broken into connected sequences of fat edge segments called *elbows* and *straights*, as described below (see Figure 2). The thickness of each such segment f is determined by $\omega_f t$, where ω_f is the weight of the segment that corresponds to the weight of its initial wire, and $t > 0$ represents the current time frame. Let $V := T \cup O$ be the set of vertices.

– An **elbow** segment, e, associated with a vertex v and two straight segments from v to u and from v to w; $u, v, w \in V$; is a connected region of the
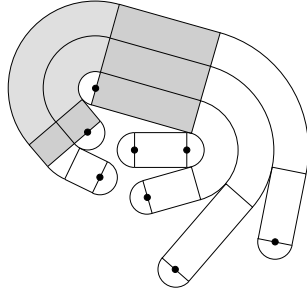
Fig. 2. An example of wires composed of straights and elbows. The shaded regions are bundles.

plane, formed by a piece of an annulus centered at v and having thickness $w_f t$. More formally $e = \{p \in \mathbb{R}^2 : r_1 \leq ||p - v|| \leq r_2$ with $r_2 - r_1 = w_f t$ and the orientation of the segment $\overline{pv} \in [\theta_1, \theta_2]\}$. The elbow e maintains the property that for any point $p \in e$ the line segment $\overline{pv}$ intersects only other elbows associated with vertex v. Note that the half-disk and the disk centered around v are also elbows. For notation, let us call these regions **terminal elbows**. Let $E_v = \{e_1, e_2, \ldots e_d\}$ be the set of elbows associated with v.

– A **straight** segment, s, associated with a segment f from $u \in V$ to $v \in V$ is a rectangular region (fully) connecting two elbows, $e_v \in E_v$ and $e_u \in E_u$. The thickness of the segment is equal to $w_f t$. Let $S_{uv} = \{s_1, s_2, \ldots s_d\}$ be the set of straight segments associated with edges between u and v.

Note that every non-terminal elbow has two associated straight segments. Every terminal elbow has at most one straight segment, the one emanating from the corresponding vertex. Throughout the algorithm the following disjointness property is maintained:

Property 1. The only way two segments can intersect is along their boundary while the interiors are disjoint. The only possible intersections are:

– Two straight segments intersect only if they are associated with the same two vertices (u, v).
– Two elbow segments intersect only if they have the same associated vertex.
– A straight segment s intersects an elbow segment e if and only if e is one of s's two connecting segments.

The total number of elbow and straight segments depends on k which can be very large, see Figure 1. In order to reduce this complexity, straights and elbows are bundled and only the bundles are manipulated.

Definition 2. *Straight bundles are made of straight edge segments and elbow bundles are made of elbow segments as described below:*

– *A* **straight bundle***, sb, of* (u, v) *is the rectangular region formed by the union of straights associated with edges from* u *to* v. *We assume that the bundles are maximized, that is, for* $s_1, s_2 \in S_{u,v}$, *the straights* s_1 *and* s_2 *intersect if and only if* s_1 *and* s_2 *belong to the same bundle. Let* $SB_{uv} = \{sb_1, sb_2, \ldots sb_d\}$ *be the set of straight bundles associated with* u *and* v.
– *An* **elbow bundle***, eb, of* v *is a connected region formed by the union of elbows associated with the same vertex and sharing the same straight bundles on both ends. As with straight bundles, we assume that the elbow bundles are maximized. Let* $EB_v = \{eb_1, eb_2, \ldots, eb_d\}$ *be the set of elbow bundles associated with* v. *A terminal elbow has at most one straight segment and belongs to its own elbow bundle.*

For any vertex v there may be many elbow bundles. It is not hard to show that for any pair of vertices (u, v) there may be at most 3 straight bundles between them — one below both vertices, one above both vertices, and a "diagonal" bundle, that is below one vertex and above the other.

Lemma 1. *If the number of straight bundles in our compact routing structure is* m *then there are at most* $O(m)$ *elbow bundles.*

Proof Sketch: This proof relies on the fact that the bundles are "ordered" on both sides of a group of elbows. As one elbow bundle ends, one straight bundle must also end on one of the two sides of the elbow groups.   □

4.2   Compact Routing Structure $\mathcal{S}$

Given the two types of segments and bundles we show how they can be stored and updated in a manageable data structure.

Definition 3. *The compact routing structure* $\mathcal{S}$ *represents a fat-wire routing as follows:*

– *Terminals and obstacles are stored with pointers to their terminal elbows*
– *Elbow and straight bundles are stored along with their weights*
– *Elbow and straight bundles store the ordered list of wires they represent*
– *A straight bundle sb stores:*
  • *two linked lists of left and right adjacent elbow bundles*
  • *the number of straight segments it represents*
  • *its weight (the sum of the weights of each of its straight segments)*
– *An elbow bundle eb stores:*
  • *the set of elbow bundles adjacent to it (both above and below)*
  • *the two adjacent straight bundles*
  • *the number of elbow segments that it represents*
  • *its weight (the sum of the weights of each of its elbow segments)*
  • *its* layered *weight (the sum of the weights of each of the elbow segments between the elbow bundle and the vertex center, not counting its own weight)*
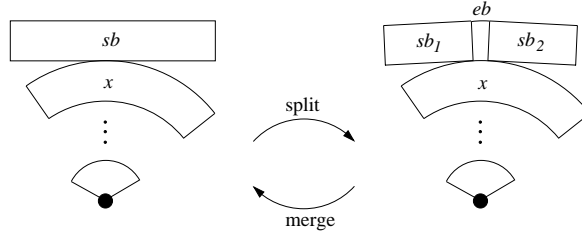
Fig. 3. Split and merge operations. The split operation splits the straight bundle at an elbow bundle into a straight-elbow-straight bundle sequence. The merge operation merges a sequence of straight-elbow-straight bundles into a single straight bundle.

After the homotopic shortest paths have been computed, we need to find out which bundles participate in the compact routing structure and what is the order of the wires inside each bundle. The next lemma descirbes this process.

Lemma 2. *Given a set of shortest paths W' the compact routing structure $\mathcal{S}$ can be initialized in $O(n^2 + k)$ time.*

For the proof of this lemma see Appendix A.

4.3 Maintaining $\mathcal{S}$

Lemma 3. *We can maintain the following operations in constant time:*

- report(sb **or** eb) *is an operation that returns a description of the bundle at the current time frame*
- split(sb, x) *(see Figure 3) is an operation that splits a straight bundle sb into three connected bundles sb_1, eb, and sb_2 such that:*
  - x *is an elbow bundle that initially intersects sb at the boundary*
  - sb_1 *is adjacent to all elbow bundles on the left portion of sb*
  - sb_2 *is adjacent to all elbow bundles on the right portion of sb*
  - eb *is adjacent to sb_1 and sb_2 and is associated with the same vertex as x*
  - *The wire lists, weights, and number of segments for the three new bundles are the same as the wire list, weight, and number of segments for sb*
- merge(sb_1, eb, sb_2) *(see Figure 3) is an operation that merges two straight bundles and an elbow bundle into one straight bundle sb such that:*
  - eb *is adjacent to sb_1 and sb_2*
  - eb *has arc length 0, implying that sb_1 intersects sb_2*
  - sb *represents the region $sb_1 \cup sb_2$ and is adjacent to the left elbow bundles of sb_1 and the right elbow bundles of sb_2*
  - *The wire lists, weights and number of segments for all the bundles are the same*
- bundlemerge(sb_1, sb_2) *is an operation that merges two straight bundles $sb_1, sb_2 \in SB_{uv}$, that intersect along their boundary, into a single straight bundle sb such that:*

- the weight (the number of straights) for sb is the sum of the weights (the numbers of straights) for sb_1 and sb_2
- the linked list of left and right elbow bundles of sb is the concatenation of the corresponding linked lists of sb_1 and sb_2
- the wire list of sb is the concatenation of the wire lists for sb_1 and sb_2

If the left (resp. right) elbow bundles of sb are adjacent to the same straight bundle to their left (resp. right), then those elbow bundles get merged into a single elbow bundle eb. The wire list, weight, layered weight, and number of elbows eb represents follow from the corresponding entries in the elbow bundles being merged.

- bundlesplit(sb, eb) is an operation that splits a straight bundle $sb \in SB_{uv}$ at the adjacent elbow bundle eb into two straight bundles $sb_1, sb_2 \in SB_{uv}$, that intersect along their boundary, such that:
    - eb is the first elbow bundle in the left or right adjacency list of sb_2
    - the linked lists of left and right elbow bundles of sb are the concatenation of the corresponding linked lists of sb_1 and sb_2
    - the wire list of sb is the concatenation of the wire lists for sb_1 and sb_2
    - the weight (the number of straights) for sb_2 is the sum of the weights (the number) of the adjacent elbow bundles
    - the weight (the number of straights) for sb is the sum of the weights (the numbers of straights) for sb_1 and sb_2

If sb is adjacent (on the side not adjacent to eb) to exactly one elbow bundle, then this elbow bundle also gets split into two elbow bundles. The weights, layered weights, and numbers of elbows those two elbow bundles represent follow from the entries in sb_1 and sb_2.

Lemma 4. *Given $\mathcal{S}$ we can uncompress the bundles of $\mathcal{S}$ into straight segments and elbow segments in time $O(k)$ where k is the final complexity of the paths.*

Proof Sketch: The idea is to go through the structure using the wire identities in the wire list and greedily unzip the bundles segment by segment starting at any terminal and progressing along its path. The paths obtained in this way consist of straight line segments and circular arcs. □

4.4 Bounding the number of bundles in $\mathcal{S}$

In this section we argue that our compact routing structure $\mathcal{S}$ contains $O(n)$ bundles in total. We do this by showing that $\mathcal{S}$ is (nearly) a planar graph implying the stated size.

Definition 4. *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: $P_f \cap P_g \subseteq \delta P_f$.*
- *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 is indeed 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 Sketch: 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 our compact routing structure 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.

Theorem 2. *Let $\mathcal{S}$ be a compact routing structure for a set of wires $W = \{w_1, w_2, \ldots, w_n\}$ 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 $O(k)$.*

Proof Sketch: 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 regions in EB_v (the elbow bundles of v). For each pair of vertices $u, v \in V$ let us consider the set of straight bundles SB_{uv}. We know that $|SB_{uv}| \leq 3$. If $|SB_{uv}| > 0$ we define $f := (u, v) \in E$ and let $P_{uv} = P_f$ be one of the three bundle regions, namely sb_1.
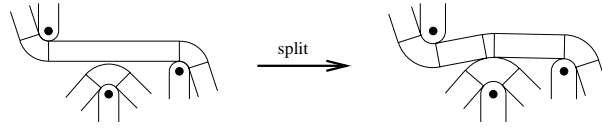
Fig. 4. A split event. The elbow bundle pushes the straight bundle up such that it gets replaced by a straight-elbow-straight bundle sequence.

These regions form a planar fat embedding for $G = (V, E)$. It follows from Theorem 1 that G is planar. Therefore, $|E|$ is $O(|V|)$ and the total number of bundles stored in $\mathcal{S}$ is $O(n)$.

Since besides the lists of wires $\mathcal{S}$ stores only a constant amount of information for each bundle, the total storage space for $\mathcal{S}$ is $O(k + n) = O(k)$. □

5 Algorithm

The algorithm uses the following steps:

1. Compute the set of shortest paths W' from the given wire set W
2. Initialize the compact routing data structure $\mathcal{S}$ with W'
3. Thicken the wires in W' (maintaining $\mathcal{S}$) until maximum possible thickness
4. Extract paths from the bundles in $\mathcal{S}$

We argued in Section 3 that the computation of the shortest paths can be done in time $O(nk)$ with $O(n+k)$ space complexity. In Section 4 we argued that the initialization of the compact routing data structure, i.e., the bundling of the edges, can be done in $O(k+n^2)$ time. 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 $O(k)$, see Section 4. In the remainder of this section we concentrate on step 3, the wire thickening process, and show that it can be done in $O(n^3)$ time and $O(k + n)$ space.

Let $t \geq 0$ be a general thickness parameter which we also refer to as *time*. At time frame t we assign to each wire $w' \in W'$ with weight ω' the thickness $\omega' t$. Starting at $t = 0$ with all wires in W' having thickness 0, we let t monotonically grow, such that the thicknesses of the wires also grow monotonically, and we maintain the invariant that the wires are as short as possible. As the wires grow three types of events can happen: *Split events*, *merge events*, and *stop events*. In a split event (see Figure 4) an elbow bundle touches and then bends a straight bundle, such that this straight bundle gets replaced by a straight-elbow-straight bundle sequence. In a merge event (see Figure 5) an elbow bundle straightens, such that the corresponding straight-elbow-straight bundle sequence gets replaced by a single straight bundle. In a stop event two elbow bundles touch each other, which means that at this time the growth process stops, because two elbow bundles cannot bend or push each other away anymore.

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
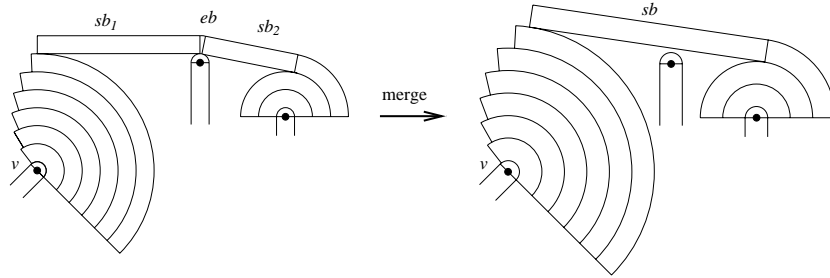
Fig. 5. A merge event. Vertex v pushes the left straight bundle sb_1 up such that the elbow bundle eb in the middle disappears. The straight-elbow-straight bundle sequence $sb_1 - eb - sb_2$ gets replaced by a single straight bundle sb.

structure and the event queue successively for each event. We obtain the events 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 hits the next elbow bundle (not taking any other straight bundles into account).
- For each elbow bundle we store the time at which it hits 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 5. *Merge or split event are done in $O(n)$ time.*

For the proof of this lemma see Appendix A.

Lemma 6. *Let $sb(t) \in SB_{v_1 v_2}$ 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 EB_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 $t \geq t_1$ $sb(t) \notin SB_{v_1 v_2}$, i.e., $v_3(t)$ keeps splitting $sb(t)$ for all $t \geq t_1$.

Proof Sketch: It can be seen that it suffices to model the situation as follows: Assume $v_1(t)$, $v_2(t)$, and $v_3(t)$ are disks growing proportional to t, and consider

$sb(t)$ to be a line tangent to $v_1(t)$, $v_2(t)$. Assume that $v_1(t)$ and $v_2(t)$ lie on different sides of $sb(t)$. Using the tangency conditions and similarity of triangles it is easy to show that $sb(t)$ rotates around a fixed center for increasing t. Plugging in another tangency condition for $v_3(t)$ gives a unique time where $sb(t)$ is tangent to $v_1(t)$, $v_2(t)$, and $v_3(t)$. Details are omitted from this extended abstract. □

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

Lemma 7. *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. WLOG let sb lie 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 6. 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 and sb_2 be defined by (u, v), and (v, w), respectively. 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 $O(n^2)$. □

The growth process stops at the first stop event. From Lemma 7 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 5, which yields a total runtime of $O(n^3)$. We need $O(k)$ space for the compact routing data structure and $O(n^2)$ space to store the events, for a total of $O(n^2 + k)$. This directly yields our main theorem:

Theorem 3. *The continuous homotopic wire routing with maximum separation can be computed in $O(n^3 + nk)$ time and $O(n + k)$ space.*

Proof. Lemmas Lemma 5 and Lemma 7 yield the result save for the $O(n + k)$ space. We can reduce the space needed for the algorithm from $O(n^2 + k)$ to $O(n+k)$ without increasing the asymptotic running time. This is done as follows: After computing the compact routing structure, the only place where superlinear space is needed is in the priority queue, used for discovering future events.

(a)   (b)
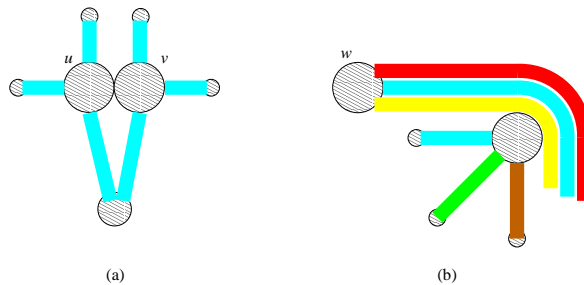
Fig. 6. The FED algorithm applied to general planar graphs. In both graphs shown the max degree is 3 and all vertices have uniform weights. Vertices of different degree have different sizes, more precisely, a vertex of degree i has diameter i. (a) The algorithm stopped because vertices u and v touched. 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.

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 each 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. □

6 General Planar Graphs

We presented a $O(nk + n^3)$ time algorithm for the FED problem. Since the FED problem restricts us to graphs with maximum degree 1, we would like to extend it to general graphs which yields the GFED problem. It is easy to extend our algorithm to an algorithm for general planar graphs. Recall the 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. 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 6.

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:

- Is $k = \Omega(2^n)$ the worst case complexity for the FED problem or can it be worse than that?
- 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?

References

1. B. Chazelle. A theorem on polygon cutting with applications. In *23th Annual Symposium on Foundations of Computer Science*, pages 339–349, Los Alamitos, Ca., USA, Nov. 1982. IEEE Computer Society Press.
2. 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.
3. D. Dolev, K. Karplus, A. Siegel, A. Strong, and J. D. Ullman. Optimal wiring between rectangles. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computation*, pages 312–317, Milwaukee, Wisconsin, 11–13 May 1981.
4. S. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, W. Rülling, and C. Storb. On continuous homotopic one layer routing. In *Proceedings of the 4th Annual Symposium on Computational Geometry*, pages 392–402, New York, 1988. ACM Press.
5. Hershberger and Snoeyink. Computing minimum length paths of a given homotopy class. *CGTA: Computational Geometry: Theory and Applications*, 4, 1994.
6. Kaufmann and Mehlhorn. On local routing of two-terminal nets. *JCTB: Journal of Combinatorial Theory, Series B*, 55, 1992.
7. M. Kaufmann and K. Mehlhorn. Routing through a generalized switchbox. *Journal of Algorithms*, 7(4):510–531, Dec. 1986.
8. D. T. Lee and F. P. Preparata. Euclidean Shortest Paths in the Presence of Rectilinear Barriers. *Networks*, 14(3):393–410, 1984.
9. 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.
10. C. E. Leiserson and R. Y. Pinter. Optimal placement for river routing. *SIAM Journal on Computing*, 12(3):447–462, Aug. 1983.
11. F. M. Maley. *Single-Layer Wire Routing*. PhD thesis, Massachusetts Institute of Technology, 1987.
12. A. Mirzaian. River routing in VLSI. *Journal of Computer and System Sciences*, 34(1):43–54, Feb. 1987.
13. J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. In *Proc. 6th Int. Symp. Graph Drawing (GD '98)*, pages 263–274, 1998.
14. R. Pinter. River-routing: Methodology and analysis, 1983.
15. D. Richards. Complexity of single layer routing. *IEEE Transactions on Computers*, 33:286–288, 1984.
16. Schrijver. Disjoint homotopic paths and trees in a planar graph. *Discrete & Computational Geometry*, 6, 1991.
17. A. Schrijver. Edge-disjoint homotopic paths in straight-line planar graphs. *SIAM Journal on Discrete Mathematics*, 4(1):130–138, Feb. 1991.

A Appendix

Lemma 2. *Given a set of shortest paths W' the compact routing structure $\mathcal{S}$ can be initialized in $O(n^2 + k)$ time.*

Proof. Assume for simplicity that there are no three collinear vertices. We start with bundling multiple edge segments between the same pair of vertices into one bundle, which can be done by traversing the wires in $O(n^2 + k)$ time. This process yields a planar graph with $O(n)$ vertices and (bundled) edges. Our next goal is to assign a linked list $\mathcal{L}_e$ of wire names to each bundle e which represents the order of the wires inside e. Note that the same wire can appear many times. We define an *arc* to be a connected part of the wire that uses (the interior) of a bundle edge. Thus, a bundle edge might contain many arcs that all belong to the same wire. We maintain with each bundle its "capacity" which is the number of arcs using this bundle. We sort the bundles around each vertex, according to their orientations, which can be done in $O(n \log n)$ time.

Since wires in W' are shortest paths, the turn angle of a wire around a vertex v must be at least π. Therefore we can assign to every vertex v_i an angle θ_i and a ray r_i emerging from v_i in orientation θ_i as follows: For each wire w that touches v_i, where v_i is not one of w's terminals, the number of arcs in w touching v_i with orientation in the range $[\theta_i - \pi, \theta_i)$ is the same as the number of arcs touching v_i with orientation in the range $[\theta_i, \theta_i + \pi)$. For a fixed bundle e adjacent to a vertex v_i, consider the division of the other bundles adjacent to v_i that lie on the same side of r_i as e into the set of bundles with smaller and the bundles with greater orientation angles than e. We store for each bundle e the sum of the capacities of the bundles in both sets.

We next traverse the graph along each wire w. Assume that $a_1, a_2 \ldots$ are the arcs of w, in the order they appear along w, and assume a_i is the arc leading from vertex v_i to vertex v_{i+1}, so v_1 is one of the terminals of w. Note that v_i is not necessarily distinct from v_j.

Suppose that we know the position of arc a_i in the bundle carrying a_i. Using this information we compute the position of a_{i+1} in its bundle as follows. Let e_i and e_{i+1} be the bundles carrying a_i and a_{i+1}, respectively. Consider the other bundles adjacent to v_{i+1} whose orientations lie in the greater angle between e_i and e_{i+1}. Then r_{i+1} divides these bundles into those being on the same side of r_{i+1} as e_i, and those being on the same side as e_{i+1}. However, we know that the sum of arcs in the bundles on the side of e_i plus the position of a_i in e_i has to equal the corresponding sum on the other side of r_{i+1}. Using the sums of bundle capacities that we stored for e_i and e_{i+1} we can compute the position of a_{i+1} in e_{i+1} in constant time. Starting with a_1, which has to be the uppermost or lowermost arc in the bundle that contains it, we can thus traverse w incrementally and track the position of the arcs in the bundles in time proportional to the length of w. Therefore all lists $\mathcal{L}_e$ can be constructed in $O(k)$ time in total. $\square$

Lemma 5. *Merge or split event are done in $O(n)$ time.*

Proof. In a split event one straight bundle gets replaced by a straight-elbow-straight bundle sequence. For each of the three new bundles we need to compute its next event and insert it in the event queue. This can be done in $O(n)$ time.

In a merge event a straight-elbow-straight bundle sequence gets replaced by a single straight bundle. Clearly the next elbow bundle causing a possible split event with this new straight bundle can be found and inserted in the queue in $O(n)$ time.

Both split and merge events cause other events in the event queue to become invalid because bundles are removed from the compact routing data structure which might still be referred to in some events in the queue. When an invalid event which 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 merged. Similarly a merge event might cause a straight bundle to be split into two straight bundles. This can be checked easily and done in constant time using the operations `bundlesplit` and `bundlemerge` of the compact routing structure described in Section 4. $\square$