Growing Fat Graphs

A. Efrat, S. Kobourov,
M. Stepp and C. Wenk
Dept. of Computer Science
University of Arizona
Tucson, AZ 85721

1. INTRODUCTION

We present an algorithm for growing fat graphs. Traditionally, graph drawing algorithms represent vertices as circles and edges as closed curves connecting the vertices. The thickness of an edge is often used as a visualization cue, to indicate importance, or to convey some additional information. We show how to grow fat graphs with edges of variable thickness. For the purpose of the demonstration we focus on a restricted class of graphs that occur in VLSI wire routing. This class corresponds to planar, max-degree-1 graphs. The underlying algorithm also extends to general planar graphs as shown in [2]. In VLSI wire routing it is often desirable to maximize the distance between different wires. Maximizing the distance between wires is equivalent to finding the drawing in which the edges are drawn as thick as possible, i.e., allowing the graph to grow as fat as possible.

The continuous homotopic routing problem [1, 3, 5] is a classic VLSI problem. The input is an initial sketch of the wiring (i.e., each wire is given a specified homotopy class), where the wires have fixed terminals. The goal is to route the wires with maximal separation between the wires, while preserving the homotopy. It is easy to see that this problem can be rephrased as the following fat-graph problem: find a planar drawing in which all the edges are drawn as fat as possible. Note that if the wiring sketch is not given or the terminals are not fixed, the problem is NP-hard [6].

The video shows the main steps of our algorithm: (1) an arbitrary wire routing is converted to a homotopic equivalent routing such that the distance between any two wires is maximized and the routing uses the wire lengths are minimized; (2) the wires are let to grow and (3) after the growth process terminates, the medial axis of each fat edge is used to route the corresponding wire. The running time of our algorithm is $O(kn + n^3)$ and the space required is $O(k + n)$ where n is the number of wires and k is the maximum of the input and output complexities. Note that k can be much larger than n. Even after shortest paths have been computed for each wire, k can be as large as $k = \Omega(2^n)$ [2].

We are now ready to outline the general technique for finding the continuous homotopic routing of maximum separation. We first define the concept of homotopic routing.

SoCG'02, June 5-7, 2002, Barcelona, Spain.
ACM 1-58113-504-1/02/0006.

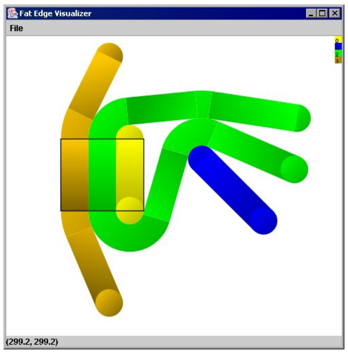

Figure 1: An example with straights and elbows.

Let $p, q : [0, 1] \to \mathbb{R}^2$ 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. $h(\lambda, 0) = p(0) = q(0)$, $h(\lambda, 1) = p(1) = q(1)$, $0 \le \lambda \le 1$
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 no two paths in P intersect and p_i is homotopic to w_i with respect to $V, 1 \le i \le n$.

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 is done in $O(nk)$ time using the algorithms of Hershberger and Snoeyink [4]. 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, Fig. 1.

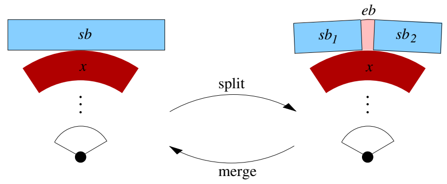

Figure 2: Split and merge events. The split event splits the straight bundle into a straight-elbow-straight bundle sequence. The merge operation is the reverse.

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, for an example, Fig. 1. We take advantage of this by grouping such parallel wires into *bundles* and maintain *straight bundles* and *elbow bundles* instead of single straights and elbows in our compact routing structure. In [2] we prove that the number of bundles stored in this data structure is only $O(n)$ and 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.

In order to fatten the edges, we keep track of collision events. 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 grows 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*, see Fig. 2 and Fig. 3. We construct a priority queue of events, with the key being the time at which the events occur. For increasing time we update the data 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 $O(n^2)$ time. It can be shown that the total number of events that the structure goes through from beginning to end is $O(n^2)$ and each event can be processed in $O(n)$ time yielding $O(n^3)$ time overall.

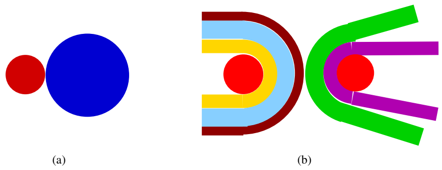

Figure 3: The two types of stopping condition: (a) collision of two vertices (b) collision of two elbows.

2. THE VIDEO

The video shows examples of graph growth, illustrating merge, split and stopping events and showing how bundles form and break apart, Fig. 1 and Fig. 4. The implementation has three distinct steps. Initial graphs are drawn in the Unix Fig format using the xfig tool. The second step is to compute the homotopic shortest paths which are then stored in a new fig file. The third step is the fat edge visualizer, written in Java 1.2 using the Swing graphics library and the Java2D graphics package. This visualizer reads the output of the shortest path algorithm and constructs an internal model of the graph. The graph model changes over time as merge, split and stop events occur, and these events are reflected on the screen. The visualizer also illustrates the presence of bundles in the graph. Implementation was done using a 1GHz Intel Celeron notebook with Java v1.2.

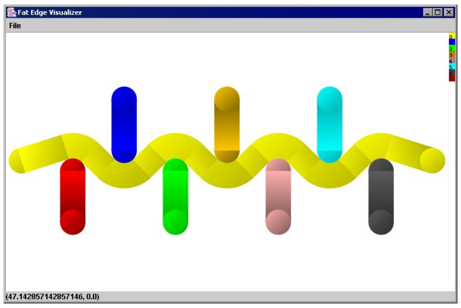

Figure 4: An example from the visualizer.

3. REFERENCES

[1] 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.

[2] C. A. Duncan, A. Efrat, S. G. Kobourov, and C. Wenk. Drawing with fat edges. In *9th Symposium on Graph Drawing (GD'01)*, pages 162–177, September 2001.

[3] 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.

[4] Hershberger and Snoeyink. Computing minimum length paths of a given homotopy class. *CGTA: Computational Geometry: Theory and Applications*, 4, 1994.

[5] C. E. Leiserson and F. M. Maley. Algorithms for routing and testing routability of planar VLSI layouts. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 69–78, 1985.

[6] C. E. Leiserson and R. Y. Pinter. Optimal placement for river routing. *SIAM Journal on Computing*, 12(3):447–462, Aug. 1983.