Curvilinear detailed routing algorithm and its extension to wire-spreading and wire-fattening

Toshiyuki Hama

IBM Research, Tokyo Reseach Laboratory
IBM Japan, Ltd.
Tel: +81-462-73-4669, Fax: +81-462-73-7413
e-mail: hama@jp.ibm.com

Hiroaki Etoh

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
Tel: +81-462-73-5066, Fax: +81-462-74-4282
e-mail: etoh@jp.ibm.com

Abstract— This article describes an algorithm for curvilinear detailed routing. We significantly improved the average time performance of Gao's algorithm by resolving its bottleneck related to generation of fan-shaped forbidden regions along a wire. We also describe a method for simultaneous wire-spreading and wire-fattening, which consists of enlarging forbidden regions generated by the detailed routing algorithm as long as there remains any space through which wires can pass. From the experiments we obtained the result that the average CPU time of the detailed routing algorithm is almost linear to the length of a wire. Since the curvilinear detailed routing is efficient in terms of space usage, the proposed algorithm is important especially for densely wired printed circuit boards such as PGA packages, BGA packages, and MCMs. We can also expect improvements on the electrical characteristics and the production yield by applying wire-spreading and wire-fattening to them.

I. Introduction

In this article, we describe an algorithm that transforms given topological wirings into curvilinear physical wirings while preserving their topology. Because of the growing need to design wiring patterns in limited routing spaces such as in PGA packages, BGA packages, and MCMs, we need to make the best possible use of available routing space. Apparently L_2 metric (Euclidean metric) is preferable in terms of space usage to any polygonal wiring rules such as rectilinear wiring and octilinear wiring. This motivated us to develop a practically feasible curvilinear detailed routing algorithm.

The curvilinear detailed routing algorithm has not been deeply studied so far. Only Gao [1] gives an $O(n^3 \log n)$ algorithm. The most time-consuming step of the algorithm consists of preprocessing for generating fan-shaped forbidden regions on every feature. We reduced the number of forbidden regions that need to be generated in order to obtain the same result, and significantly improved the average time performance of the algorithm. Another contribution of this article is a method for simultaneous wire-spreading and wire-fattening, which is achieved by appropriately enlarging forbidden regions generated by the detailed routing algorithm.

In the following sections, we first give a brief overview of detailed routing algorithms. Next, we give an improved algorithm for curvilinear detailed routing, based on constrained Delaunay triangulation. As an extension of the algorithm, we then describe a method for simultaneous wire-spreading and wire-fattening. Finally we discuss results of the experiments on several benchmarks.

II. Detailed routing problem

Separation of topological routing and physical routing began with Leiserson and Maley's pioneering work [2]. We define the detailed routing problem as follows:

> Given a distribution of features (terminals and inhibited regions) and paths of wires between them, the detailed routing problem is to transform the paths into physical paths (shortest physical paths in most cases) of wires meeting the underlying design rules while preserving the topology of given wires.

We assume that the given wires do not intersect and that they are disjoint from all the inhibited regions. Leiserson and Maley introduced the rubber-band equivalent (RBE), a shortest polygonal path homotopic to a given path, as a canonical form of topological path representation. They gave an algorithm for constructing RBEs from given paths of wires and also an algorithm for transforming the RBEs into physical paths of wires meeting the design rules in the L_∞ metric (rectilinear wiring). The detailed routing is generally achieved in two steps. The algorithm generates forbidden regions on every feature and then, using these regions as barriers, it constructs a physical wire as a shortest path between the regions. Since the shape of the forbidden regions depends on the underlying wiring rule, details of the algorithm also differ under different wiring rules.

Maley [3] extended the algorithm to the arbitrary polygonal wiring rule. Although it can conceptually generate curvilinear wirings in a limiting case, it is not a working curvilinear detailed routing algorithm. The algorithm generates spokes, which represent forbidden regions in the polygonal wiring rule, from all the features, and constructs a shortest polygonal path between spokes. Maley adopted a plane-sweep method for generating spokes. On the other hand, Dai's algorithm [4] traces each wire and generates spokes only where they are nec-

essary. A plane-sweep method is generally fast, but it sometimes generates a lot of unnecessary spokes, because spokes far from an RBE rarely contribute to the final shape of the shortest polygonal path. Therefore, although Maley's algorithm is fast in the worst case, Dai's algorithm sometimes outperforms Maley's in practical applications.

While the detailed routing algorithm under polygonal wiring rules has been studied by several authors [4, 5], Gao's theoretical paper [1] is the only one in the literature that describes a curvilinear detailed routing algorithm. Gao formally defined a detailed routing problem as the continuous homotopic one-layer routing problem (CHRP), and gave an algorithm for constructing curvilinear physical wiring from RBEs in $O(n^3 \cdot \log n)$ time. Gao's algorithm is an extension of a plane sweep method to a universal cover space of the plane. In detailed routing under the L_2 metric, forbidden regions cannot be represented simply by spokes from a feature, as in the polygonal wiring rules. We need to deal with fan-shaped forbidden regions directly. After generating fan-shaped forbidden regions on all the features visible from an RBE, Gao's algorithm sweeps a line perpendicular to the RBE on the cover space of the routing region, and constructs a curvilinear shortest path between the forbidden regions by using the funnel method.

Our curvilinear detailed routing algorithm is an improvement on Gao's original algorithm. A dominant part of the original algorithm is the generation of fan-shaped forbidden regions on all the features. The radius of a fan-shaped region is determined by the underlying design rules and the number of wires that pass between a feature and an RBE. The number of wires also varies depending on the angle in which the feature is viewed. Therefore the number of forbidden regions the algorithm generates for each RBE is $O(n^2)$. However, we can observe that, even if we generate fan-shaped forbidden regions on all the features visible from a RBE, only a small number of them actually contribute to the final shape of a curvilinear shortest path. We can thus expect a similar improvement to the one that Dai made to Maley's algorithm under polygonal wiring rules.

We generate fan-shaped forbidden regions as we travel from features near an RBE to more distant features, and halt the generation when we come to a feature that is so far from the RBE that we can ignore a forbidden region on the feature. Although our algorithm does not improve the computational complexity of the original algorithm, it significantly improves the average time performance. In the following section we describe the algorithm in detail.

III. Curvilinear detailed routing algorithm

Given a set of topological wires, the algorithm transform each wire into a physical wire independently. The algorithm consists of forbidden region generation and curvilinear shortest path construction between the forbidden regions. In this section, first, we describe a data structure for representing the paths of topological wires. Next, we describe an algorithm for generating only necessary fan-shaped forbidden regions on features on both sides along a wire. Finally, we describe an algorithm for constructing a curvilinear wire as a shortest path between the forbidden regions.

A. Data structure

We assume that the boundaries of a routing region and all inhibited regions are polygons, and that all the terminals are circles. We divide routing regions into triangles by constrained Delaunay triangulation [6, 7], where the boundaries of a routing region and all inhibited regions are regarded as constrained edges, and all the terminals are regarded as points. That is, all the boundaries become edges of triangles and all the terminals become vertices of triangles. The topological path of a wire is represented as a sequence of points at which it crosses triangle edges, as shown in Figure 1. If we avoid detours, this representation uniquely defines a path homotopic to a given RBE.

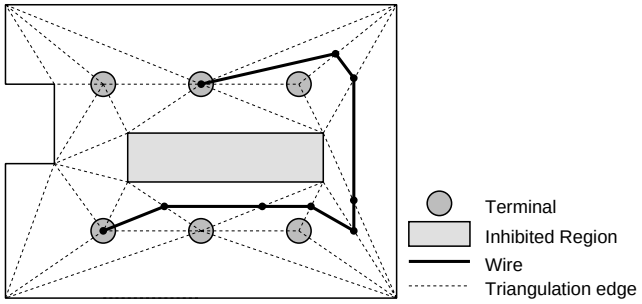


Fig. 1. Topological path representation

B. Generation of forbidden regions

Given a wire to transform, the algorithm generate forbidden regions on features on both sides along a wire, traversing triangles from ones which the wire passes. The followings are a brief outline of forbidden region generation procedure at each starting triangle.

Forbidden region generation: (*triangleABC*, *wireW*)
begin
  // we assume wire W crosses edge AB, AC
  Generates forbidden regions on vertices A, B, C;
  Estimates a range of areas wire W may pass;
  // traverse adjacent triangles beyond edge BC
  Traverse(*triangleBCD*)
where
proc *Traverse*(*triangleBCD*) ≡
  if *GenerationTest*(*D*)
    then generate forbidden region on vertex D; **fi**;
  if *HaltingTest*(*BD*)
    then *Traverse*(*triangleBDE*); **fi**;
  if *HaltingTest*(*CD*)
    then *Traverse*(*triangleCDF*); **fi**; .
end

Estimation of a range of areas the wire may pass is necessary for *GenerationTest()* and *HaltingTest()*. We estimate the range without any assumption on the distribution of features and wires outside the triangle. In Figure 2, let a black wire be transformed. The lower limit of the range is apparently the boundary of the forbidden region on the lowest vertex A. However, the upper limit of the range is not so simple, because forbidden regions on a feature beyond edge AB and AC (P for example) and wires around the feature may push up the wire. Recalling that the triangle is constructed by constrained Delaunay triangulation, we know that there are no other features, which is visible from the triangle, in the circumcircle of the triangle. The figure shows two cases in which the wire is pushed up most. Feature P lies on the circumcircle of the triangle ABC and all the wires between A and B flows between P and B. In (a) cut AP and BP are full of wires, and in (b) cut BP and CP are full of wires. The position of such feature P and the flow of wires around P can be computed easily. Then upper-limit in this case is also computed. Considering the counterpart of the feature P beyond edge AC, the upper-limit of the range is defined as a union of the upper limits pushed up by a feature from beyond edge AB and AC.

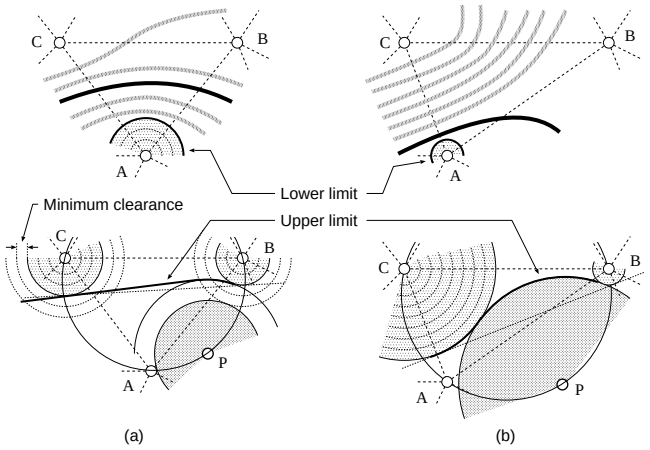


Fig. 2. Two types of upper limits of the route of a wire

The algorithm now travels from the upper adjacent triangle (BCD in Figure 3) to more distant triangles in order to generate the necessary forbidden regions in the recursive procedure *Traverse()*.

GenerationTest()

The algorithm checks whether a forbidden region should be generated on the most distant vertex (D) of the triangle. In the recursive call of the Traverse(), the algorithm accumulates the number of wires that flow between the underlying wire and the vertex D. The radius of the forbidden region is determined by the number of wires and clearances between wires. If the forbidden region overlap with the range of the wire defined above, the forbidden region is generated.

HaltingTest()

The algorithm then decides whether it should go farther to the adjacent triangle beyond the edge of the triangle. First of all, if the edge is invisible from the starting edge or it is a boundary of the routing region, this test fails. Then it checks the possibility that beyond the edge there is a feature whose forbidden region reaches the upper limit of the range of the wire. Figure 3 shows the positions of a feature (Q) beyond edge BD whose forbidden region most possibly reaches the upper limit. In (a) all the wire between B and D flows between B and Q, and in (b) flows between D and Q. In both cases feature Q lies on the circumcircle of triangle BCD, and cut BQ and DQ or cut BQ and CQ are full of wires. Thus the position of Q and the flow of wires around Q can be computed. Then the radius of the forbidden region on feature Q is determined. If none of the forbidden regions in the four cases overlap with the range of the wire defined above, this test fails.

In the procedure Traverse(), the algorithm collects generated forbidden regions in a list. Hence the generated forbidden regions are naturally sorted in the list from right to left.

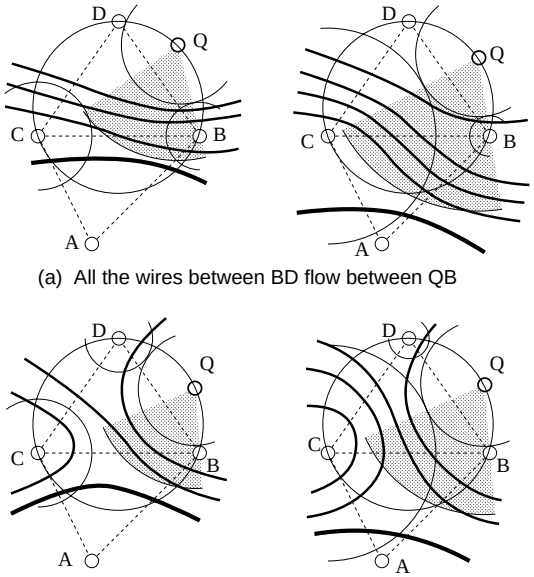


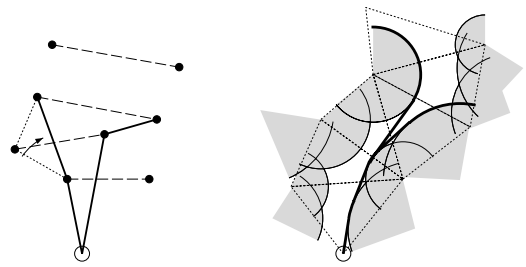
Fig. 3. Feature positions used for halting test

C. Construction of a curvilinear path

The procedure explained above is iterated along a sequence of triangles through which a wire passes. We now have two sequences of forbidden regions; sequences of forbidden regions on the right and left side of the wire. The final shape of the wire is given by the curvilinear shortest path between the two sequences of forbidden regions. We construct a curvilinear shortest path incrementally by tracing the two sequences, instead of using a plane-sweep method as in Gao's algorithm.

Our algorithm for constructing a curvilinear path is a modified version of Maley's Algorithm W [3], which is an algorithm for constructing the shortest polygonal path (rubber-

band) through a corridor consisting of a sequence of line segments (broken line segments in Figure 4 (a)). As it is tracing the line segments, Algorithm W maintains two sequences of points that represent the shortest paths from the starting point to both ends of the current line segment. When Algorithm W reaches the destination point, the two sequences of points coincide and give the shortest path along the corridor. Our algorithm constructs the curvilinear shortest path by maintaining two sequences of forbidden regions in a similar manner to Algorithm W as shown in Figure 4 (b).



(a) Maley's Algorithm W (b) Curvilinear path construction

Fig. 4. Path construction algorithm

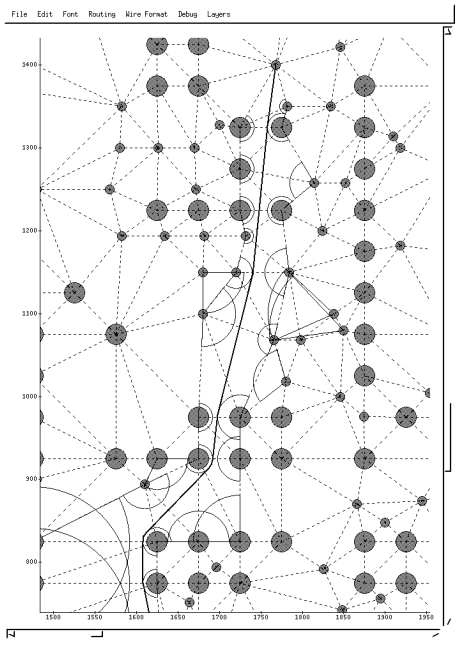


Fig. 5. Shortest path between forbidden regions

We need to modify Algorithm W to cope with the following differences between a corridor and two sequences of forbidden regions:

- While an end point of a line segment is a single point, a forbidden region is a part of a circle.
- While an end point of a line segment has a corresponding opposite end point, a forbidden region on one side does not always have a corresponding forbidden region on the other side.

Algorithm W traces a sequence of line segments, finds each end point of a line segment, and updates a corresponding sequence of points that it maintains. Since Algorithm W updates two sequences in turn, a new shortest path never intersects the shortest path on the other side of the wire. However, that is not the case with our problem. A new forbidden region may intersect a curvilinear shortest path on the other side of the wire. Unfortunately we cannot know the right order in advance without using a line-sweep method. Instead our algorithm undoes the construction of the shortest path on the other side until the new forbidden region does not intersect the shortest path. Then it tries the new forbidden region first and reconstructs the shortest path on the other side. In the worst case, our algorithm constructs a curvilinear shortest path in $O(n^2)$ time for n forbidden regions.

Figure 5 is a snap shot of the shortest path construction. Fan-shaped forbidden regions are generated on both near sides along a wire. Since we reduced the number of the forbidden regions by generating only necessary ones, the worst case complexity $O(n^2)$ is not a serious problem in terms of the overall performance of the curvilinear detailed routing algorithm.

IV. Simultaneous wire-spreading and wire-fattening

The curvilinear detailed router generates the shortest wires that meets the underlying design rules while preserving the topology of the original wires. Although it precisely keeps the minimum clearance between wires, that alone is not ideal in terms of productivity. Because the minimum clearance is usually defined in accordance with the limits of the current fabrication technology, and it does not always mean sufficiently wide clearance. Therefore wire-spreading and wire-fattening are important technologies for improving a yield of the production.

Our idea is very simple. We have already established a detailed routing algorithm for generating curvilinear wiring by constructing a shortest path between fan-shaped forbidden regions. If there remains more room for wires between features, we can enlarge the forbidden regions a little for each wire. The result of the detailed routing will then be curvilinear wiring with more clearance. As for wire-fattening, we regard a wire as a thin region surrounded by right- and left-side boundaries and transform each boundary independently into a curvilinear path with enlarged forbidden regions .

To obtain a curvilinear wire by using the algorithm described in section C, we have to satisfy the following conditions on enlarged forbidden regions:

- Forbidden regions on either side of a wire never intersect (Figure 6 a).
- Forbidden regions overlap with neither the starting point nor the ending point of a wire (Figure 6 b).

- Either corner of forbidden regions must not stick out of another forbidden region on the same side of a wire (Figure 6 c).

The first two conditions must be satisfied to ensure that there exists a path from the starting point to the ending point between forbidden regions. The last condition is imposed so as to avoid exception-handling in the algorithm, and to obtain a smooth wire. It should be noted that none of the undesirable situations described above occur provided that forbidden regions are not enlarged and the routability of the given wires is ensured.

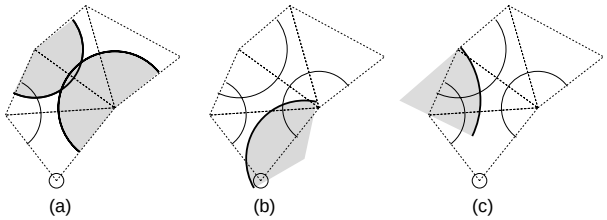


Fig. 6. Undesirable situations

We introduce two constants D_c and D_w ($D_c + D_w = 1$), which define the distribution of spare space between the clearance and the wire width. $D_c = 1$ ($D_w = 0$) means simple wire-spreading and $D_c = 0$ ($D_w = 1$) means simple wire-fattening. By adjusting the values, we can achieve wire-spreading and wire-fattening simultaneously. We also define two variables $\alpha_i$and β_i for each vertex i; α_i gives the ratio by which the clearance is enlarged and β_i gives a ratio by which the wire width is enlarged when we generate a forbidden region on the vertex i.

The first two conditions on forbidden regions can be satisfied by appropriately adjusting the values of α_i and β_i ($i = 1, \ldots, n$). Each critical cut from vertex i gives upper bounds for α_i and β_i. Let's say there is a critical cut CC_{ij} between vertex i and vertex j, the length of CC_{ij} is $length(CC_{ij})$, the radii of the features at vertex i and vertex j are R_i and R_j, the sum of the widths of wires crossing C_{ij} is W_{ij}, and the sum of the minimum clearances is C_{ij}. The spare space SS_{ij} needed to satisfy the first condition is given by

$$SS_{ij} = length(CC_{ij}) - R_i - R_j - W_{ij} - C_{ij}.$$

We can distribute the spare spaces SS_{ij} to clearances and wires so as to enlarge them. Therefore the upper bounds of α_i and β_i are given by

$$\begin{aligned} \alpha_i &\leq (D_c \cdot SS_{ij} + C_{ij})/C_{ij}, \\ \beta_i &\leq (D_w \cdot SS_{ij} + W_{ij})/W_{ij}. \end{aligned}$$

If a wire start from vertex j and its width is w_j, similarly the spare space SS'_{ij} for satisfying the second condition is given by

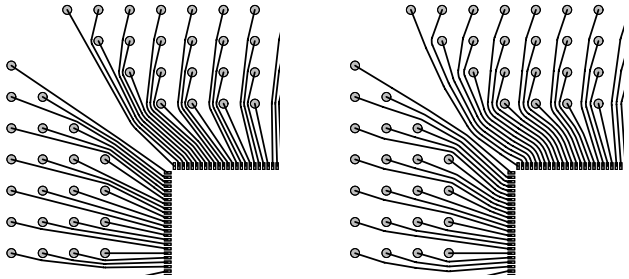
$$SS'_{ij} = length(CC_{ij}) - R_i - w_j/2 - W_{ij} - C_{ij}.$$

The upper bounds of α_i and β_i are given by

$$\begin{aligned} \alpha_i &\leq (D_c \cdot SS'_{ij} + C_{ij})/C_{ij}, \\ \beta_i &\leq (D_w \cdot SS'_{ij} + W_{ij} + w_j/2)/(W_{ij} + w_j/2). \end{aligned}$$

Since all the critical cuts from vertex i can be accessed by sweeping around a ray from vertex i, we can appropriately set all the values of α_i and β_i ($n = 1 \ldots n$) by iterating the same process from every vertex.

Although the third condition is also satisfied by making α_i and β_i smaller, we do not want to forego a better result just for the sake of making a wire smooth. Instead, we reduce the radius of a violating forbidden region so that either of its corners does not stick out of an arc of another forbidden region. It is clear that even if we shrink a violating forbidden region independently, we can ensure the minimum clearance defined in the design rules. Because the clearances between wires are always dominated by at least one of the enlarged forbidden regions, and never fall below the minimum clearance.



(a) Original wiring (b) Spreaded wiring

Fig. 7. Result of wire-spreading

Figure 7 shows a result of applying the wire-spreading technique to the detailed routing of a PGA package. The right figure is a result of the detailed routing under the original design rule. Wires turn around a pin with minimum clearance. The left figure is a result of the detailed routing with wire-spreading. Wires are spreaded depending on the local congestion of wires around a pin.

V. Evaluation

We have implemented the curvilinear detailed routing algorithm described in section III and the wire-spreading algorithm (in the case $D_c = 1$ and $D_w = 0$) in C++ on AIX 4.1. We evaluated the performance of the algorithm, using several benchmark PCBs and PGA packages.

The performance of the algorithm is dominated by the generation of fan-shaped forbidden regions. We have measured how many outer vertices the algorithm visits from each triangle during the forbidden region generation. Figure 8 shows histograms of the number of visited vertices and the number of actually generated forbidden regions for routing an inner layer of a 3D graphics board with more than 7000 terminals as a benchmark. We have also obtained histograms of a similar shape from other benchmark boards. For each triangle, the average number of the visited vertices was 3.79, and the average number of generated forbidden regions was 0.23. In more than

90% of the cases, the number of the visited vertices is less than 7. Thus, considering the benchmark board has more than 7000 terminals, we can conclude the depth of the traverse of outer triangles is empirically independent of the number of terminals in an ordinary printed circuit board. On the other hand, the forbidden region generation in Gao's algorithm depends on the number of terminals. Therefore a great performance improvement will be expected in a printed circuit board with a large number of terminals.

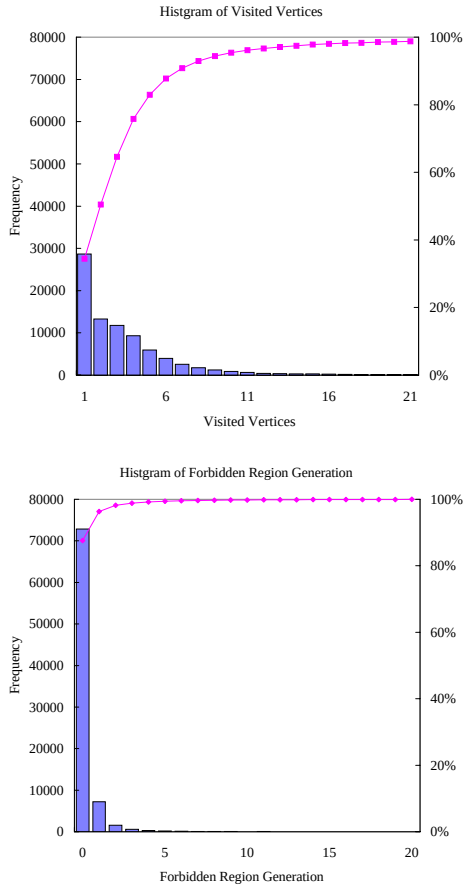


Fig. 8. Results of experiments

We also found only few forbidden regions were generated outside triangles through which a wire passes, which improves performance of the curvilinear shortest path construction. Figure 9 shows the average CPU time of the curvilinear detailed routing for each wire against its length measured by the number of triangles. Even though there is a little deviation, the average CPU time is almost linear to the length of a wire as is expected from the shapes of the histograms.

On the other hand, the wire-spreading algorithm has a bottleneck in adjusting factors to enlarge forbidden regions on all the features. Since the algorithm inherently needs to check all the critical cuts to obtain appropriate enlarging factors, even the average complexity is $O(n^2 \log n)$ for a printed circuit board with n features.

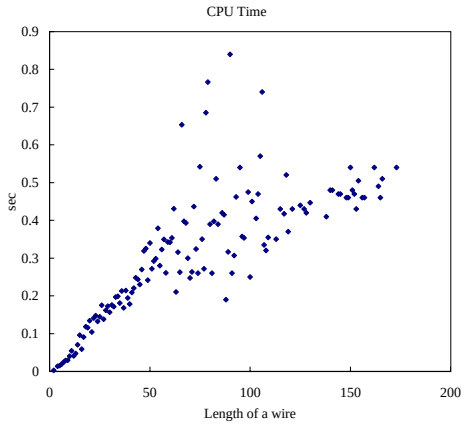


Fig. 9. Average CPU time for transforming wires

VI. Summary

We have described an algorithm for curvilinear detailed routing. We proposed a method for generating only those forbidden regions that are necessary to transform a wire correctly. The method improved the performance of generating forbidden regions by visiting only features near a wire, instead of sweeping all the features. The algorithm thus resolved the bottleneck of Gao's algorithm, and significantly improved the average time performance in an ordinary printed circuit board.

We also described an algorithm for simultaneous wire-spreading and wire-fattening, which consists of enlarging forbidden regions as long as there remains spaces for wires. We gave three conditions on the size of forbidden regions in order for the same detailed routing algorithm to be applied to enlarged forbidden regions. However, the performance problem still remains.

References

[1] 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 ACM Symposium on Computational Geometry*, pp. 392–402, ACM, 1988.

[2] 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*, pp. 69–78, ACM, 1985.

[3] F. M. Maley, *Single-Layer Wire Routing and Compaction*. MIT Press, 1990.

[4] W. W.-M. Dai, R. Kong, and M. Sato, "Routability of a rubber-band sketch," in *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pp. 45–48, 1991.

[5] F. M. Maley, "Testing homotopic routability under polygonal wiring rules," *Algorithmica*, vol. 15, pp. 1–16, 1996.

[6] L. P. Chew, "Constrained delaunay triangulations," *Algorithmica*, no. 4, pp. 97–108, 1989.

[7] Y. Lu and W. Dai, "A numerical stable algorithm for constructing constrained delaunay triangulation and application to multichip module layout," in *Proceedings of 1991 International Conference on Circuits and Systems*, pp. 644–647, June 1991.