Topological Routing Path Search Algorithm with Incremental Routability Test

Toshiyuki Hama

IBM Research, Tokyo Research Laboratory
IBM Japan Ltd.
1623-14, Shimoturuma, Yamato-shi,
Kanagawa-ken 242, Japan
Tel: +81-462-73-4669
Fax: +81-462-73-7413
e-mail: hama@trl.ibm.co.jp

Hiroaki Etoh

IBM Research, Tokyo Research Laboratory
IBM Japan Ltd.
1623-14, Shimoturuma, Yamato-shi,
Kanagawa-ken 242, Japan
Tel: +81-462-73-5066
Fax: +81-462-73-7413
e-mail: etoh@vnet.ibm.com

Abstract— This article describes a topological routing path search algorithm embedded in our auto-router for printed circuit boards. The algorithm searches for a topological path that is guaranteed to be transformable into a physical wire satisfying design rules. We propose a method for incrementally verifying design rules during topological path search in a graph based on constrained Delaunay triangulation, and describe several improvements to the routing path search algorithm that remedy the overhead of the routability test and avoid combinatorial explosion.

I. Introduction

We have been developing an auto-router specifically for a single-layer printed circuit board. The auto-router completes routing in two phases: a global routing phase and a detailed routing phase. In the global routing phase, the auto-router determines a topological wiring pattern and does not fix the physical positions of wires. In the detailed routing phase, the auto-router fixes the physical positions of wires to meet design rules. In this article we describe a topological routing path search algorithm used in the global routing phase. A notable feature of the algorithm is that it guarantees the routability of all topological paths found, using an incremental routability test.

In the following sections, we describe the basic data structure that we use to find a topological path in a routing region on a printed circuit board. We then explain an incremental routability test algorithm and a routing path search algorithm that includes it. We also describe a technique for decreasing the overhead of the incremental routability test embedded in the routing path search algorithm. Finally, we discuss a performance problem and give the results of experiments using several benchmark printed circuit boards.

II. Data structure

We constructed a basic data structure by using Constrained Delaunay Triangulation to partition routing regions into triangles [3], with each terminal regarded as a point and the boundaries of printed circuit board and inhibited regions as constrained edges. Since our auto-router fixes only the topological path of wires in the global routing phase, each wire is represented as a sequence of crossing points on triangulation edges that uniquely represents the topological path in the routing regions [6].

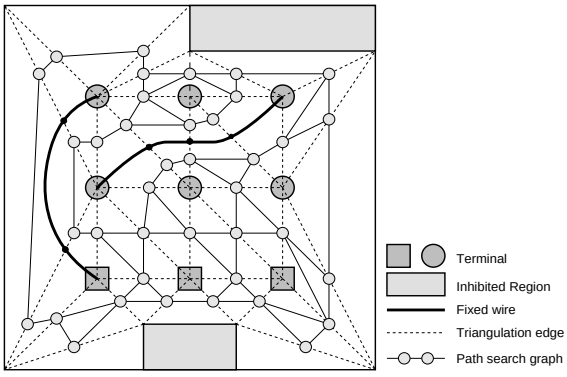

Fig. 1. Path search graph

The path search algorithm searches for a path that does not cross fixed wires. Since the path search algorithm, like ordinary shortest-path algorithms, works on a graph data structure, we construct a graph (path search graph) on which no path crosses fixed wires (see Figure 1). This path search graph is dependent on a fixed wiring pattern. Each time the auto-router fixes a new wire or rips up a fixed wire, the path search graph is updated.

ASP-DAC '97
0-89791-851-7$5.00

III. Incremental routability test

As we cannot be sure that a given topological wiring pattern can be transformed into a physical wiring pattern that satisfies design rules, we generally need to apply a routability test before we move on to the detailed routing phase. Since the pioneering work on routability test algorithms and rubber-band expression of wires by Leiserson and Maley [2], several improvements have been made to routability test algorithms [1, 4]. In the original paper, Leiserson and Maley proved that wires are routable if all the critical cuts are safe ("safe" means that the length of a critical cut is long enough to accommodate all the crossing wires). Dai and others then proved that wires are routable if all the wires are successfully transformed into extended rubber-band expressions. Recent improved routability test algorithms were developed on the basis of the latter proof.

These routability test algorithms are used as filters to prevent unroutable topological wiring patterns from being passed to the detailed routing phase. However, if a given topological wiring pattern is found to be unroutable, we need to find another wiring pattern and test its routability. The problem, we argue, is that the topological path search algorithm in the global routing does not recognize the routability of the current search path.

According to Leiserson and Maley's result, we can guarantee the routability of wires if we can ensure that all the critical cuts are safe. We detect crossing critical cuts and check their safety whenever the path search algorithm extends the current search path by one edge. In the following, we explain how to detect crossing critical cuts, given a path on a path search graph.

In a triangulated region, a critical cut connects a triangle vertex to another triangle vertex or to a triangle edge if the latter is a constrained edge. We call an union of triangle regions penetrated by a critical cut penetrates an enclosing region of the critical cut.

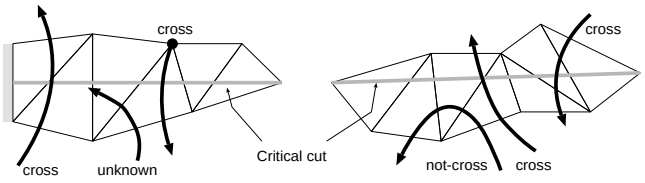

Fig. 2. Enclosing region and various paths

Given a path starting from a certain terminal, from the geometrical relation between the path and an enclosing region, we can tell whether the path crosses the critical cut, as shown in Figure 2.

cross: a path enters an enclosing region through a boundary of the region and leaves it through a boundary on the opposite side of the critical cut.

not-cross: a path enters an enclosing region through a boundary of the region and leaves it through a boundary on the same side of the critical cut.

unknown: a path enters an enclosing region and remains in the region.

Thus, when a search path comes to a boundary of an enclosing region of a certain critical cut, we activate the cut and remember from which side of the boundary the search path entered the region. Then, when the search path comes to a boundary of the same enclosing region again, we deactivate the cut and check whether the path actually crosses the cut by comparing the leaving side with the entering side. If we find that the path crosses the critical cut, we check the safety of the cut. Since a list of crossing wires is stored in each critical cut in the global routing phase, it is easy to check the safety. An incremental routability test can be realized by iterating these steps whenever the path search algorithm extends the current search path by one edge.

To determine the critical cut whose enclosing region a current search path is about to enter, we need another data structure. If a triangle edge or vertex belongs to the boundary of an enclosing region of a certain critical cut, we add the following three data to the edge or the node:

- A pointer to the critical cut
- The side of the boundary to which the edge or vertex belongs
- The direction of the path when it enters the region

IV. Path search algorithm

In ordinary auto-routers, Dijkstra's shortest-path algorithm is used to find a path of a net. However, we cannot directly apply Dijkstra's algorithm to the routing path search with the incremental routability test. As we explained in the previous section, it is unknown whether the current path is routable until it leaves the enclosing region of the cut. What is worse, a path sometimes turns out to be unroutable after it has crossed the same cut more than once.

Figure 3 shows an example in which Dijkstra's algorithm does not work correctly. In this example, we assume that capacity of a critical cut is only three wires and no more wires can pass through the cut. Let us trace Dijkstra's algorithm for this example. A path from vertex a arrives at vertex c and is found to be shortest, so the distance and the pointer to the previous vertex (vertex a) are updated at vertex c. Then, a path from vertex b arrives at vertex c, but the path is not shortest, so this path is given up here. The shortest path to vertex c is determined. A path a c then arrives at vertex d and is found to cross a critical cut. Since the cut becomes unsafe, not only the path a c d but also the data on vertex

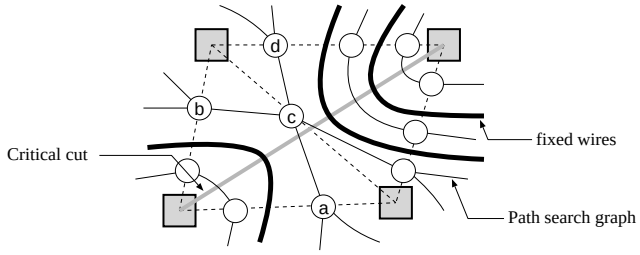

Fig. 3. Difficult case for Dijkstra's algorithm

c become invalid. However, the distance and the pointer to the previous vertex cannot be restored at the vertex c in Dijkstra's algorithm.

We therefore need to distinguish paths that enter an enclosing region of a certain critical cut from a different side of the cut, and also paths that cross different set of critical cuts. To do so, we duplicate the vertex of a path search graph and add a set of critical cuts as a label to the vertex if a vertex with the same label has not yet been created, when the current wave front of Dijkstra's scan reaches the vertex. The set of critical cuts consists of the following critical cuts:

- Critical cuts whose enclosing region the current path entered and stayed in, and the side of the region's boundary through which the path entered.
- Critical cuts crossed by the current path crossed

Thus, we augment a path search graph dynamically by creating a vertex labeled with a set of critical cuts and apply Dijkstra's shortest-path algorithm to the augmented graph.

When this modified algorithm is used in practical applications, the size of an augmented graph is our main concern. In the worst case, the size of an augmented graph increases exponentially because the number of different labels of the same vertex is proportional to the number of combinations of crossing critical cuts. To keep the variation as small as possible, labels of the same vertex are checked for an inclusion relation when a new labeled vertex is created. If a label (a set of critical cuts) for path A includes a label for path B, and path A is not shorter than path B, we need not create a new labeled vertex for path A.

V. Performance improvement

The performance of the path search algorithm with the incremental routability test depends on the number of critical cuts, in two senses:

- Growth of an augmented path search graph caused by combinatorial explosion of crossing critical cuts.
- Overhead of detecting critical cuts crossed by the current path. crosses.

The number of critical cuts for a printed circuit board with N terminals is $O(\mathrm{N}^2)$. Thus, simple implementation of the algorithm will be vulnerable to combinatorial explosion.

We have categorized the critical cuts into three groups, and identified those that we actually need to check during routing path search. The groups are as follows:

- Critical cuts that are always safe for any wiring pattern
- Critical cuts that are always safe for the current wiring pattern
- Critical cuts that are literally critical

The first group is identified before the global routing and is ignored during routing path search. Since other groups depend on topologically fixed wiring patterns, the classification of all critical cuts is updated whenever a new wire is fixed or a fixed wire is ripped up. In the following, we explain the classification method.

After triangulation of the routing regions, we estimate the upper bound of the flow of wires for each critical cut. First, we set initial value of the maximum flow for a triangle edge. If the edge is a constrained edge, the maximum flow is zero (Figure 4(a)). If an edge of a triangle is a constrained edge and there is a critical cut between the edge and the opposite vertex, the maximum flow of the critical cut is its capacity, and the maximum flows of the other edges are identical to the capacity of the critical cut (Figure 4(b)). Otherwise the maximum flow of a triangle edge is its capacity.

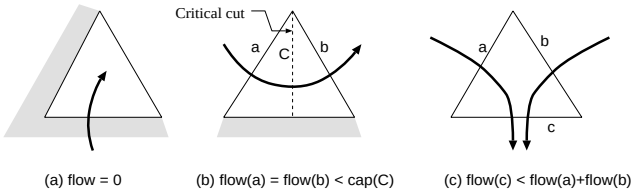

Fig. 4. Flow of wires of triangle edge

Next, by applying the following rule iteratively, we update the maximum flow of all the triangle edges. To simplify the explanation, we ignore the flow of a wire from a triangle vertex.

For any triangle abc, we can bound the maximum flow of edge c by the inequality (Figure 4(c)),

$$\mathrm{Maxflow}(c) \leq \mathrm{MaxFlow}(a) + \mathrm{MaxFlow(b)}.$$

The maximum flows of all the triangle edges are now determined. Next, applying a similar rule, we can bound the maximum flow of each critical cut.

For any critical cut C the right boundary of whose enclosing region consists of triangle edges r_1, r_2, ..., r_n, and the left boundary of whose enclosing region consists of triangle edges l_1, l_2, ..., l_m,

$$\text{Maxflow}(C) \leq \text{Min}\{\sum_{i=1}^{n} \text{Maxflow}(r_i),\ \sum_{i=1}^{m} \text{Maxflow}(l_i)\}$$

If the capacity of a critical cut is greater than its estimated maximum flow, the critical cut never becomes unsafe for any wiring pattern. Therefore we categorize such critical cuts as the first group. In an experiment using several benchmark printed circuit boards, 70%-80% of the critical cuts on average were categorized into the first group.

Among the other critical cuts, we identify the second group by using the following very simple test. If N wires cross a critical cut in the current wiring pattern, a new path never crosses the critical cut more than N+1 times. Therefore, if the capacity of the critical cut is greater than the flow of 2N+1 wires, the critical cut is always safe, at least for the current wiring pattern.

VI. Discussion

The topological routing path search algorithm with an incremental routability test is vulnerable to combinatorial explosion. However, our concern is the applicability of the algorithm to printed circuit boards of a practical size. We have described two techniques for preventing an explosion in the size of an augmented path search graph:

- Reducing the number of labels of a vertex by checking for inclusion between labels.
- Reducing the number of critical cuts by estimating the upper bound of a flow of wires.

We conducted experiments on the growth of the augmented graph for several benchmark printed circuit boards. In these experiments, we measured the number of vertices in the original path search graph, the number of labeled vertices generated during a search, and the number of critical cuts needing to be checked.

Board	Terminals	Vertices	New vertices	Cuts
Sample 1	363	1662	3175	54
Sample 2	756	3958	98373	185
Sample 3	1203	4215	221089	20
Sample 3'	1203	4215	5470	20

Fig. 5. growth of augmented graph

The results in Figure 5 shows the worst case for each benchmark board. For the first three samples, inclusion relations between labels of vertices were not taken into consideration. With sample 3, we encountered an actual combinatorial explosion. The result for the last sample was obtained by using label size reduction in the path search algorithm. This result shows that it is really possible to reduce the graph size by using inclusion relations between labels, and that we were able to avoid combinatorial explosion for a benchmark board of a practically large size. Furthermore, the reduction in the number of critical cuts to be checked is remarkably small, considering the number of terminals of in the benchmark boards.

According to our observations, only a small number of wires are generally fixed at the beginning of the global routing, and the number of literally critical cuts is also small. The path search algorithm rarely encounters a combinatorial explosion. The possibility of combinatorial explosion increases as the global routing proceeds. At the end of the global routing phase, on the other hand, fixed wires partition the routing region into small areas, and reachable vertices on a path search graph are confined to a small portion of the graph. Thus, the possibility of combinatorial explosion decreases toward the end of the global routing, and is relatively high in the middle of the global routing phase.

VII. Summary

We have described a topological routing path search algorithm with an incremental routability test. The algorithm finds a topological path that is guaranteed to be transformable into a physical wire that meets design rules. Although the algorithm is inherently vulnerable to combinatorial explosion, we also described two techniques for avoiding this problem and showed that the algorithm is applicable to printed circuit boards of a practical size.

References

[1] 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*, pages 45–48, 1991.

[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*, pages 69–78. ACM, 1985.

[3] 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*, pages 644–647, June 1991.

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

[5] D. Staepelaere, J. Jue, T. Dayan, and W. W.-M. Dai. "Surf: A rubber-band routing system for multichip modules," *IEEE Design & Test of Computers*, 10(4):18–26, 1993.

[6] H. Tanaka, M. Kanazawa, H. Tanaka, M. Satoh, and T. Ohtuki. "A multi-layer routing system based on sketch model (in Jananese)," *IPSJ SIG Notes DA 70-9*, pages 63–70, 1994.