# Area Routing Oriented Hierarchical Corner Stitching with Partial Bin[†]

Zhang Yan  Wang Baohua  Cai Yici  Hong Xianlong

Department of Computer Science and Technology

Tsinghua University

Beijing 100084, P.R. China

Tel: +86-10-62785564

e-mail: {zhangy, wangbh, caiyc, hong}@tiger.cs.tsinghua.edu.cn

**Abstract—An efficient layout data structure is of great importance to a gridless area routing algorithm. Nowadays, the rectangular corner stitching structure, whose point searching and tile insertion are $O(N^{1/2})$, is the most popular data structure used by gridless area routers. In this paper, we discuss a novel database—hierarchical corner stitching with partial bin, or hierarchical PB corner stitching, which combined the bin-based structure and the trapezoidal corner stitching. Its point searching and tile insertion are both enhanced to $O(N^{1/2}/r)$. We also derive the algorithms of the operations, such as point searching, area searching, plowing, etc., according to the specialties of area routers.**

## I. INTRODUCTION

With the development of semiconductor technology, designers' demands for area routers improve. The Top 10 Problems in Physical Design, which was put forward by SRC in 1999[14], include RLC routing. SRC mentioned that multi-layer general area routing is also among the important physical design problems.

The problem of area routing can be classified in terms of the representation of the area and whether there exists constraint of the position of route. Grid based routers[1][2][3][4][5] use grid graphs to symbolize the routing area and the positions of paths are confined to the grids. Gridless routing directly considers the geometrical patterns, and has no additional constraints for the paths except for connectivity and design rules. Gridless routing now attracts more concerns for these advantages: it reduces the memory requirements by using line based or tile based data structures, thus could handle a larger problem size; it enjoys more flexibility in accommodating complicated design rules; and it could help to reach a higher connection rate. Hence, our target is a general gridless area router with high connection rate and high performance. Our area router would handle 45° paths in respect that introducing 45° routing would help to save routing resources, to improve connection rate, and to optimize the routing results.

Most of the proposed gridless routers are "gridless maze routers", which means an incremental routing process, like the Lee algorithm and the line search, in which not only the path-finding process but also the postprocessing of updating geometrical information for subsequent interconnection occupies a major portion of total processing time for interconnecting nets one after another. Therefore, a fast and convenient layout data structure is of great importance to a gridless area router.

In this paper, we propose a new area routing oriented hierarchical structure, which combines the bin-based structure and trapezoidal corner stitching. Also, we renewed some operations for area routing. The rest of the paper is organized as follows: Section II introduces some existing layout data structures. Section III describes the proposed hierarchical PB corner stitching structure and the algorithms of the operations, and compares the complexity of layout data structures. Section IV shows experimental results. Section V concludes the paper.

## II. EXISTING DATA STRUCTURES

The simplest structure for representing figures is to keep all of them in a linked list. A linked list is easy to implement but does not support efficient operations. A bin-based structure locates figures according to imaginary divided bins, thus enhances the speed of operations. A K-D tree[6] or a multi-dimensional binary tree work efficiently when searching for layout figures at a point or in a rectangular area, assuming the tree is balanced. However, K-D trees have some disadvantages: neighbor searching still requires a O(logN) algorithm, where N is the total number of layout figures in the area.. Also, unless a lot of additional overhead is incurred during insert and delete operations, a K-D tree often becomes unbalanced and begins to resemble a linked list structure. Like a K-D tree, the point searching, area searching, insertion, and deletion operations of a quad tree[7][8] are quick when the tree is balanced. Because quad trees are generally not as deep as K-D trees and tend to stay more balanced, quad trees work better in practice than K-D trees.

The above structures have a common drawback: they provide no assistance in locating empty space for routing, since only the occupied spaces are represented explicitly. Corner stitching[9] and trapezoidal corner stitching[13] arose from a consideration of this weakness. During the process of area routing, a path from the source point to the target point should be searched in the empty spaces, and when a path could not be directly found, the router might push aside some

[†] This project is supported by 973 National Key project G1998030413 on Foundational Research.

nearby existing nets so as to make room for the blocked net, so that a layout representing structure with management of empty space and fast neighboring operations would enhance the total performance of a routing algorithm. These two demands are just two notable virtues of corner stitching. Therefore, corner stitching is probably the most suitable layout database for a gridless area router, so it's no wonder most of the published gridless area routers[10][11][12] are based on it. However, we haven't found any published area routers that use trapezoidal corner stitching as layout database.

## III. Hierarchical PB Corner Stitching

Our layout data structure is designed according to the demands of the routing algorithm. From Section II, we notice that corner stitching is the most suitable layout database for a gridless area router, but it still has some deficiencies. Firstly, the speed of point searching should be improved. Point searching is pivotal to the efficiency of the structure, since it would be called during several frequently used operations, such as area searching and insertion. Unfortunately, point searching is $O(N^{1/2})$ for corner stitching, while for 4-D tree and quad tree, point searching is only O(logN). Next, the data structure should be able to handle figures with 45° edges. Finally, the structure should represent multilayers.

### *A. Hierarchical Structure*

In corner stitching, every tile points to its neighboring tiles, which means that the structure keeps plenty of inter-tile information, or local neighboring information. But those structures with high-speed point searching, such as quad tree, 4-D tree and bin-based structure, all contain each tile's global position information, which indicates the relationship between individual tile and the whole area, for example, which quadrant does a tile belongs to, what is the tile's relative position to the median tile, or which bin does the tile cover. It's just because of lacking such global position information that point searching is inefficient for traditional corner stitching.

To remedy this, we combined corner stitching with the bin-based structure and came out with a new hierarchical PB corner stitching that contains both global position information and local neighboring information of tiles. The new structure has merits of both structures: fast point searching and high efficiency of neighboring operations. The structure consists of two layers: corner stitching layer and bin layer, as shown in Fig. 5.

### *A.1 Corner Stitching Layer*

The structure at this layer is a routing-oriented trapezoidal corner stitching. Our router divides a multi-layer problem into 2-layer or 3-layer routing problems. We use different planes to store the different routing layers. For horizontal(H) and vertical(V) planes, the definitions of the structure have some slight differences. Both horizontal(H) corner stitching and vertical(V) corner stitching have 9 different tile shapes. For H planes, all tiles are trapezoids whose top and bottom edges are parallel with the x-axis and whose left and right edges have an angle of 45°, 90°, or 135° with the x-axis, as shown in Fig.1. Fig.2 shows an example of horizontal trapezoidal corner stitching in a layout plane. For V planes, the left and right edges of all tiles are vertical to the x-axis, and the top and bottom edges have an angle of 45°, 90°, or 135° with the x-axis. Also, at V planes, the segmentation strategy for the empty space is just the opposite of the H planes, each tile is constructed as a vertical trapezoidal strip which is as tall as possible, then as wide as possible. The maximal H (V) strip tiles on the H(V) tile plane allow fast tile expansion in the H(V) direction. We mirror the coordinates of a V tile plane at 45 degrees, as shown in Fig.3, to unify the implementation of H planes and of V planes. In the rest of this paper, we only discuss horizontal corner stitching, from which the condition of vertical corner stitching can be easily derived.

The corner stitching layers of the hierarchical structure maintain similar properties to traditional corner stitching. Every point in a tile plane is covered by one and only one tile. Each tile contains its left and bottom edges and never its right and top edges.

A tile structure in corner stitching layer contains:

- the x and y coordinate of the tile's lower left corner.
- four pointers: tr, rt, bl and lb.
- the tile shape.
- the tile contents, includes whether it's solid or empty; the type(cell/pin/net) of the tile if tile is solid; the net number if the tile is a net.
- the modification plan of the tile. This field is used in the plowing operation.

Corner stitches are defined as follows:

- tr and bl stitches are the same with the definitions in [9]. tr stitch points to the top most neighboring tile on the right and bl stitch points to the bottom most neighboring tile on the left.
- Considering the conditions of top or bottom edges becoming a point, the rt and lb stitches are modified. rt stitch points to the right most tile whose bottom edge contains the

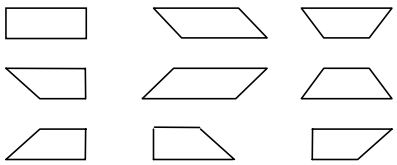

Fig. 1. Nine possible tile shapes of horizontal trapezoidal corner stitching

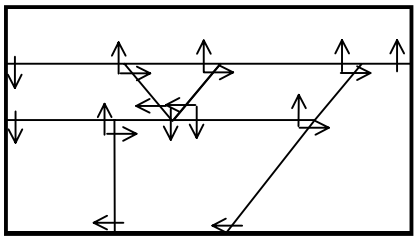

Fig. 2. Example of trapezoidal corner stitchingin a layout plane.

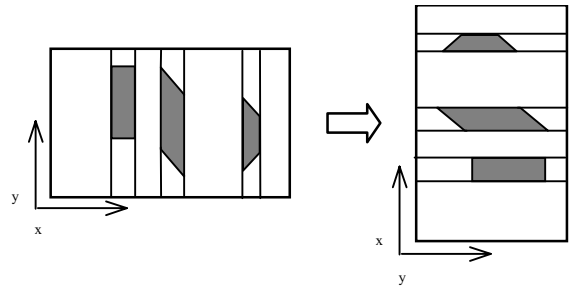


Fig. 3. Transformation of V plane.

upper right corner. lb stitch points to the left most tile whose top edge contains the lower left corner. Fig. 4 shows the modifications of rt and lb.

*A.2 Bin Layer*

The size and shape of the bin layer are the same with the corner stitching layer. All tiles in the corner stitching layer are projected to the bin layer, as shown in Fig. 5. An imaginary square grid divides the area into m×n bins, which are managed by a two-dimensional array. The bins are indexed by its position at x and y directions, such as Bin(1,3) or Bin(5,2). Given the coordinates of a point and the size of bins, we can easily determine the indexes of the bin that contains the point.

If we directly add the bin-based structure onto the corner stitching layer, which means that all of the tiles intersecting a particular bin are linked together and stored in that bin, then too much data redundancy would occur. So we simplify the bin layer by cutting out the superfluous, and get the new hierarchical PB corner stitching structure (Hierarchical Corner Stitch with Partial Bin), as shown in Fig. 5. A tile is only kept in the bin where the lower left corner of the tile's projection locates. In Fig. 5, the solid tile's projection on the bin layer covers six bins: Bin(3,1), Bin(3,2), Bin(4,1), Bin(4,2), Bin(5,1) and Bin(5,2)，but only one of them—Bin(3,1) keeps a pointer to the tile. Also, when a bin covers more than one projection's lower left corner, as Bin(1,1) in the figure, only one tile is kept in the bin. This idea is based on the fact that corner stitching is efficient at neighbor searching. When performing point searching, first a tile near the given point is obtained from the bin layer, and then that tile would be used as a starting tile to search in the corner stitching layer. Since the searching in the corner stitching layer is started near the target tile, the scope of the searching is limited; thus the searching time would not be decreased.

In hierarchical PB corner stitching, bin layer keeps the global position information of some tiles, which greatly improves the speed of obtaining tiles from coordinates, namely point searching. Besides, the additional overhead incurred to manage the bin layer during insert and delete operations is little. Our bin layer is different from the traditional bin based structure, which would incur lots of inefficient linked list operations when there are many tiles in one bin. In PB corner stitching, each bin keeps no more than one pointer, and the searching within a bin is via point searching in corner stitching layer, which is faster than searching in linked lists. With the increasing of tile number, insertion of PB corner stitching would be even faster than that of corner stitching, since the time saved from point searching exceeds that used in managing the bin layer. Compared to traditional corner stitching, the memory requirement of PB corner stitching structure increases $r^2$, while $r^2$ is the number of bins.

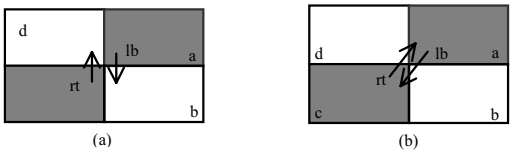


Fig. 4. The changes of rt and lb stitches.

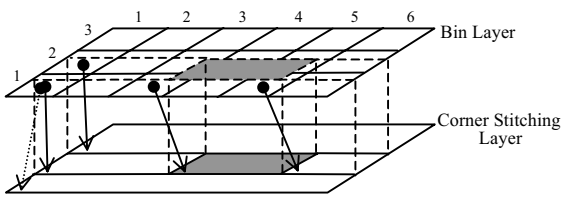


Fig. 5. The configuration of hierarchical PB corner stitching.

*B. Algorithms*

The operations of hierarchical PB corner stitching could be classified into 3 groups in terms of the visit to the bin layer, as shown in Fig. 6. The first group only queries the bin layer and the corner stitching layer, such as point searching and area searching/enumeration. The second group not only queries the two-layer structure, but also updates the database, such as insertion, deletion and plowing. Note that the two layers represent the same layout area, so any modifications should be made in both layers at the same time. The third group doesn't need any orientation by coordinates, thus only queries the corner stitching layer, such as neighbor enumeration.

We have adapted algorithms from [9] and [13] for the operations of the corner stitching layer. However, PB corner stitching is aiming at routing, so the algorithms have some differences with [9] and [13], for instance, the rotated figures are not considered. Also in routing phase, the plowing operation has its own specific characteristics, which are considered in our plowing algorithm.

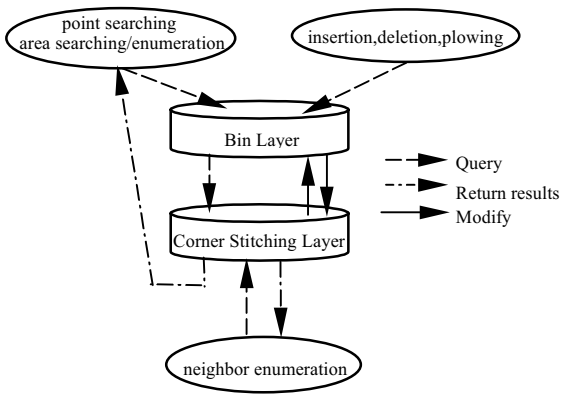


Fig. 6. Operations of the hierarchical PB corner stitching

*B.1 Point Searching*

Point searching is used to locate the tile containing a given point P. The algorithm first obtains a tile near the point from bin layer, then switches to the corner stitching layer, starts with that tile, and iterates in y and then x until the target tile is found. The algorithm goes as follows:

1) Given the coordinates of P: Xp,Yp, and the size of bin: S, figure out the bin that P lies in, called bin(a,b), where a=⌊Xp/S⌋, b=⌊Yp/S⌋.

2) If bin(a,b) is not empty, then set the starting tile to bin(a,b), go to step 3. Otherwise, search all the bins that lie to the lower left of bin(a,b), until a bin that is not empty, or bin(a1,b1) is found, where 0<=a1<=a, 0<=b1 <=b, then set the starting tile to bin(a1,b1).

3) Move upwards using rt stitches or downwards using lb stitches until a tile is found whose vertical range contains Yp.

4) If the tile contains P, then stop. Otherwise move left using the bl stitches, or move right using the tr stitchs.

5) Go to step 3).

Suppose the bins are much larger than the average tile size, then the speed of the point searching depends on step 3) to step 5), which is traditional corner stitching point searching operation in a bin's range. Thus the complexity of point searching is $O(N^{1/2}/r)$, where N is the tile number, $r^2$ is the number of bins.

*B.2 Area enumeration*

Area enumeration is used to report all tiles in a tile plane by giving a bounding box as the area. In our algorithm, the definition of area is different from [13]. The area can be one of the 9 possible shapes in Fig. 1, but can't be any rotated rectangles. Also, we change the visit sequence of the tiles. Tiles are visited from left to right, and from bottom to top instead of from top to bottom. This modification is based on the fact that when moving from one tile to the tile above it, the algorithm can directly use the y coordinate of the current tile's top edge to do a point searching, but when moving downwards, the y coordinate used in point searching should be the y coordinate of the bottom edge minus 1, which implies that the coordinates in the plane should be integers.

The area enumerate complexity assumes the tile that covers the lower left corner of the area has already been found. Since the searching area is limited, the point searching used in area enumeration is the traditional corner stitching point searching. Area enumeration is O(n), where n is the tiles in the area.

*B.3 Tile Insertion/Deletion*

Our insertion/deletion algorithms have the same main flow with [9], our modifications focus on the split and merge functions used in both insert and delete operations.

For trapezoidal corner stitching, the shapes of tiles might be changed after being split or merged. For splitting a tile into two, if the cutting line is horizontal, then the shapes of the two resultant tiles are the same with the original tile. If the cutting line has an angle of 45°, 90° or 135° with the x-axis, then the resultant tiles' shapes depend not only on the original tile shape, but also on the slope of the cutting line. For merging two tiles into one, if the two tiles share a common horizontal edge, then the shapes of two tiles should be exactly the same, and the resultant shape is also the same with the original shape. If the common edge is not parallel with the x-axis, then the right edge of the left tile should have the same slope with the left edge of the right tile, and the resultant shape depends on both tiles.

When splitting or merging the tiles, contents of the bin layer should also be changed to maintain the consistency of the database. After a tile is split, the pointer of that tile should be deleted from the bin layer if there exists one, and two pointers of the resultant tiles should be added into the bin layer. Similarly, after two tiles are merged, two old pointers should be deleted from the bin layer if they exist, and the new pointer should be added into the bin layer. When updating the bin layer, the algorithm first finds the target bin by the lower left corner of the tile. When a tile is added, the content of the target bin would be set to the new tile pointer. When a tile is being deleted, the content would be set to null if that tile pointer is kept in the bin, or just remains unchanged if the pointer doesn't exist in the bin.

*B.4 Plowing*

Given an area A and a direction d, plow operation is used to clear out the solid tiles in area A by moving the involved tiles along the d direction. Since our data structure would be used in routing algorithm, so plowing would be quite different from that of [9] and [13]. Firstly, in routing phase, plowing is used to push aside the existing nets that blocked the way of the extension path, but the positions of the cells and pins, which are determined during the placement phase, should not be changed. Therefore, the plowing algorithm has to consider the type of a solid tile, only net tiles could be modified. Secondly, when modifying the net tiles, the constraints not only include the movable range of the tiles, but also include connectivity of the nets and the design rules. Finally, nets are flexible, the position, size and shape of the net tiles could all be changed, though the ends of the nets have to remain where they are.

Like the plow operations in [9], our algorithm also visits the tiles twice to do the plowing. During the first visit, the algorithm determines for all related tiles the modification type, including moving, shortening, elongating and transforming, the range of the change, and where the transformation starts and ends if a transformation is compulsory. Notice that plowing doesn't take place in only one layer, but on all layers that the related nets span. After moving a net tile, all the conjoint tiles that belong to the same net should be modified to maintain connectivity. During the second visit, the tiles are really modified according to its modification plan, which is determined during the first visit. Fig. 7 is an example of plowing, where tile e is unmovable, A is the given area, and d is the given direction. After plowing, tile b is shortened, tile d is elongated, tile c is moved, tile a has to be transformed since tile e is on its way.

The tile structure has to increase a field to store the modification plan structure of the tile. The modification plan

would include the type and range of the modification and the starting and ending positions of a transformation if necessary. In order to save memory, the modification plan could be kept as a pointer in the tile structure.

The algorithm goes as follows:

1 Work out the modification plan of all related tiles:

For each solid tile T found by area enumeration in area A, do 1) —5):

1) If the type of T is not net, return failure.

2) Otherwise, determine the modification plan for T, and insert T into the front of the tile list for that particular layer.

3) Get the A1 area that tile T would pass when being modified. If A1 is empty, then return.

4) For each solid tile V that intersects with A1, if V is visible to T, do 4.1) and 4.2).

4.1) Figure out the new modification plan of tile V, if it is not covered by the original modification plan of V, then update the modification plane of V.

4.2) If this is the last time to visit V, then recursively call the procedure 1) —5) for tile V.

5) Modifications to maintain the connectivity of the nets. For each tile C of a different layer that meets the following requirements: be joined together with T, belongs to the same net with T, do 5.1) and 5.2).

5.1) Work out the modification that C needs according to the changes of T. If the modification is not included by the original modification plan, then update the modification plan of C.

5.2) If this is the last time to visit C, then recursively call the procedure 1) —5) for tile C.

2 Actually modify each related tile. For each related layer, visit the tile list of that layer. For each tile in the linked list, first delete the tile from the database, and then insert the tile back into the database according to its modification plan and original position.

### *C. Complexity Comparisons*

In this section, we compared the speeds of the proposed hierarchical PB corner stitching and some existing data structures. The results are listed in TABLE I, where N is number of tiles, $r^2$ is number of bins, T is the number of solid figures threshold per quadrant, and n is the number of tiles in the area.

It should be pointed out that although the speeds of several operations of PB corner stitching decrease when $r^2$ increases, it doesn't mean that the larger $r^2$ is, the faster the algorithm is. When estimating the complexity of PB corner stitching, we assume that the average tile number in one bin is larger than one, otherwise, the results would be different, for instance, point searching would be $O(r^2/N)$. Our experiments showed that when the average tile number in one bin ranges from 1 to 100, the efficiency of the structure would be most satisfactory.

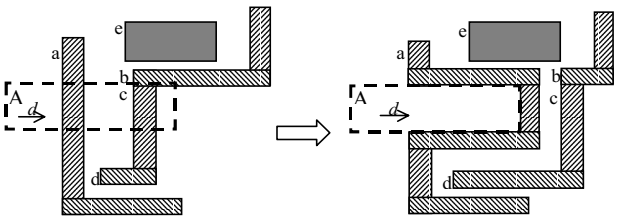

Fig. 7. Plowing.

## IV. Experimental Results

We implemented corner stitching and PB corner stitching on Sun Ultra Enterprise 450 work station, and tested the performance of the two structures by layout examples whose tile number vary from 42 to 39802. We chose the bin number as 30×30. Fig. 8 shows the average time used for 10 point search operations. When ln(N) is less than 6.4, that is, tile number less than 600, point searching of PB corner stitching is not faster than that of corner stitching, that is because the average tile number per bin is far less than 1. With the increasing of tile number, point searching of PB corner stitching becomes faster than traditional corner stitching. Fig. 9 shows the average time for inserting N tiles into an empty area. Also, when the tile number is small, the insertion of PB corner stitching has no advantage over that of corner stitching, since there are additional works to manage the bin layer. When the tile number is more than 500, the inserting speed of PB corner stitching becomes faster than that of traditional corner stitching, since the time saved from point searching exceeds that used in managing the bin layer.

## V. Conclusions

The hierarchical PB corner stitching, which combines the traits of both bin-based structure and corner stitching, enhances some basic operations needed in routing algorithm, such as point searching, area searching, insertion, deletion, etc, thus could handle a larger problem size. The memory requirement of PB corner stitching increases $r^2$, which is quite acceptable considering the constant increase of computer memory.

The number of bins would affect the efficiency of PB corner stitching. If the bin number is too small, and then PB corner stitching tends to resemble traditional corner stitching, the improvement of efficiency is limited. If the bin number is too large, then a tile may cover several bins. It might be slower to find a nearby tile to a point in the bin layer, thus affect the speed of point searching. Also, if there are too many bins, the memory requirement would increase too much.

TABLE I
Comparison Of Speeds

| Operation | Linked List | Bin | Quad Tree | 4-D Tree | Corner Stitching | PB Corner Stitching |
|---|---|---|---|---|---|---|
| Point Searching | O(N) | O(N/r) | O(logN +T) | O(logN) | $O(N^{1/2})$ | $O(N^{1/2}/r)$ |
| Neighbor Searching | O(N) | O(N/r) | O(logN +T) | O(logN) | 1 | 1 |
| Area Searching | O(N) | —— | O(T+ n) | O(n) | O(n) | O(n) |
| Insertion | 1 | $O(N/r^2)$ | O(log N) | O(logN) | $O(N^{1/2})$ | $O(N^{1/2}/r)$ |
| Deletion | O(N) | $O(N/r^2)$ | O(logN +T) | O(logN) | $O(N^{1/2})$ | $O(N^{1/2}/r)$ |

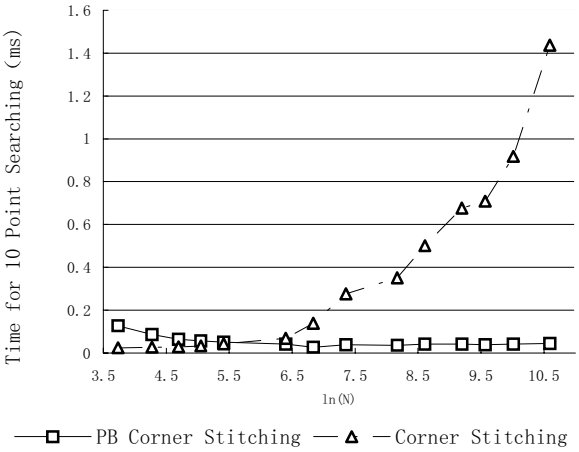


Fig. 8. The speed of point searching

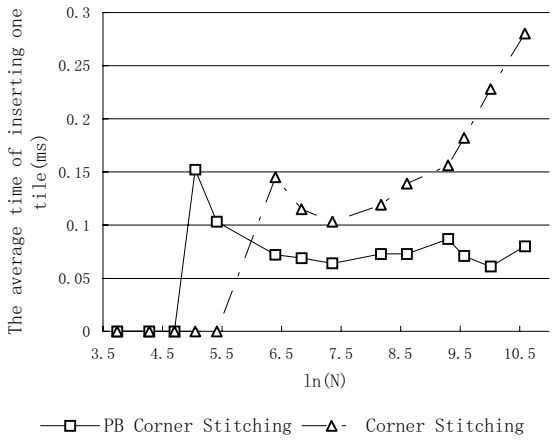


Fig. 9. The speed of insertion

## REFERENCES


[1] C.Y. Lee, "An Algorithm for Connections and Its Application", *IRE Trans. on Electronic Computers,* pp. 346-365, 1961.

[2] S.B.Akers, "A Modification of Lee's Path Connection Algorithm", *IEEE Trans. on Electronic Computers,* pp. 97-98, 1967.

[3] F. Rubin, "The Lee Connection Algorithm", *IEEE Trans. on Computers,* pp. 907-914, 1974, 23.

[4] Hadlock, "A Shortest Path Algorithm for Grid Graphs", *Networks,* pp. 323-334, 1977,7.

[5] J. Soukup, "Fast Maze Router", *Proc. of 15th Design Automation Conference,* pp. 100-102, 1978.

[6] Jon L. Bentley, "Multidimensional binary search trees used for associative searching", *Commun. ACM,* vol. 18, no. 9, pp. 509-517, 1975, 9.

[7] R. Brown, "Multiple storage quad trees: A simpler faster alternative to bisector list quad trees", *IEEE Trans. on Computer-Aided Design,* vol. CAD-5, pp.413-419, 1986,7.

[8] A. Pitaksanonkul, S. Thanawastien, and C. Lursinsap, "Comparisons of Quad Trees and 4-D Trees: New Results", *IEEE Transactions on Computer-Aided Design,* vol. 8, No. 11, pp.1157-1164, 1989,11.

[9] J. Ousterhout, "Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools", *IEEE Trans. on CAD,* pp.87-99, 1984, 3(1).

[10] A. Margarino, A. Romano, A. De Gloria, F.Curatelli and P. Antognetti, " A Tile-Expansion Router." *IEEE Transactions on Computer-Aided Design,* Vol CAD-6, No.4, pp.507-517, July 1987.

[11] C. Tsai, S. Chen and W. Feng, "An H-V Alternating Router", *IEEE Transactions on Computer-Aided Design,* Vol. 11, No.8, pp.976-991, 1992,8.

[12] L. Liu, H. Tseng, and C. Sechen, "Chip-Level Area Routing", pp.197-204, ISPD '98 Monterey CA USA.

[13] D. Marple, M. Smulders and H. Hegen, "Tailor: A Layout System Based on Trapezoidal Corner Stitching", *IEEE Transactions on Computer-Aided Design,* Vol. 9. No.1. 1990,1.

[14] J. Parkhurst, N. Sherwani, S. Maturi, D. Ahrams, and E. Chiprout, "SRC Physical Design Top Ten Problems", pp. 55-58, ISPD '99 Monterey CA USA.