A Gridless Router Based on Hierarchical PB Corner Stitching Structure*

Zhang Yiqian, Cai Yici, Hong Xianlong, Zhang Yan and Xie Min

(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084, *China*)

Abstract: A multi-layer gridless area router is reported. Based on corner stitching, this router adopts tile expansion to explore path for each net. A heuristic method that penalizes nodes deviating from the destination is devised to accelerate the algorithm. Besides, an enhanced interval tree is used to manage the intermediate data structure. In order to improve the completion rate of routing, a new gridless rip-up and rerouting algorithm is proposed. The experimental results indicate that the completion rate is improved after the rip-up and reroute process and the speed of this algorithm is satisfactory.

Key words: gridless area routing; rip-up and reroute; corner stitching structure; VLSI

EEACC: 2570 **CCACC**: 7410D

CLC number: TN47 **Document code**: A **Article ID**: 0253-4177(2003)02-0141-07

1 Introduction

With the development of semiconductor technology, designers' demands for area routers improve a lot. The top 10 problems in physical design, which were put forward by SRC in 1999, 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 exist constraints of the position of route. Grid based routers[1,2] 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 the advantages: it reduces the memory requirements by using line based or tile based data structures, thus could handle a larger size of work; it enjoys more flexibility in accommodating complicated design rules; and it could reach a higher completion rate. Hence, our target is a general gridless area router with high completion rate and super performance.

In this paper, we first propose a new area routing oriented hierarchical structure, which combines the bin-based structure and corner stitching structure. Also, we renew some operations for area routing. Based on this data structure, a multi-layer gridless area router is given. And a new gridless rip-up and reroute algorithm is proposed. Our router combines an efficient data structure with a suitable routing algorithm, and the experimental results indicate that the completion rate is improved after the rip-up and reroute process and the

* Project supported by National Natural Science Foundation of China (No. 60166016) and China 973 National Fundamental Funds (No. G1998030403)

Zhang Yiqian female, was born in 1977, PhD candidate. Her research interests focus on VLSI layout design.

Received 12 July 2002, revised manuscript received 8 October 2002

speed of this algorithm is satisfactory.

2 Hierarchical PB corner stitching structure

There are some existing data structures, including a linked list, a bin-based structure, K-D tree, and quad tree. These structures have common drawbacks. They provide no assistance in locating empty space for routing since only the occupied spaces are represented explicitly. The corner stitching[5] and trapezoidal corner stitching[6] arise from a consideration of this drawback. And the corner stitching is probably the most suitable layout database for a gridless area router. However, we have not found any published area router that use trapezoidal corner stitching as layout database.

The layout data structure is designed according to the demands of the routing algorithm. Firstly, the speed of point searching should be improved. Point search 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 $O(\lg N)$. Next, the data structure can handle figures with 45° edges. Finally, the structure should represent multi-layers.

To remedy this, we combine corner stitching with the bin-based structure and come out with a new hierarchical PB corner stitching structure (hierarchical corner stitching with partial bin) 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, the corner stitching layer and bin layer, as shown in Fig. 1.

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 belong to, what is the tile's relative position to the median tile, or which bin does the tile cover. It is just because of lacking such global position information that point searching is inefficient for traditional corner stitching.

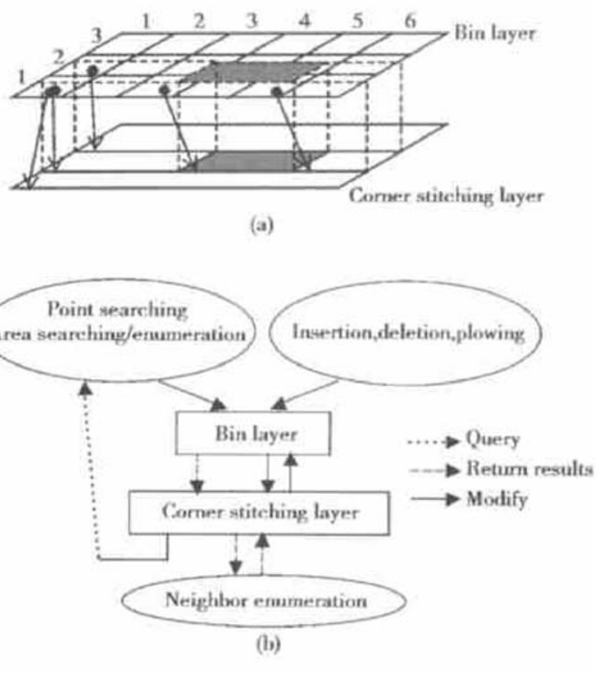

Fig. 1 (a) Configuration of hierarchical PB corner stitching; (b) Operation of hierarchical PB corner stitching

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. 1(a). An imaginary square grid divides the area into $m \times n$ 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). But if we add the bin-based structure directly 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 stitching with partial bin), as shown in Fig. 1(a). A tile is only kept in the bin where a low left corner of the tile's projection locates.

The structure at the corner stitching layer is a routing-oriented corner stitching. Our router divides a multi-layer into 2-layer or 3-layer routing. 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. 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. The experimental results are shown in Table 1. We recommend a more detailed survey by Ref. [7] to the interested reader.

Table 1 Comparison of speeds

Operation	Linked list	Bin	Quad tree	4-D tree	Corner stitching	PB corner stitching
Point searching	$O(N)$	$O(N/r)$	$O(\lg N+T)$	$O(\lg N)$	$O(N^{1/2})$	$O(N^{1/2}/r)$
Neighbor searching	$O(N)$	$O(N/r)$	$O(\lg N+T)$	$O(\lg N)$	1	1
Area searching	$O(N)$	—	$O(T+n)$	$O(n)$	$O(n)$	$O(n)$
Insertion	1	$O(N/r^2)$	$O(\lg N)$	$O(\lg N)$	$O(N^{1/2})$	$O(N^{1/2}/r)$
Deletion	$O(N)$	$O(N/r^2)$	$O(\lg N+T)$	$O(\lg N)$	$O(N^{1/2})$	$O(N^{1/2}/r)$

N is the number of tiles, r^2 is the number of bins, T is the number of solid figures threshold per quadrant, and n is the number of tiles in the area.

3 Gridless area router

3.1 Tile expansion algorithm[4]

Figure 2 illustrates an example of H-V tile expansion used in routing a two-terminal net (from S to T). First, a starting space tile (H4 in this example) from the left side of terminal S and the H-V tile expansion beginning, S is completed when one of the tiles, H7 or V6, is reached. In this H4 expansion, only one V-space tile, V1, is generated. Hence, the V expansion starts from space tile V1. There are five H-space tiles (i.e., H1, H2, H3, H4, and H5) generated in this V1 expansion. Note that H-space tile H4 has been expanded in the previous expansion level and that tile H2 belongs to the bypass tile where no tile expansion is allowed. The H-space tile H3 is chosen as the next H expansion because its estimated cost is the lowest. Also the V-space tile V3 is the next new V expansion tile since other tiles (V1 and V7) belong to the negligible tiles in the H3 expansion. Finally, when the tile H7 is reached in the V3 expansion, the H-V tile expansion can be terminated. As the H-V tile expansion proceeds, an expanding tree is constructed, as shown in Fig. 2(b), where the heavy solid lines de-

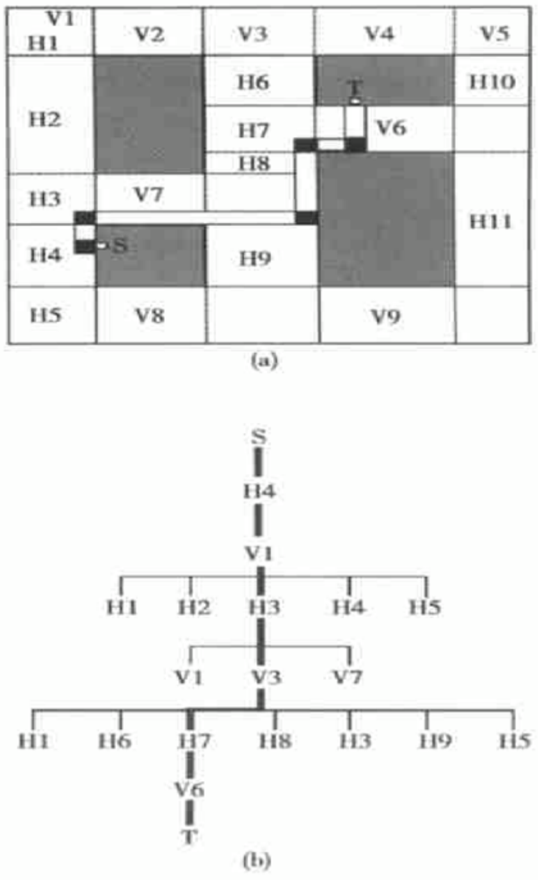

Fig. 2 (a) Tile expansion; (b) Expanding tree note the connection path of the net. Once the expanding tree is constructed, all wire segments are routed along the connection path backward from the tile V6 to the starting space tile H4 with an ordering of (V6, H7, V3, H3, V1, H4). The connection path thus obtained is the shortest one between terminals S and T, as shown in Fig. 2(b).

3.2 Destination-oriented heuristic cost function

The cost of tile expansion includes two parts:

$cost_1 = I_{d1}(\text{source, current}) \times \text{ManDist}(\text{source, current})$

$cost_2 = I_{d2}(\text{current, destination}) \times \text{ManDist}(\text{current, destination})$

where I_{d1} is the penalty concerning the direction of tile expansion, I_{d2} is the penalty concerning the tile's relative position to the destination. In our algorithm, some heuristic information is added so that the destination-oriented tiles can be expanded firstly. For the tiles that are expanded towards the destination, the values of I_{d1} are small. On the contrary, the values of I_{d1} are large for the tiles that are deviating from the destination. So the speed of our algorithm is improved. An example shown in Fig. 3, four tiles, A, B, C, and D, are all obtained by multi-layer expansion. Since tile C and tile D are

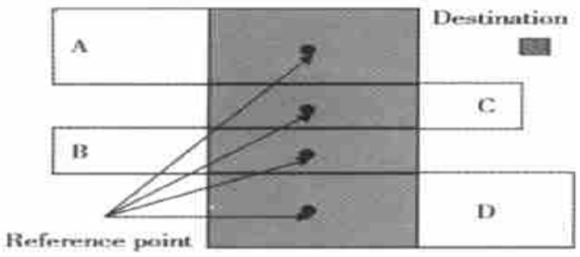

Fig. 3 Calculating of tile expansion cost

expanded towards the destination while tile A and tile B are expanded away from the destination, the costs for C and D are lower than the costs for A and B. Thus, C and D are expanded firstly.

3.3 Rip-up and reroute process

The algorithm flow is shown in Fig. 4. Firstly, some necessary pretreatment in detail is done for each net. Then the nets that will be routed are determined. After that, the algorithm adopts tile expansion to explore path for each net. Our algorithm considers restraints of routing resources when exploring path for each net, thereby some unnecessary explorations and comparisons are eliminated. Finally, rip-up and reroute is done, and in order to indicate the circumstance in the routing area, the congestion variables are updated dynamically according to the procedure of rip-up and reroute. To avoid the endless of rip-up and reroute process, a threshold is adopted. And the tiles that exceed the threshold are rejected in tile expansion algorithm.

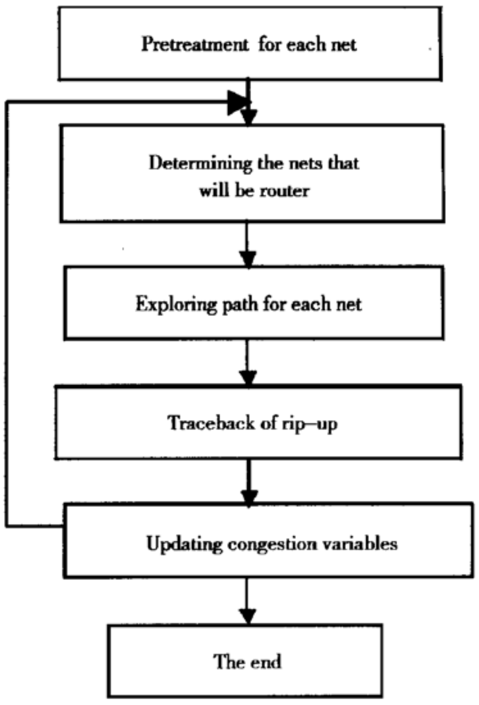
Pretreatment for each net
Determining the nets that will be router
Exploring path for each net
Traceback of rip-up
Updating congestion variables
The end

Fig. 4 Algorithm flow

Rip-up and reroute also determine the path for each net by tile expansion while tiles can be expanded in the same layer and same direction additionally. This kind of expansion goes on until meeting obstacles. And the expanding tiles and the routed nets overlap each other. The multi-layer expansion can not produce the paths obtained in the expansion of the same layer and same direction, so the space of solution is enlarged greatly compared with initial routing.

The key of rip-up and reroute is to choose the nets that are needed to be rip-up. The purpose is not only to route the current net successfully, but also to route the nets that have been rip-up easily. So the net should be routed to avoid going through the congest area in the process of rip-up and reroute. Since the path obtained by tile expansion algorithm keeps plenty of local neighboring information, we can use the information to direct the choice of the nets that is rip-up. So, a new gridless rip-up and rerouting algorithm is presented in this paper. The algorithm considers the congestion of the routing area while exploring path for each net, and combines the choice of the nets that are rip-up with the path exploration. In our algorithm, some variables are used to express the congestion around the nets explicitly. Accordingly, the cost of congestion is added to the expansion cost of each tile in tile expansion algorithm.

$$\mathrm{cost}_3 = -\sum_{i\in \mathrm{OSet}} \mathrm{congest}(\mathrm{Net}\ i)$$

$$\mathrm{OSet} = \{i \mid \mathrm{Net}_i \text{, overlaps the path from source to current node}\}$$

where Net_i is the net that overlaps the current path. Congest is calculated based on two factors. One is the congestion variable of the area that the nets belong to. The other is the overlapping area.

The heuristic method of calculating congest cost is shown in Fig. 5. Two candidate nets (A and B) are found from the current tile. But the nets that are around net B are sparse than the nets that are around net A. So the congest cost of net B is lower than that of net A.

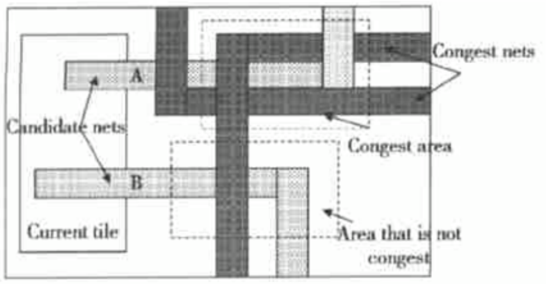

Fig. 5　Calculating of the congest cost

Therefore, the exploring process of rip-up and reroute lies on not only the path length, but also the congestion of the area that the nets that will be rip-up belong to. Finally, we get a path that is a tradeoff between path length and congestion. The tiles with high congestion costs also have high expansion costs. Our tile expansion algorithm chooses the tile which has the lowest expansion cost to expand. Thus the exploring process considers the nets that are not in the congest area firstly. Similarly, the congestion variables are also used to guide routing in the procedure of trace.

In order to indicate the circumstance in the routing area, the congestion variables are updated dynamically according to the procedure of rip-up and reroute. When some nets are rip-up, the values of the congestion variables in relative areas increase in proportion.

$$\mathrm{congest}(\mathrm{Net}_i)_{\mathrm{new}} = \mathrm{congest}(\mathrm{Net}_i)_{\mathrm{old}} + \mathrm{delta}$$

Additionally, this method can influence the tiles´ congestion weights in the later expansions for other nets, in case of that the nets that are just routed are chosen to rip-up.

3.4　Enhanced interval tree

Each node in the process of exploring represents for an available rectangular routing area. It is possible that different paths arrive at the same area. In order to enlarge the exploring area, our algorithm combines neighboring tiles and introduces the expansion in the same layer and same direction, but in this case, some tiles overlap each other. If two tiles have same directions, the algorithm only holds the tile that has a smaller expansion cost in order to reduce redundant exploration. Four comparisons are needed to judge the relationship of two tiles, and each new tile should be compared with the tiles in Openlist one by one, so the time complexity is $O(n)$, n represents the number of tiles in the Openlist. When n is getting larger, more time is needed in the judgement. So an efficient data structure is needed to organize these tiles so that some needless judgements are eliminated.

In this paper, an enhanced interval tree is adopted to manage tiles hierarchically. This data structure holds the hierarchical organization so that the needless judgements can be eliminated. At the same time, directions are partitioned uniformly, so some repeat operations and excess maintenance are also eliminated. And many long and narrow tiles are produced in the process of tile expansion because of the existence of original obstacles and the routed nets. So in an interval, the area occupied by each tile is quite different from the area that the tree node indicates during the process of routing. To reduce redundant comparisons further, our algorithm improves the original interval tree, and records the very left boundaries and the very right boundaries of the set of tiles. As shown in Fig. 6, according to the information about the left and the right boundaries, area 1 and area 3 do not need to be compared with the four tiles in the tree node, but area 2 needs to be compared with the four tiles one by one. Compared with the interval tree, adding the information about boundaries for each node leads to the additional $O(t)$ time complexity, t represents the number of tiles in one node. Experimental results show that comparison times are reduced by 50% after adopting the enhanced interval tree.

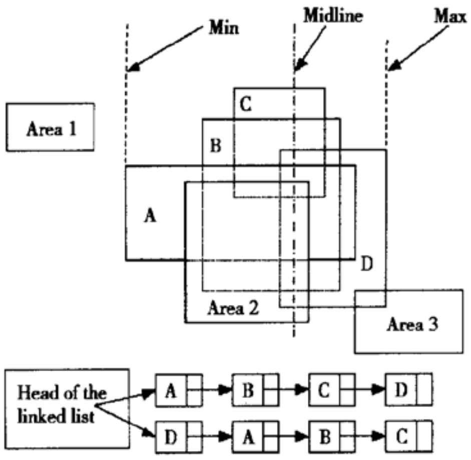

Fig. 6 Linked list in tree node

4 Experimental results

Our routing system is implemented in C language on SUN Enterprise E450. The test circuits, C2, C5 and C7 are provided by MCNC, and the routing algorithm[1,3] is adopted to do the initial routing before rip-up and reroute. The results are shown in Table 2. For the same example, the grids for global routing can be partitioned differently, and different partitions can lead to different initial completion rate and total wire length. In Table 2, C2_chip is the result of doing detailed routing directly without global routing. It is obvious that the running time increases greatly, but the total wire length is reduced by 15% compared with the result of doing detailed routing after global routing. Figure 7 shows one of the results in Table 2. Experimental results show that our algorithm can improve the completion rate greatly.

Table 2 Experimental results

Bench mark	Number of areas	Number of nets	Running time/s	Completion rate before rip-up/%	Completion rate after rip-up/%	Total wire length /μm
C2	9×11	745	197	92.8	99.9	5.424×10^5
C2_9	3×3	745	245	97.3	99.9	5.166×10^5
C5	16×18	1764	400	89.1	100	1.380×10^6
C5_30	5×6	1764	443	96.8	100	1.307×10^6
C7	16×18	2356	609	89.7	99.9	2.152×10^6
C7_30	16×18	2356	679	94.2	99.9	2.039×10^6
C2_chip	1×1	745	618	91.9	100	4.629×10^5

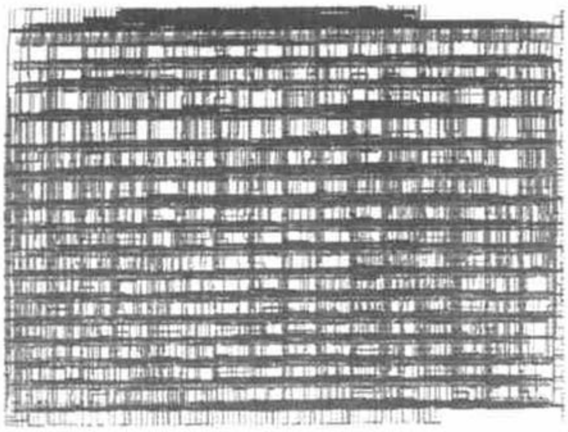

Fig. 7　Experimental result for C7

5　Conclusions

In this paper, we present a multi-layer gridless router based on hierarchical PB corner stitching structure. Experimental results indicate that the completion rate is improved after the rip-up and reroute process, and the speed of this algorithm is satisfactory.

References

[1]　Lee C Y. An algorithm for connections and its application. IRE Trans Electronic Computers, 1961, EC-10: 346

[2]　Soukup J. Fast maze router. Proc of 15th Design Automation Conference, 1978: 100

[3]　Margarino A, Romano A, De Gloria A, et al. A tile-expansion router. IEEE Trans Comput-Aided Des Integr Circuits Syst, 1987, 6(4): 507

[4]　Tsai C, Chen S, Feng W. An H-V alternating router. IEEE Trans Comput-Aided Des Integr Circuits Syst, 1992, 11(8): 976

[5]　Ousterhout J. Corner stitching: a data structuring technique for VLSI layout tools. IEEE Trans Comput-Aided Des Integr Circuits Syst, 1984, 3(1): 87

[6]　Marple D, Smulders M, Hegen H. Tailor: a layout system based on trapezoidal corner stitching. IEEE Trans Comput-Aided Des Integr Circuits Syst, 1990, 9(1): 66

[7]　Zhang Yan, Wang Baohua, Cai Yici, et al. Area routing oriented hierarchical corner stitching with partial bins. Proc Asp Dac, 2000: 105

[8]　Ma Qi, Yan Xiaolang. MARS: a general multilayer area router. Chinese Journal of Semiconductors, 2001, 22: 516

[9]　Xu Jingyu, Bao Haiyun, Hong Xianlong, et al. A fast and efficient global router for congestion optimization. Chinese Journal of Semiconductors, 2002, 23: 136

一个基于层次式 PB 角钩链结构的区域布线器*

张轶谦　蔡懿慈　洪先龙　张　雁　谢　民

(清华大学计算机科学与技术系, 北京　100084)

摘要: 提出一个新的基于层次式 PB 角钩链结构的多层无网格布线器. 该布线器基于 PB 层次式角勾链数据结构和网块扩展算法, 使用朝向目标的加速策略提高算法的运行速度, 并使用改进的二叉区间树管理算法的中间数据. 还提出了基于拥挤度的无网格拆线重布算法. 通过显式记录每个线网段周边的拥挤状况, 并将其结合到网块扩展的费用当中, 使拆除线网的选择和待布线网的路径搜索统一起来. 实验结果表明, 该布线器能有效地提高布通率, 且算法运行速度较快.

关键词: 无网格区域布线; 拆线重布; 角钩链; VLSI

EEACC: 2570　　**CCACC**: 7410D

中图分类号: TN47　　**文献标识码**: A　　**文章编号**: 0253-4177(2003)02-0141-07

* 国家自然科学基金(批准号: 60167016)及国家重点基础研究发展规划(No. G1998030403)资助项目

张轶谦　女, 1977 年出生, 博士研究生, 主要从事超大规模集成电路布图算法的研究.

2002-07-12 收到, 2002-10-08 定稿