Topological Routing amidst Polygonal Obstacles

Swarup Bhunia *Subhashis Majumder*
Ayan Sircar
Delsoft India Pvt. Ltd.
Calcutta 700091, India
{*swarup,sm,ayan*}*@cal.delsoft.com*

Susmita Sur-Kolay
Bhargab B. Bhattacharya
Indian Statistical Inst.
Calcutta 700035, India
{*ssk,bhargab*}*@isical.ac.in*

Abstract

This paper presents a fast graph-traversal based greedy approach for solving the problem of topological routing in the presence of polygonal obstacles. The polygonal obstacles represent pre-routed nets or groups of circuit blocks. Routing paths for all the nets are constructed incrementally and concurrently. Design rules for separation are modeled as constraints on edges and vertices. The experimental results obtained are very encouraging.

1 Introduction

The problem of topological routing is reincarnation of global routing in the state-of-the art automated layout synthesis. Topological routing approach has gained importance, particularly for the mixed block and cell design style [8] — a combination of full-custom and standard-cell style. The objective is to generate a *sketch* [4, 1, 2], i.e., a set of lines connecting the terminals of the nets on pre-designed and placed blocks. This sketch is subsequently transformed into a geometrical routing satisfying the design rules [3]. Testing homotopic routability of a given sketch on a single layer is solvable in polynomial time and space [4, 5, 6, 7].

2 Motivation

In topological routing, the motivation is to solve global routing efficiently. In sequential maze routers or line probe methods [8], nets are routed one by one, usually along the current shortest available path among obstacles. The ordering of nets plays a critical role in determining whether 100% routing is achievable. Re-ordering cannot guarantee completion even though a solution exists because routers emphasize on finding the shortest path at every instance. Further, finding two vertex disjoint paths between two vertices in a planar graph is NP-hard [9].

A better solution may be obtained if the problem is viewed globally in a concurrent manner, rather than net by net. While there are concurrent global routers primarily based on integer programming formulation, topological routing approach is more appropriate for large prolem instances. For routing in the presence of pre-routed special nets or groups of blocks, these pre-routed groups can be represented as polygonal obstacles for the router. Thus the challenge is to route all the nets concurrently with polygonal blocks instead of rectangular ones. Such a routing paradigm along with efficient layer assignment is likely to address many of the salient issues in current VLSI layout synthesis.

In this paper, the problem of topological routing of m nets $\{n_1, n_2, \ldots, n_m\}$ in the presence of k non-overlapping *convex polygonal* obstacles $\{P_1, P_2, \ldots, P_k\}$ is studied. These obstacles represent one or more functional blocks with pin alignment and/or local routing completed. A net may appear at most once on a polygonal obstacle and the pins of the corresponding nets are placed along the edges of the polygons. The sketch is generated by processing all the nets concurrently to circumvent the net ordering problem in traditional global routers.

Most of the existing topological routers for multilayer MCMs consider routing space with vias as the only obstacles, whereas we are concerned with routing in the presence of pre-placed finite polygonal obstacles in the routing layers. The associated layer assignment problem is not considered here.

The method presented here comprises of
(i)decomposing the free space available for routing into trapeziums,
(ii) constructing a routing graph with separation constraints as weights,
(iii) traversing the graph to find routing paths for all the nets and at the same time satisfying separation constraints.

The organization of this paper is as follows. Section 3 presents the algorithm for trapezoidal decomposition and graph generation. The core algorithm for generating the global routing paths appear in Section 4. The performance of our algorithm along with a brief description of generation of arbitrary problem instances and experimental results are given in Section 5. Section 6 gives the concluding remarks.

3 Trapezoidal decomposition

As outlined above, our method consists of two major subtasks. First, the free intermodular space on the floor is partitioned into trapeziums so that the edges of these trapeziums satisfy the minimum separation design rule. A computational geometric horizontal line

sweep approach is used. Then a directed graph D which is the geometric dual of the planar trapezoidal decomposition, is constructed from it.

Input : A rectangular routing space R' with convex polygonal obstacles. The polygons are pre-routed modules with interconnection terminals for a set of nets N on their boundaries. Without loss of generality, we assume that no polygon touches R' and all terminals on a polygon belong to distinct nets.

Output : A directed acyclic graph where each vertex corresponds to a trapezium and an edge between vertices v_i and v_j corresponds to a common trapezoidal edge. The trapeziums are generated by partitioning the free space (rectangular area minus the polygonal area) into a set of mutually exclusive and collectively exhaustive trapeziums.

Method : There are two parts in the process -
(a) generation of trapeziums,
(b) generation of dual graph from the trapeziums.

Generation of trapeziums : We do this by passing a horizontal sweep line S over the routing space R' from top to bottom. S can be moved from any side of R' to the opposite one, resulting in a different set of trapezoidal partitions, but the set is unique for any particular direction.

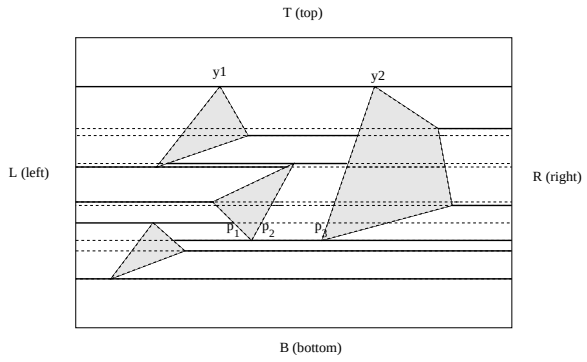

Figure 1: Generation of Trapeziums

Implementation Steps :

1. The rectangular routing space R' and the convex polygonal obstacles are read from a file and the vertices of the polygons are kept sorted in descending order of their y-coordinate values in a list L.

2. For each distinct value of y occurring in L, do

- 2.1 Obtain all the vertices with the same y-coordinate. Let $x_1, x_2, \ldots, x_k$ be the x coordinates of those polygonal vertices lying on S.
- 2.2 Obtain all the intersections of S with the set of polygonal edges. The edges are kept sorted in terms of their maximum y-value. Let these intersection points be $x_m, x_{m+1}, \ldots, x_n$
- 2.3 Perform merge sort on $x_1, \ldots, x_k$ and $x_m, \ldots, x_n$
- 2.4 Traverse the list of points sorted in ascending order of x-values and identify all *viable* trapezium edges. For a viable trapezium edge both the end-points cannot lie on the same polygon, eg., in Fig. 1 (p_1, p_2) is not a viable trapezium edge. Moreover, at least one end of the edge should end in a polygonal vertex or has to be either on sides L or R. So (p_2, p_3) is also not a viable edge. The thick solid lines in Fig. 1 are viable edges. Let this edge-set be $e_i, \ldots, e_k$ which consists of points $p_l, \ldots, p_m$.

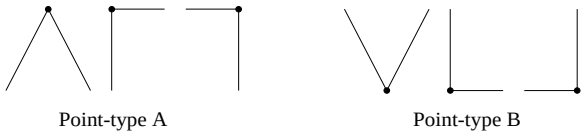

Figure 2: (a) Point type-A (b) Point type-B

- 2.5 Traverse the list of end-points of viable edges to identify actual trapezium edges. The top edge of the first trapezium will be side T of R'. Similarly, the bottom edge of the last trapezium will be side B (Fig. 1) of the rectangular floor. The criteria for identifying the top and bottom edges of trapeziums are as follows. If a point p_i is of type-A (Fig. 2a), we do not consider it as an end-point of a bottom edge of a trapezium and proceed to the next point on S to the right of p_i. Similarly, points of type-B (Fig. 2b) are ignored for identifying the end-points of a top-edge of a trapezium. The points on S which are neither of type-A nor of type-B, are declared as endpoints of trapezium edges.
- 2.6 To report a trapezium, a bottom edge has to be matched with a previously detected top edge. A list UT of unmatched top edges is maintained. It is initialized to side T and this is matched with the sole bottom edge generated at the first position of S. However, new top edges are detected and inserted in the list UT. As soon as a bottom edge e_b matches with top edge e_t in UT, the edge e_t is deleted from UT and a new trapezium is reported.

  Matching e_b with an e_t is performed as follows:

  (i) among all edges in UT, consider the edge e_t whose midpoint is nearest to that of e_b;

  (ii) then check whether the left end-points of both the edges lie on the same polygonal edge, if yes, match e_t with e_b (Fig. 3) otherwise the top edge with next nearest mid-point is checked(Fig. 4).

The above steps are repeated for all positions of S, i.e., all distinct values of y-coordinates. The last position of S will generate only one top edge which is matched with side B of R.

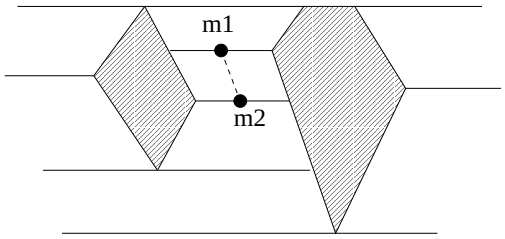

Figure 3: Matching of top and bottom edges

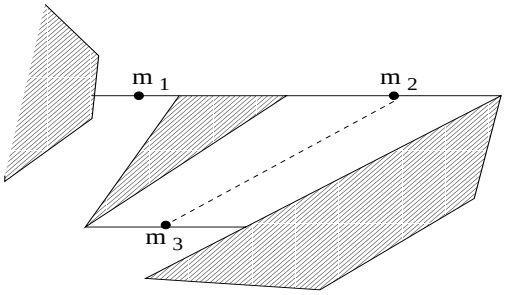

Figure 4: Matching of bottom edge with the next best choice for top edge

Lemma 1 *The trapezoidal decomposition algorithm for n polygonal vertices is* $O(n \log n)$.

Proof : Follows from the standard sweep line techniques [10]. □.

Dual Graph Generation : The trapeziums are stored in two arrays A and B. Array $A(B)$ contains the set of trapeziums sorted in order of the y-coordinates of their top(bottom) edges respectively.

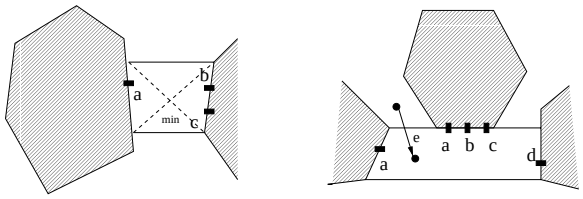

Figure 5: Computation of (a) vertex constraint, and (b) edge constraint

- For each trapezium i in A, create a vertex v of the directed dual graph D. The lists of incoming and outgoing edges of v are initialized and the constraint associated with v is computed as the minimum of the lengths of the two diagonals of i (Fig. 5a).
- Each vertex v has a list $Orgt(v)$ of original terminals which correspond to the terminals lying on the edges of i. These terminals of v are obtained by examining the polygonal edges of i (Fig. 5a).
- For the bottom edge $e_b(i)$ of each $i \in A$, obtain from array B the list l_i of trapeziums with top edges overlapping it, i.e., having same y-coordinate and intersection of the horizontal intervals. Create a directed edge from v corresponding to i to each of the vertices corresponding to trapeziums l_i.
- Each edge e in D has an associated constraint computed as the minimum of the lengths of the bottom and the top edge whose overlap is responsible for e (Fig. 5b).

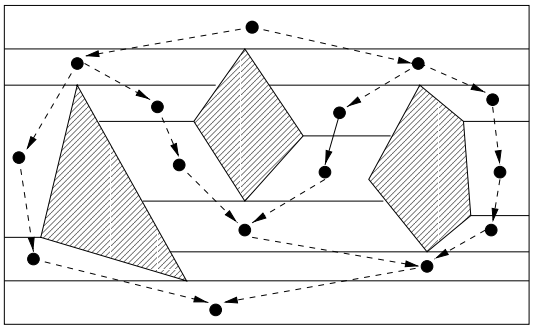

Figure 6: Dual graph for a typical example

Lemma 2 *The dual graph D is planar and acyclic.*

Proof : It follows easily from the fact that the input is planar and edges are directed from top to bottom. □.

4 Routing algorithm

4.1 Overview

A multi-pass greedy heuristic algorithm produces the routing of the nets concurrently by traversing the trapezoidal regions in the line-sweep order, alternately starting at the top and the bottom boundaries of the layout. One pass of the greedy heuristic essentially involves a traversal of the vertices of the graph D in a topologically sorted order. Initially, the list of nets whose terminals appear on the boundary of a trapezium are associated with the corresponding vertex in D. During the top-to-bottom pass, an unrouted net n_i is broadcasted from a vertex to all its successors. At any intermediate stage of a pass, the sweep line touches all the trapeziums at the same horizontal level. If a net n_j occurs in the netlists of two trapeziums on the sweep line, then net n_j is topologically routed through a path from each of the two trapeziums to their common ancestor in D, if that common ancestor has an original terminal for that net subject to constraint satisfaction. The nets which are left unrouted in one pass are reconsidered during the next pass where the sweep is in the reverse direction.

Local congestion is avoided by assigning a capacity constraint to the edges of the trapeziums. At any stage of the routing algorithm, a net is passed on from one trapezium to its neighbor, only if the density of the edge between the two trapeziums is less than its capacity. However, capacity constraints on the trapezoidal edges may be inadequate in modeling congestion. There are counter-examples where capacities of some additional cutlines have to be included, hence the need for the vertex constraints to take into account the smallest constriction in a trapezium.

Algorithm for routing by traversal of dual graph

Input: A directed graph $D = (V, E)$ which is the dual graph resulting from the the trapezoidal decomposition; each vertex v having a list $Orgt(v)$ of terminals.

Output: Global routing path in D for each net with constraints of edges and vertices satisfied.

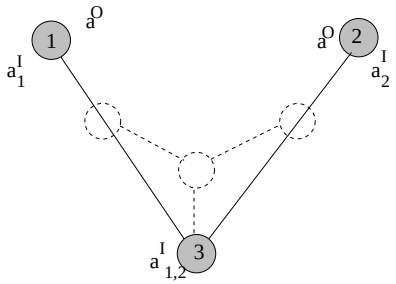

Figure 7: Original and inherited terminal lists for net a

Method:

For each vertex v, there is a list $Inht(v)$ of terminals of nets which it inherits from its predecessors as the traversal proceeds from top to bottom. For each terminal t (of net a, say), there is a list $route(t)$ consisting of those vertices having original terminals belonging to the same net a, which have been already connected to t by some path. Whether a terminal is associated with a vertex v in $Orgt(v)$ ($Inht(v)$), is indicated by superscript O (I). For example, a terminal of net a inherited by vertex 1 is denoted by a_1^I. In Fig. 7, suppose initially for net a, there are original terminals of a in vertices 1 and 2; during the forward pass, terminals of a in other predecessors are inherited into $Inht(v_1)$ and $Inht(v_2)$ respectively and then $route(a_1^I)$ and $route(a_2^I)$ contains 1 and 2 respectively. Next, if there exists a routing path between vertices 3 and 1 as well as 3 and 2, then terminals of net a (original and inherited) in 1 and 2 get connected through 3, and the labels 1 (for a_1^I) and 2 (for a_2^I) are inserted into $route(a^I)$ respectively.

The traversal is performed by considering the vertices in topologically sorted order first from top to bottom followed by reverse mode from bottom to top. The terminals, both original and inherited, in each vertex is broadcast to all its successors. In the forward pass, the routing path is however determined in reverse traversal mode. This should not be confused with backward pass. Symmetrically in the backward pass, the actual routing will be done in forward traversal mode.

- Step 1 Topologically sort V of digraph D using depth-first search.
- Step 2 For each vertex v_i in the topological order do -

  For each net a do -

  - IF (terminal t of a exists in $Inht(v_i)$) then
    - IF ((original terminal of a exists in $Orgt(v_i)$) OR (there is multiple inheritance from more than one predecessors but intersection of route list of the inherited terminals is empty))
      - then call RoutBack(v_i, t).
      - IF successful, update the routing path in backtrack mode; insert union of route lists of inherited terminals of net a in the route list of the new inherited terminal for a(Fig. 8) ELSE insert the original terminal to the inherited list; for multiple inheritance keep the route list(s) of the inherited terminal(s) whose capacity constraint(s) is(are) not saturated yet.
    - ELSIF (there is intersection in the route lists)
      //no need to route,
      //already in same cluster
      insert union of route lists of inherited terminals of net a in the route list of the new inherited terminal for a
      // they are already clustered
    - ELSE
      //no original terminal
      //no multiple inheritance
      keep the inherited terminal intact
    - ENDIF

    ENDIF
  - Pass $Orgt(v_i)$ and $Inht(v_i)$ to successors of v_i;
- Step 3. (Reverse mode) Proceed exactly as step 2 processing the vertices in reverse topological order. Instead of successors now pass the original and inherited terminals to the predecessors. Also process only the nets which are still unrouted.

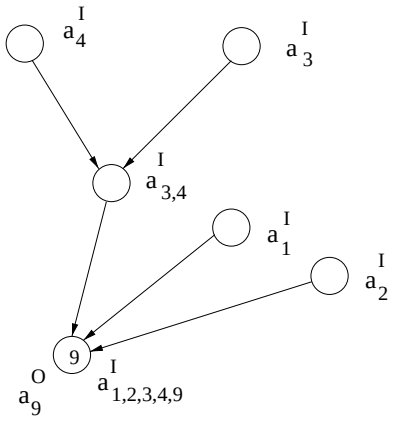

Figure 8: Routing requirements in backtrack mode

Routing in Backtrack mode

proc RoutBack(v_i, t)
Input: Vertex v_i and terminal t.
Output: A routing path for the net a of t from v_i to a vertex which has a terminal of net a in its original terminal list. The routing path has to meet the vertex and edge constraints.

Steps: Consider the set V' of predecessor vertices of v_i which has a terminal of net a (original or inherited).

- 1 For all vertex $v_j \in V'$ having terminal of net a in its original terminal list, and the edge (v_j, v_i) and vertex v_j meets the routing constraints, append this edge in the temporary route path for this net. Finally put all the edges from temporary route path to actual route path of this net and return TRUE.

- 2 Get the x-median of all the terminals of the net a. for each vertex $v_m \in V'$ in the decreasing order of distance of the mid-point of its bottom-edge from the x-median DO

  - IF (edge (v_m, v_i) meets the routing constraint, push the edge in a temporary route path for net a and call RoutBack(v_m, t)) return TRUE;

- 3 return FALSE.

Remark: We call this routine for a vertex iff it has an original terminal of net a or has multiple inherited list of terminals, the intersection of whose route list is null. For the case of two inherited terminal lists from predecessors, there is a recursive call with each predecessor having terminals in *Inht* whose route list has null intersection with that of other inherited terminals. If the routing is successful, the vertex v_i is added to the route list of the inherited terminals for this net which is then propagated back till a vertex with original terminal of a is reached. In other words, vertex v_i becomes a Steiner vertex. If there are more than two inherited lists, a set of inherited lists whose route lists are mutually exclusive is determined and routing through each path is done.

Theorem 1 *Each forward and backward pass of the greedy topological routing method above runs in $O(nT)$ time where there are n trapeziums and T terminals in all.*

Proof : The graph D is planar. In a single pass, any edge is traversed once during broadcast and then once more during backtrack, thus the pass requires polynomial time. Computing the x-median is done in linear time at the beginning and with the help of pointers, it is updated in constant time during the pass as each original terminal is connected. It may also be noted that the edges of trapeziums at any position of the sweep line and hence the corresponding dual edges in D are already sorted during trapezoidal decomposition. □

The method being greedy in nature, the optimality of path lengths as well as routability however cannot be provably guaranteed.

5 Performance

Since standard benchmarks for this problem are not known to exist, floorplans containing polygonal obstacles are generated to test our algorithm. First, our algorithm for arbitrary generation of problem instances is outlined briefly.

5.1 Generation of a polygon within a given rectangle

Generating convex polygon within a closed rectangular area involves two major steps. Firstly, required number of non-overlapping rectangles generation, and secondly, within each such generated rectangle, generation of convex polygon. In the second step, a number of random points are generated within a rectangle, and the convex hull of those points are found out. The quick rejection algorithm is used to determine the overlapping rectangles and the convex hull is found using Graham's Scan method [10]. Following are the elaborated steps of the algorithm:
proc **genrect(N)**;

1. for i in 1 to N do
2. produce vertices of rectangle within boundary limit and following size constraints;
3. for each R in global rectangle array GR do
4. if (R does not overlap with new rectangle NR)
5. add R to GR.

proc **genpoly()**

1. for each R in GR do
2. generate random point set S within boundary of R;
3. find the convex hull P of S;
4. add P in global polygon array GP.

5.2 Generation of terminals on polygons

After the polygon generation is complete, the global polygon array GP is iterated and terminals or net-points are put (generated) on each polygon according to a minimum bound, currently it is 2. The net-points are generated such that, no two net-points on same polygon comes from same net. The algorithm is:

proc **gennetpt()**

1. for each polygon P in GP do
2. for (random number of times ≥ 2) do
3. get arbitrary net N;
4. generate net-point NP of N on P;
5. if (N not already in P)
6. put NP on P;
7. for each polygon P in GP do
8. if (number of net-points on $P < 2$)
9. while (number of net-points on $P < 2$) do
10. get arbitrary net N;
11. generate net-point NP of N on P;
12. if (N not already in P) then put NP on P.

5.3 Results

Our algorithm yields a valid solution very fast for most examples of moderate size in only two passes — one top-to-bottom sweep followed by one bottom-to-top sweep. The paths generated are obstacle-free but not necessarily shortest distance. Thus this simple heuristic produces solution to the topological routing problem avoiding routing deadlocks caused by shortest path criterion or net ordering in sequential approaches. Table 1 gives a summary of our experimental results. The program was run on a Sparc 5 machine and was implemented in the C language. The polygonal obstacles, nets and terminals were generated arbitrarily and hence with increasing floor size, the number of terminals varied though the number of nets remained constant. The percentage of success increased when floor size was increased keeping the number of distinct nets constant. This is expected as with increasing floor size more routing area became available. In only one case, the table shows an anomaly where number of polygons were 30 and number of nets were 15. But this again happened because the testcases were generated arbitrarily.

Table 1: Experimental Results

Number of			Floor	% of	CPU Time
Polys.	Nets	Terms.	Size	Success	(sec)
10	5	32	400 x 300	100.00	.110
10	10	53	400 x 300	100.00	.100
10	20	116	400 x 300	95.00	.200
10	20	123	500 x 400	100.00	.210
20	10	87	400 x 300	100.00	.270
20	20	161	400 x 300	60.00	.430
20	20	170	500 x 400	95.00	.690
20	40	288	400 x 300	77.50	1.110
20	40	295	500 x 400	77.50	.960
20	40	303	700 x 500	95.00	1.490
30	15	136	400 x 300	80.00	.880
30	15	143	500 x 400	86.67	.880
30	15	130	700 x 500	73.33	.640
30	30	240	400 x 300	100.00	1.180
30	60	448	400 x 300	43.33	2.110
30	60	436	700 x 500	91.67	2.480
40	20	186	400 x 300	85.00	1.270
40	20	193	500 x 400	100.00	1.180
40	40	323	400 x 300	80.00	1.750
40	40	341	700 x 500	95.00	4.070
40	40	339	800 x 700	100.00	2.000
40	80	599	400 x 300	51.25	2.560
40	80	606	700 x 500	76.25	2.680
40	80	626	1200 x 1000	95.00	5.880
50	25	231	400 x 300	48.00	2.160
50	25	232	700 x 500	76.00	1.630
50	25	237	800 x 700	100.00	1.660
50	50	411	400 x 300	54.00	2.500
50	50	396	700 x 500	82.00	3.620
50	50	406	1200 x 1000	100.00	3.470
50	100	741	400 x 300	40.00	3.990
50	100	751	700 x 500	68.00	5.550
50	100	719	1200 x 1000	94.00	53.120

6 Concluding Remarks

In this paper, we have tackled the problem of topological routing of all the nets simultaneously, instead of one Steiner tree after another. The routing was done in the presence of pre-routed nets or groups of circuit blocks. An efficient graph traversal based greedy heuristic method is proposed. The results obtained thus far for the implementation are very encouraging. Layer assignment algorithm for the global routing paths of the nets is being studied currently.

References

[1] W. W. Dai, T. Dayan and D. Staepleare, "Topological routing in SURF: Generating a rubber-band sketch," Proc. 28th Design Automation Conference, pp. 39-41, 1991.

[2] W.W. Dai, R. Kong and J. Jue, "Rubber band routing and dynamic data representation," Proc. 1990 International Conference on Computer-Aided Design, pp. 52-55, 1990.

[3] S. Haruyama, D. F. Wong and D. S. Fusell, "Topological channel routing" IEEE Trans. on CAD, 10(10): 1117-1119, 1992.

[4] C. E. Leiserson and F. M. Maley, "Algorithms for routing and testing routability of planar VLSI Layouts," Proc. 17th Annual ACM Symposium on Theory of Computing, pp. 69-78, 1985.

[5] K. F. Liao, M. Sarrafzadeh and C. K. Wong, "Single-layer global routing," IEEE Trans. on CAD/ICS 13(1): 38-47, 1994.

[6] A. Lim, S. Sahni and V. Thanvantri, " A fast algorithm to test planar topological routability," Proc. Int'l Conference on VLSI Design, pp. 8-12, 1995.

[7] M. Marek-Sadowska and T. T. -K. Tarng, " Single-layer routing for VLSI : Analysis and algorithms," IEEE Trans. on Computer Aided Design, CAD-2(4): 246-259, 1983.

[8] N. A. Sherwani, S. Bhingrade and A. Panyam, *Routing in the Third Dimension*, IEEE Press, 1995.

[9] M. Garey and D. S. Johnson, *Computers and Intractability: A complete guide to NP-completeness*, Freeman, 1979.

[10] T. Cormen, C. E. Leiserson and R. L. Rivest,*Introduction to Algorithms*, MIT Press, 1990.