

Via-Aware Global Routing for Good VLSI Manufacturability and High Yield *

Yang Yang, Tong Jing, Xianlong Hong, Yu Hu
Computer Science & Technology Dept.
Tsinghua Univ.
Beijing, P. R. China

Qi Zhu Xiaodong Hu, Guiying Yan
EECS Dept. Inst Applied Math
UC at Berkeley CAS
U. S. A. Beijing, P. R. China

Abstract

CAD tools have become more and more important for integrated circuit (IC) design since a complicated system can be designed into a single chip, called system-on-a-chip (SOC), in which physical design tool is an essential and critical part. We try to consider the via minimization problem as early as possible in physical design. We propose a routing method focusing on minimizing vias while considering routability and wire-length constraint. That is, in the global routing phase, we minimize the number of bends, which is closely related to the number of vias. Previous work only dealt with very small nets, but our algorithm is general for the nets with any size. Experimental results show that our algorithm can greatly reduce the count of bends for various sizes of nets while meeting the constraints of congestion and wire-length.

1. Introduction

Nowadays, the seriousness about effects of very deep-submicron (VDSM) technology has led to a greater and greater reliance on CAD tools in VDSM physical design [1]-[3]. Global routing is an essential part of physical design. It usually includes multiple optimization goals. Congestion and wire-length have long been focuses of research [4]-[7]. Besides, interconnection delay [8]-[9] is an important issue for high-performance routing.

While very large scale integrated circuits (VLSI) feature size continues to shrink in the VDSM regime, the number of vias becomes a critical issue, which has a great effect on the circuit performance, layout size

and yield rate. In order to reduce the yield loss by via failure, [10] proposed a redundant-via enhance maze routing algorithm. On the other hand, minimizing the count of vias is also an effective method to reduce the yield loss, and moreover, it can meanwhile improve the circuit performance and layout size. Traditionally, via minimization is done in the detailed routing phase or layer assignment [4], [11]-[12]. However, when the size of design features keeps decreasing and the complexity of circuits keeps increasing, it will be more flexible and effective to minimize the number of vias as early as in the global routing phase.

In the global routing phase, bends denote both the corner points and the Steiner points in a Steiner tree. A bend usually imply a switching of layers, therefore cause the use of more vias. Also more bends require more routing resources due to the larger via pitch and reduce reliability [13]-[14]. Some papers focused on bend minimization problem in global routing phase [13]-[16]. [15] introduced a four-bend routing for 2-terminal nets. [13] decomposed multi-terminal nets into several 2-terminal nets, and presented an algorithm to solve 2-terminal nets. [14] mainly used "Z-edge" to bound the number of bends. [16] used "L-shaped" and "Z-shaped" pattern routing to reduce the number of bends for two-terminal nets.

Different from these previous works dealing with some specific models or small nets, the main contribution of this paper is to propose a general method to effectively reduce the number of bends for nets with any size, while still keeping the qualities of congestion and wire-length.

The rest of this paper is organized as follows. In Section 2, we formulate the problem and introduce some basic definitions and theorems. In Section 3, a new algorithm is proposed. The experimental results are shown in Section 4. And in Section 5, we conclude our work and introduce the future work.

* This work was supported in part by the NSFC under Grant No.60373012, the SRFDP of China under Grant No.20020003008, and the Hi-Tech Research and Development (863) Program of China under Grant No.2004AA1Z1050.

2. Preliminaries

2.1. Problem formulation

Global routing problem is normally formulated as follows: the entire routing region is tiled into an array of rectangular "global routing cells (GRCs)". Based on these GRCs, a "global routing graph (GRG)" $G(V_G, E_G)$ is constructed, where a vertex $v \in V_G$ corresponds to a GRC, and an edge $e \in E_G$ corresponds to the border between two GRCs. Since there is only finite routing space on the borders, each edge has a capacity. A set of nets $N = \{N^1, N^2, \dots\}$ is given to represent the interconnection between circuit elements. Each net $N^i \in N$ contains a set of terminals $V^i = \{v_0^i, v_1^i, \dots\}$ which are represented by the corresponding vertices in V_G . In the global routing phase, every net is routed by a Steiner tree.

In the routing phase, the number of wires routed across an edge is designated as the *demand* of the edge. If for an edge e , $demand(e) > capacity(e)$, then the edge e is called *over-congested*, or we say there is an *overflow* [5] on edge e . In our model, we do not allow over-congested edges.

We assign additional cost to the route once it reaches 80% capacity [17], by defining the weight of an edge $e \in E$ as follows.

$$weight(e) = (1 + k \times congest(e)) \times length(e)$$

$$congest(e) = \max(0, \frac{demand(e) - 0.8capacity(e)}{capacity(e)})$$

$$\text{The weight of a route } r \text{ is } \sum_{e \in r} Weight(e).$$

This paper focuses on the global routing problem with the objective of reducing the number of bends, eliminating over-congested edges, and satisfying the constraint of wire-length, which can be formulated as follows.

$$\min \sum_{\forall N^i \in N} b^i$$

$$\text{s.t. } demand(e_j) \leq capacity(e_j), \quad \forall e_j \in E_G$$

$$\sum_{\forall N^i \in N} w^i \leq (1 + \alpha)W_0$$

In the above formulas, b^i and w^i respectively denote the number of bends and the weight of the routing tree of net N^i ; α is an adjustable parameter; W_0 denotes a standard weight value. We use the total weight value got by SSTT [7], an efficient global router focusing on congestion and wire-length, as W_0 in our algorithm.

2.2. Definitions and theorems

2.2.1. Definition 1, STST. The definition of STST in our paper is extended from [18]. A special kind of Steiner tree consisting of a single horizontal/vertical line segment and vertical/horizontal line segments is called STST-H (single horizontal trunk Steiner trees)/STST-V (single vertical trunk Steiner trees). A STST is either a STST-H or a STST-V, as shown in Figure 1. The single horizontal/vertical line segment in a STST-H/STST-V is called "trunk", and the vertical/horizontal line segments are called "branches".

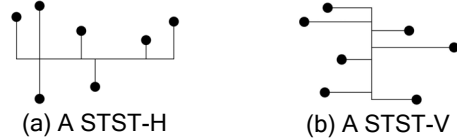


Figure 1. Examples of the STST

2.2.2. Definition2, MSTST. A MSTST (*minimal single trunk Steiner tree*) is the STST with minimal total wire-length among all STST's. A MSTST-H / MSTST-V (*minimal single horizontal / vertical trunk Steiner tree*) is the STST-H / STST-V with minimal total wire-length among all STST-H / STST-Vs.

A MSTST must be either a MSTST-H or a MSTST-V. It should be noticed that in most cases a MSTST is not a SMT (Steiner minimal tree). In worst case, a MSTST may have much longer wire-length than a SMT.

2.2.3. Definition3, path and soft path. In this paper, a *path* in a Steiner tree denotes a sequence of adjacent segments with no terminals or Steiner points sharing two adjacent segments (however, the endpoints of the line may be terminals or Steiner points).

[5] defined the "soft edge" of connecting 2-terminal nets. We extend this definition to the "soft path" in a Steiner tree. A soft path is a path connecting two terminals or Steiner points $v_i(x_i, y_i), v_j(x_j, y_j) \in V$, so that: 1. $x_i \neq x_j$ and $y_i \neq y_j$; 2. its length l_{ij} is fixed; 3. the precise edge routing between v_i and v_j is not determined.

2.2.4. Theorem 1, location of the trunk in a MSTST. In a STST, we denote the number of segments above (or to the right of) the trunk as n_a , below (or to the left of) the trunk as n_b , and the number of terminals on the trunk but not on any of the segments as n_o . Then the trunk in a MSTST must have the property: $|n_a - n_b| \leq n_o$ (Examples are shown in Figure 2). The proof is omitted due to the paper length limitation.

This property helps us to find out the trunk location of a MSTST quickly in our algorithm. Also it shows that there may exist more than one MSTST for a set of terminals, as shown in Figure 2.

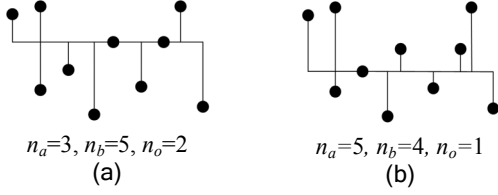


Figure 2. Examples of two different MSTST's routing the same set of terminals

2.2.5. Theorem 2, the number of bends in a MSTST. A unique character of a MSTST is that in a STST connecting n points, the number of bends must be no more than n . The proof is omitted due to the paper length limitation.

In our algorithm, we use MSTST's as subtrees to construct Steiner trees for nets, because MSTST's have relatively small number of bends compared with other forms of subtrees.

2.2.6. Theorem 4, subtree concatenation = finding spanning trees on hypergraphs. "Concatenation" and "hypergraphs" are defined in [19]. It is proved in [19] that the problem of finding a MST in a hypergraph H is equivalent to solving the subtree concatenation problem for the net N^f . Finding the MST in a hypergraph problem (MSTHG) is NP-hard when the hypergraph contains edges of cardinality four or more. The examples are shown in Figure 3 and 4.

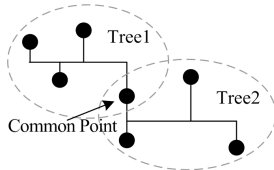


Figure 3. An example of tree concatenation

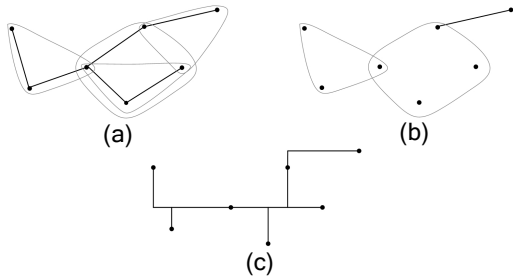


Figure 4. Examples of (a) a hypergraph; (b) a spanning tree in the hyper graph; (c) the corresponding Steiner tree in the subgraph

2.2.7. Definition 6: overlapping of two edges. We call two edges in a hypergraph *overlapping* if and only if the two corresponding subtrees in the subgraph have more than one point in common.

3. Single-trunk-subtrees Concatenating Global Routing (SCGR) Algorithm

By utilizing the definitions and theorems introduced above, an algorithm called "single-trunk-Steiner-tree concatenating global routing (SCGR)" is proposed. The objective of this algorithm consists of three parts: to minimize the number of bends, to eliminate the over-congested edges, and to satisfy the wire-length constraint.

3.1. Outline of the SCGR algorithm

The overall pseudo code of the SCGR algorithm is illustrated in Figure 5.

```

Step1: For every multi-terminal net
        Construct a set of subtrees for the net;
Step2: Pre-estimate congestion map;
Step3: For every 2-terminal net
        Route the net;
        Update congestion map;
Step4: For every multi-terminal net
        Route the net by concatenating its subtrees;
        Update congestion map;
Step5: Solve the remaining congestion problem;

```

Figure 5. Pseudo code of the SCGR algorithm

The SCGR algorithm contains five steps. In Step 1, we selectively construct subtrees which are flexible MSTST's for each multi-terminal net, and a soft path for each 2-terminal net. In Step 2, we pre-estimate a congestion map [17] by probability calculation method. In Step 3, we route the 2-terminal nets by fixing the soft path to be a solid path with minimum bends and satisfying the weighted constraint. In Step 4, we first fix subtrees according to their weights and then concatenate subtrees to form a tree with minimum bends under a weighted constraint. The subtree concatenation problem is equivalent to finding a minimal spanning tree in a hypergraph (MSTHG). We formulate the MSTHG problem as an integer programming (IP) problem. In Step 5, if there is any tree of which more than 30% is in the over-congested area, we re-fix and re-concatenate it under the updated congestion map. After that, we use the rip-up and reroute technique to solve the over-congested area. Since we keep updating the congestion map during Step 3 and 4, and carefully consider the congestion problem during the first four steps, usually only a little over-congested area remains for Step 5.

3.2. Details of the steps in SCGR

3.2.1. Step 1, subtree construction. In this step, for every net N^i with a set of terminals $V^i = \{v_0^i, v_1^i, \dots\}$, we construct a set of $m_i (m_i \geq 1)$ subtrees, each of which connects a subset vertices $V_k^i \subseteq V^i, k = \{1, \dots, m_i\}$. These subtrees are constructed under following three constraints.

Constraint 1: they must be MSTST's.

Constraint 2: all the terminals in V^i must be in at least one of the subtrees, i.e. $\bigcup_{k=1}^{m_i} V_k^i = V^i$.

Constraint 3: they should not have bad wire-lengths. Here, we do not strictly constrain wire-lengths of subtrees, because a subtree with not very good wire-length may probably be part of a good routing tree. More strict and exact constraint about wire-length of a whole tree will be in Step 4. This constraint helps to reduce the number of the subtrees in the early step, so as to improve the efficiency of the algorithm.

3.2.2. Step 2, pre-estimate the congestion map. We calculate the pre-estimated congestion map by the subtrees and the soft paths constructed in Step 1. We calculate the *probabilistic demand* (PD) of every soft path in subtrees and 2-terminal nets using the similar method as [14]. Besides, we calculate *probabilistic usage* (PU) of every subtree for every multi-terminal net. Then, we get the pre-estimated congestion map by adding up the demand of edges of all the nets. The concrete method is described below.

For a soft path p , we calculate probabilistic demand similarly as in [14] as below. In order to limit the number of bends, we only allow the path to be a "Z"-path. For each soft path p , we define a *probabilistic demand* $PD_p(e)$ to indicate the possibility that the soft path p will take a route across the edge $e \in E_G$. Suppose the two ends of p are $v_i(x_i, y_i)$ and $v_j(x_j, y_j)$, we can conclude that there are totally $T = |x_i - x_j| + |y_i - y_j|$ "Z"-paths between the two ends. We assume the soft path p has the uniform probability to take any one of "Z"-paths. So an edge has a probabilistic demand $PD_p(e) = C(e)/T$ from this path p , where $C(e)$ is the number of the "Z"-paths that passing through the edge. Then we can calculate the total probabilistic demand of an edge e from a subtree st by $PD_{st}(e) = \sum_{p \in st} PD_p(e)$.

For every multi-terminal net $N^i \in N$, which has a set $STSet^i$ of subtrees constructed in Step1, we calculate approximate probabilistic usage of each subtree $st \in STSet^i$ by the equations below, where $TC(st)$ is the terminal count of st , and t stands for a terminal.

$$PU(st) = \frac{1}{TC(st)} \times \left(\sum_{\forall \text{terminal } t: t \in st} \frac{1}{\sum_{st' \in STSet^i} C(st', t)} \right)$$

$$C(st', t) = \begin{cases} 1 & t \in st' \\ 0 & t \notin st' \end{cases}$$

Finally, the congestion map can be described by the demand of every edge $e \in E_G$ as *demand*(e)

$$= \sum_{\forall N^i \in N_m} \sum_{\forall st \in STSet^i} PD_{st}(e) \times PU(st) + \sum_{\forall N^j \in N_t} PD_{p^j}(e),$$

where $N_m = \{N^i \mid N^i \text{ is multi-terminal net, } N^i \in N\}$, $N_t = \{N^j \mid N^j \text{ is 2-terminal net, } N^j \in N\}$; $STSet^i$ is the set of subtrees constructed for the multi-terminal net N^i , and p^j is the path constructed for the 2-terminal net N^j constructed in Step 1.

3.2.3. Step 4, route multi-terminal nets by subtree concatenation. Geo-Steiner algorithm [19] constructs the Steiner minimal tree by FST generation and FST concatenation. We borrow its idea in some parts of our algorithm. But we use the idea in a different way, because our objective is quite distinct from Geo-Steiner: rather than constructing a Steiner tree with minimal wire-length for one net, we focus on routing a large number of nets in a routing region, considering the performance of all the nets as a whole. We construct MSTST's in Step 1, which are a particular kind of FST's, for the sake of reducing bends; moreover, we make the MSTST's with soft paths, slidable trunks and Steiner points so as to have more flexibility. Then in Step 4, we first fix the MSTST's according to congestion and wire-length constraints, and then concatenate them under the objective function about the number of bends and the constraints of congestion and wire-length.

Details of the subtree concatenation method in Step 4 are described below.

In order to do subtree concatenation for every multi-terminal net $N^i \in N$, we solve the corresponding MSTHG in the hypergraph H^i according to Theorem 4. The MSTHG is solved by setting up an integer programming (IP) formulation in our algorithm.

A $|E_H^i|$ -dimensional binary vector is denoted by x : each element x_e has value 1 if the hyperedge $e \in E_H^i$ (corresponding to a subtree in the induced subgraph) is chosen to be part of the MST in the hypergraph H^i and 0 otherwise. The IP formulation is firstly formulated as follows.

$$\min b \cdot x^T \quad (1)$$

$$\text{s.t.} \quad \sum_{e \in E_H^i} (|e| - 1)x_e = |V| - 1 \quad (2)$$

$$\sum_{e \in E_H^i: |e \cap S| \geq 1} (|e \cap S| - 1)x_e \leq |S| - 1, \quad \forall S \subset V^i, |S| \geq 2 \quad (3)$$

$$w \cdot x^T \leq (1 + \alpha)w_0 \quad (4)$$

The b and w are both $|E_H^i|$ -dimensional binary vectors: each element b_e equals to the number of bends in the subtree corresponding to the hyperedge $e \in E_H^i$; each element w_e equals to the weight of the subtree corresponding to the hyperedge $e \in E_H^i$; $(1+\alpha)w_0$ is the weighted constraint of the net, where w_0 is the weight of the routing tree got by SSTT [7], and α is an adjustable parameter.

The major difficulty in solving the IP is the exponential number of constrains in (3). We take full advantage of the special properties of the subtrees, and finally cut down the number of constraints to only $O(|E_H^i|^2)$. The IP formulation used in our algorithm is:

$$\min b \cdot x^T \quad (5)$$

$$\text{s.t.} \quad \sum_{e \in E_H^i} (|e|-1)x_e = |V|-1 \quad (6)$$

$$\begin{aligned} & (\min y_p \min y_q + \max y_p \max y_q)x_p \\ & + (\min y_p \min y_q + \max y_p \max y_q)x_q \\ & \leq (\min y_p \min y_q + \max y_p \max y_q) \\ & + (\min y_p \max y_p + \min y_q \max y_q), \\ & 1 \leq p < q \leq |E_H^i| \quad (7) \end{aligned}$$

$$x_p + x_q \leq 1, 1 \leq p < q \leq |E_H^i| \& p, q \text{ overlap} \quad (8)$$

$$w \cdot x^T \leq (1+\alpha)W_0 \quad (9)$$

The $\max y_k$ and $\min y_k$ ($k=p, q$) in (7) respectively stand for the maximum y value and the minimum y value of the corresponding subtree in the subgraph.

The objective (5) is to minimize the total bends of the chosen hyperedges subjecting to four constraints. Equation (6) enforces the correct number and cardinality of hyperedges to construct a spanning tree. Constraints (7) and (8) have the same effect in our algorithm as (3), which is to eliminate cycles in every subset of V^i . Constraints (6), (7) and (8) operate together to ensure that the chosen hyperedges can construct a corresponding Steiner tree in the subgraph, while (9) is the weighted constraint.

There are totally $|E_H^i|(|E_H^i|-1)/2$ constrains in (7) and m constraints in (8), where m is the number of overlapping pairs of hyperedges and is $O(|E_H^i|^2)$.

The IP solving progress in our algorithm cost short running time, because we not only restrict the number of subtrees (equal to the dimension of x in IP formula) in Step 1, but also cut down the number of constraints in IP formula as explained above.

4. Experimental results

The SCGR algorithm has been implemented in C language. We compare it with SSTT [7]. For fair

comparison, both programs are compiled and run on a SUN V880 workstation. MCNC and IBM benchmarks are used as the test cases in Table 1. Besides, we randomly generate two more test cases, by setting the percentage of multi-terminal nets and large nets.

Table 1. Description of test circuits

Test circuit	Grid size	Net count	Multi-terminal net count (percentage)	Large net ^① count (percentage)
test1	30×30	5000	2500 (50%)	1500 (30%)
test2	30×30	5000	3500 (70%)	2500 (50%)
c2	9×11	745	198 (26.6%)	4 (0.5%)
c5	16×18	1764	250 (14.2%)	95 (4.8%)
c7	16×18	2356	626 (26.6%)	23 (0.97%)
ibm01	134×135	11753	5481 (46.6%)	1751 (14.9%)
ibm02	79×78	18688	7865 (66.9%)	2776 (14.9%)
ibm07	234×235	44681	19272 (43.1%)	6694 (15.0%)
ibm08	245×245	48230	19002 (39.4%)	6251 (13.0%)

Table 2. Comparing SCGR with SSTT

Test circuit	$\alpha^{\text{②}} = 10\%$		$\alpha = 20\%$	
	Wire-length ^③	Bend count ^④	Wire-length	Bend count
test1	102.5%	79.3%	105.1%	72.8%
test2	104.8%	63.4%	109.5%	56.0%
c2	98.1%	96.0%	98.7%	95.3%
c5	98.3%	94.2%	99.3%	92.8%
c7	99.9%	93.8%	102.9%	89.1%
ibm01	105.7%	91.7%	112.5%	88.5%
ibm02	103.4%	86.5%	107.1%	83.9%
ibm07	104.4%	94.8%	114.8%	89.3%
ibm08	104.6%	93.2%	110.5%	90.5%

Table 2 shows the comparison of routing result between SCGR and SSTT. It should be noted that both SCGR and SSTT completely eliminate over-congested edges. We constrain the wire-length of SCGR to be not more than 110% and 120% of SSTT, by setting the adjustable parameter α (explained in Section 2.1). Table 2 shows that SCGR greatly reduces the number of bends while meets this wire-length constraint: when $\alpha=0.1$, the number of bends decreases for 11.9% on the average while the wire-length increasing 2.4%; when $\alpha=0.2$, the number of bends decreases for 15.7% on the average while the wire-length increasing 6.8%. We also notice that, generally, more decreasing can be got

^① The number of nets with more than 5 terminals in every test case

^② α is a parameter used to constrain the ratio of wire-length of SCGR to SSTT (explained in Section 2.1)

^③ The ratio of total wire-length by SCGR to total wire-length by SSTT

^④ The ratio of total number of bends by SCGR to total number of bends by SSTT

for test cases with larger portion of multi-terminal nets and large nets.

Table 3 shows the running time of SCGR. It is shown that our algorithm is efficient enough considering the effective results and the scale of test cases. From experiments, we notice that the IP step takes the major portion of running time. So far, our program applies an existing IP solver called GLPK [20]. A shorter running time could be achieved when more efficient solver is used.

Table 3. CPU time

Test circuit	SSTT (s)	SCGR (s)	Percentage of IP process in SCGR
test1	21.84	75.210	89.1%
test2	26.20	98.580	92.3%
c2	0.52	1.56	74.5%
c5	1.20	3.37	78.9%
c7	1.77	3.59	71.7%
ibm01	35.54	110.13	84.5%
ibm02	22.17	114.64	83.2%
ibm07	215.72	948.32	84.1%
ibm08	463.61	1207.86	82.9%

5. Conclusions and further work

We propose a general bend minimization global routing algorithm in this paper. This algorithm focuses on the competing objectives of minimizing number of bends, eliminating congestion and satisfying wire-length constraint simultaneously. We apply flexible single-trunk Steiner trees, soft paths, pre-estimated and updated congestion map in our algorithm to approach a good global routing result. The experimental results show that this algorithm can greatly reduce the number of bends, especially for the nets with relatively more terminals.

We will continue studying on the performance-driven global routing in the future. We plan to integrate the consideration of interconnect delay in our SCGR algorithm.

6. References

- [1] J.K. Wee, P.J. Kim, *et al*, "An Effective Routing Methodology in the Era of 0.2 μ m and Beyond Technologies for Reducing the DRAM Design Cost", in Proc. AP-ASIC, 1999, pp. 392-395.
- [2] T. Watanabe, Y. Ohtomo, *et al*, "An Effective Routing Methodology for Gb/s LSI Using Deep Submicron CMOS/SIMOX Technology", in Proc. CICC, 1997, pp. 569-572.
- [3] M. Lavin and L. Liebmann, "CAD Computation for Manufacturability: Can We Save VLSI Technology from Itself?", in Proc. ICCAD, 2002, pp. 424 -431.
- [4] C. Chiang, M. Sarrafzadeh, *et al*, "Global Routing Based on Steiner Min-Max Trees", IEEE Trans. on CAD, 1990, pp. 1318-1325.
- [5] J. Hu and S.S. Sapatnekar, "A Timing-Constrained Algorithm for Simultaneous Global Routing of Multiple Nets", in Proc. ICCAD, 2000, pp. 99-103.
- [6] C. Albrecht, "Provably Good Global Routing by a New Approximation Algorithm for Multicommodity Flow", in Proc. ISPD, 2000, pp 19-25.
- [7] T. Jing, X.L. Hong, *et al*, "SSTT: Efficient Local Search for GSI Global Routing", J. of Compute Science and Technology (JCST), 2003, pp.632-639.
- [8] J. Huang, X.L. Hong, *et al*, "An Efficient Timing-Driven Global Routing Algorithm", in Proc. DAC, 1993, pp. 596-600.
- [9] J. Hu and S.S. Sapatnekar, "A Timing-Constrained Algorithm for Simultaneous Global Routing of Multiple Nets", in Proc. ICCAD, 2000, pp. 99-103.
- [10] G. Xu, L.D. Huang, D.Z. Pan, *et al*, "Redundant-Via Enhanced Maze Routing for Yield Improvement", in Proc. ASP-DAC, 2005, pp. 1148-1151.
- [11] C.C. Chang and J. Cong, "An Efficient Approach to Multilayer Layer Assignment with and Application to Via Minimization", IEEE Trans. on CAD, 1999, pp. 608-620.
- [12] M.C. Yildiz and P.H. Madden, "Preferred Direction Steiner Trees", IEEE Trans. on CAD, 2002, pp. 1368-1372.
- [13] J. D. Cho and M. Sarrafzadeh, "Four-Bend Top-Down Global Routing", IEEE Trans. on CAD, 1998, pp. 793-802.
- [14] J. Hu and S.S. Sapatnekar, "Performance Driven Global Routing through Gradual Refinement", in Proc. ICCD, 2001, pp. 481 - 483.
- [15] J.D. Cho, "Wiring Space and Length Estimation in Two-Dimensional Arrays", IEEE Trans. on CAD, 2000, pp. 612-615.
- [16] R. Kastner, E. Bozorgzadeh, *et al*, "Pattern Routing: Use and Theory for Increasing Predictability and Avoiding Coupling", IEEE Trans. on CAD, 2002, pp. 777-790.
- [17] R.T. Hadsell and P.H. Madden, "Improved Global Routing through Congestion Estimation", in Proc. DAC, 2003, pp. 28-31.
- [18] N.A. Sherwani, *Algorithms for VLSI physical design automation*, 3rd edition, Kluwer Academic Pub. Norwell, MA, USA, 1999, pp. 121-122.
- [19] M. Zachariasen, *The Rectilinear Steiner Tree Problem: A Tutorial*, Steiner Trees in Industries, Kluwer Academic Publishers, 2001, pp. 467-507.
- [20] <http://www.gnu.org/software/glpk/glpk.html>