FastRoute: An efficient and high-quality global router

Min Pan[1], Yue Xu[2], Yanheng Zhang[3] and Chris Chu[2]

[1]Synopsys, Inc.
[2]Iowa State University
[3]Cadence Design Systems, Inc.

***Abstract*—Modern large-scale circuit designs have created great demand for fast and high-quality global routing algorithms to resolve the routing congestion at the global level. Rip-up and reroute scheme has been employed by the majority of academic and industrial global routers today, which iteratively resolve the congestion by recreating the routing path based on current congestion. This method is proved to be the most practical routing framework. However, the traditional iterative maze routing technique converges very slow and easily gets stuck at local optimal solutions. In this work, we propose a very efficient and high-quality global router - FastRoute. FastRoute integrates several novel techniques: fast congestion-driven via-aware Steiner tree construction, 3-bend routing, virtual capacity adjustment, multi-source multi-sink maze routing and spiral layer assignment. These techniques not only address the routing congestion measured at the edges of global routing grids but also minimize the total wirelength and via usage, which is critical for subsequent detailed routing, yield and manufacturability. Experimental results show that FastRoute is highly effiective and efficient to solve ISPD07 and ISPD08 global routing benchmark suites. The results outperform recently published academic global routers in both routability and runtime. In particular, for ISPD07 and ISPD08 global routing benchmarks, FastRoute generates 12 congestion free solutions out of 16 benchmarks with a speed singifinicantly faster than other routers.**

I. Introduction

As the feature size of modern VLSI design continues to shrink and the on-chip communication becomes extremely complicated, the ascending circuit density poses greater challenges for VLSI routers. Modern designs are liable to congestion problems due to increasing on-chip communication, concentrated routing demands and limited routing resources. Designs with IP blocks usually create narrow channels which further increase the difficulty of routing. Routability has become a major issue for the large designs. Besides, rapidly growing problem size sets a stringent requirement on the speed of routers.

In order to tackle such a complex issue, the routing problem is usually solved by a two-stage approach: a global routing stage followed by a detailed routing one. Global routing works on abstracted tiles. It allocates the routing demand globally over the circuit area and guides the subsequent detailed routing to finish the track assignment and via creation. Although global routing neglects the routing details such as tracks and design rule check (DRC), it generates interconnect information very close to the final routing implementation and can be used for accurate estimation of interconnect topology, wirelength, congestion and timing.

In addition to routability issue, as the continuous shrinking feature size poses great difficulty on manufacture process. Routing is a key step to consider the design-for-manufacture/yield (DFM/DFY) during the design process. It would determine whether a layout would have high yield or not. Vias, one major source for circuit failure, have larger process variation that impacts the timing/yield of circuits in a less predictable way. Thus via minimization is another important goal for global routing.

Routing is one of the traditional VLSI design automation area, along with placement and synthesis. Hu and Sapatnekar [1] gave a detailed survey for global routing algorithms. Recently, the global routing algorithms have been improved significantly with the ISPD 2007 and ISPD 2008 global routing contests held successfully. In the ISPD 2008 invited paper "The Coming of Age of (Academic) Global Routing", Moffitt *et al.* [2] presented the recent progress in the global routing area.

There are two major categories of global routing approaches: concurrent and sequential. Concurrent approach tries to handle multiple nets simultaneously. Albrecht [3] proposed a multi-commodity flow approximation algorithm to solve the global routing problem. The flow technique is used to solve a linear programming relaxation of global routing. BoxRouter [4] employed a hybrid approach with the application of ILP to simultaneously handle multiple nets and achieved reasonably good runtime. However, evidence suggests that the integer linear programming based routers run much slower than the sequential routers. Sequential approach generally employs a rip-up and reroute (R&R) framework. It takes an initial routing solution and iteratively improves the solution one net at a time. In each iteration, a net passing through congested area is ripupped and rerouted to avoid the currently congested regions. The sequential approach has been proved to be very effective in practice and considerably faster than concurrent approach.

Most recently developed global routers employ this R&R strategy but proposed different techniques to improve solution quality or speed. Kastner *et al.* [5] proposed a pattern routing scheme by using L-shaped and Z-shaped patterns to speed up the routing. Hadsell and Madden [6] propose to guide the routing by amplifying the congestion map with a new congestion cost function. In ISPD 2007 global routing contest,

several routers (BoxRouter 2.0 [7], Archer [8], NTHU-Route [9] [10], NTUgr [11] and FGR [12]) employed a negotiation-based R&R approach which was introduced by PathFinder [13] and successfully applied to FPGA routing. The negotiation-based cost functions are used by maze routing to drive the nets away from consistently congested regions.

In both ISPD 2007 and ISPD 2008 global routing contests, 3-dimensional benchmarks include the costs on vias for performance evaluation to encourage the global routers to consider the via effect. There are two categories of 3D techniques. The first category tries to solve the 3D problem directly on the 3D routing grids, FGR [12] belongs to this category. The second category employs layer projection to transform the 3D routing problem into a 2D one. After solving the 2D problem, the 2D solutions are mapped to 3D ones by layer assignment. Almost all recent global routers (BoxRouter 2.0 [7], Archer [8], MaizeRouter [14], NTHU-Route [9] [10], NTUgr [11] and default algorithm in FGR [12]) belong to this category. Although theoretically the direct 3D technique should produce better solutions, in practice it is less successful in both solution quality and runtime than 2D routing with layer assignment [15].

In this work, we develop a very efficient and high-quality global router FastRoute to tackle the 3D global routing problem. FastRoute integrates novel techniques introduced in [16] [17] [18] [19]. Our key contributions are:

1. A carefully designed framework to perform 3D global routing effectively and efficiently.
2. A congestion-driven, via-aware Steiner tree generation technique to form good starting topologies for multi-pin nets.
3. A segment shifting technique to direct routing demand away from congested region by moving some tree edges without increasing wirelength.
4. A 3-bend routing technique to quickly explore the routing paths between a source pin and a sink pin with a balance between congestion reduction and control on the number of vias.
5. A multi-source multi-sink maze routing technique to reconnect two subtrees in a multi-pin net without fixing the end points on both subtrees.
6. A virtual capacity technique which is a systematic way to guide the maze routing to avoid congested regions.
7. A new adaptive cost function based on logistic function to direct 3-bend routing and maze routing to find less congested paths.
8. A spiral layer assignment technique to extend a 2D routing solution into its 3D counterpart.

Our first contribution is the FastRoute framework that coordinates the proper functioning of quite a few novel global routing techniques we propose. Although each new technique targets to improve the global routing quality, their cumulative effects could be counteractive. We study the interactions between the various global routing techniques and design the framework to maximize the improvement. The second and third contributions focus on the optimization of tree structure before any actual routing. They can improve the routing quality of nets in congestion free region and effectively reduce the runtime for the actual routing process. The fourth and fifth contributions propose two new routing techniques. While 3-bend routing offers a new degree of balance among congestion reduction, via generation and runtime, multi-source and multi-sink maze routing relaxes a major constraint on traditional maze routing and thus greatly improves the quality of global routing. The sixth and seventh contributions are enhancement techniques to further help global router to reduce congestion in a more efficient manner. The last contribution, the spiral layer assignment technique, is a representative of various layer assignment techniques proposed between 2007 and 2010.

This paper is organized as follows. Section II introduces the general model in global routing. Section III describes the framework while the key techniques and algorithms used in FastRoute are presented in Section IV. The experimental results are provided in Section V and we conclude in Section VI.

II. Global Routing Grid Model

During global routing, complex design rules are abstracted away and a design is captured in a grid graph. As illustrated in Fig. 1, each layer of the entire routing region is partitioned into rectangular regions called global cells, each of which is represented by one node in the grid graph. The boundary on each metal layer between two global cells is represented by one 3D grid edge in the grid graph on the specific layer. The capacity for a grid edge, i.e., c_e, is defined as the maximum number of wires that can cross the grid edge. The usage, i.e., u_e, is defined as the actual number of wires crossing the grid edge. The overflow o_e is defined as $\max(u_e - c_e, 0)$. In the 3D model, a via is defined as a segment of wire that vertically connects one metal layer to a neighboring layer.

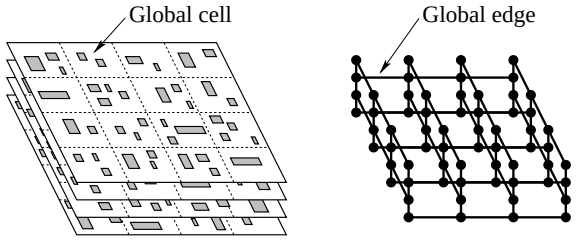


Fig. 1. Global cells and corresponding 3D global routing grid graph.

III. FastRoute Framework

FastRoute uses a sequential rip-up and reroute scheme to first solve the 2D version routing problem and later map the 2D solution to 3D by layer assignment. The flow of FastRoute is illustrated in Fig. 2.

First, we construct congestion-driven via-aware Steiner topologies for each net followed by segment shifting techniques. After the tree structures are decomposed into 2-pin nets, a pattern routing step using L-shape and Z-shape will initiate the routing solution. We initialize the virtual capacity based on current routing status. The virtual capacity technique is proposed to tackle the congestion problem in a systematic

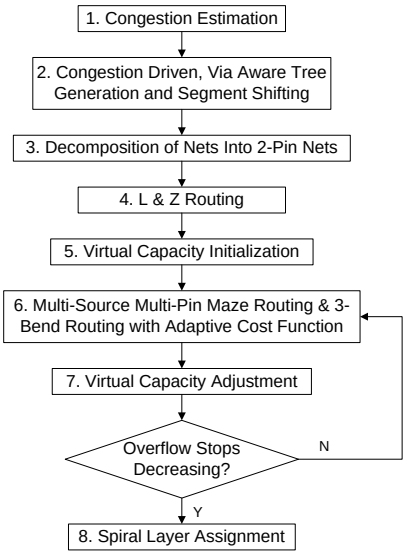


Fig. 2. FastRoute framework.

manner to guide the iterative rip-up and reroute stage with an adaptive cost function. During rip-up and reroute, we apply two major techniques: 3-bend routing and multi-source multi-sink maze routing to effectively avoid routing congestion and minimize via usage. Finally, after we obtain the 2D solution, we extend it to a full 3D solution by a spiral layer assignment algorithm.

This framework is the most practical one for global routing. Although we see solutions with shorter wirelength generated by full-3D concurrent approach like GRIP [20], that solution quality is achieved by impractically long runtime. The other framework like full 3D approach [12] or concurrent 2D approach [4] do not lead to better solution or shorter runtime. Breaking down 3D global routing problem into 2D routing problem plus layer assignment achieves has achieved the best balance between solution quality and runtime so far. FastRoute uses this framework. But more importantly routing techniques develped do not blindly improve one performance metric at a significant cost of others and they choose a suitable metric to improve in the right place. For congestion, before maze routing, FastRoute doesn't encourage too much detour because they may create artifical congestion hot spot. L/Z routing and 3-bend routing helps to eliminate easy overflow with short runtimme and leave difficult regions for maze routing. On the other hand, via count is properly controlled throughout the routing flow because FastRoute only rips-up net in congestion region so routing solution in any stage might be the final solution for one specific net, there might be no opportunity to optimize its routing topology again.

In FastRoute 4, we propose the routing algorithms with one important guideline: for the three performance metrics of wirelength, via count and routing speed, any technique either improve a single metric without degrading the other two or it improves two metrics with little sacrifice in the one left. Looking at the techniques used in topology generation, routing and convergence enhancement techniques, everyone of them helps to speed upthe routing process and improves wirelength and via. The major routing techniques, like congestion-driven via aware RSMT generation, 3-bend routing, layer assignment techniques, multi-souce multi-sink maze routing improves all three metrics. Other assisting techniques, like the initial congestion estimation and virtual capacity adjustment, uses little runtime but provides much accurate information to guide routing techniques to work more efficiently so their aggregate effect is still positive.

IV. FastRoute Techniques

A. Topology Generation

The first part of FastRoute framework is topology generation. Because FastRoute tries to avoid rip-up and reroute to reduce both wirelength and runtime, the initial tree topology has significant impacts. We find that the topology for each net is the determining factor for the quality of routing solution with regard to routability and the number of vias. So instead of just using rectilinear minimal spanning tree (RMST) or rectilinear Steiner minimal tree (RSMT), FastRoute generates tree topologies that greatly reduce congestion and vias.

1) Congestion Estimation: Before we can construct Steiner tree to help reduce the routing congestion, we need a congestion map to start with. Since this is the first shot and we are going to update the congestion map in later stages, we are aiming at a very fast but fairly good congestion estimation technique.

First, we generate the Steiner trees for all the nets using FLUTE [21] [22]. FLUTE is a very fast and accurate rectilinear Steiner minimal tree algorithm. It generates optimal RSMT for nets up to degree 9 and is still very accurate for nets up to degree 100, and is much faster than other RSMT algorithms. It is very suitable for our application. Second, after generating the Steiner trees, we break all Steiner trees into 2-pin nets. For every 2-pin net, we assign the demand to the grid edges in the 2D grid graph in the following manner. If the two pins of a net have the same x coordinates or y coordinates, we assign demand 1.0 to each grid edge on the straight line connecting the two pins. If the two pins of a net have different x and y coordinates, we assume two possible L-shape (sometimes called 1-bend) routings for it. For each grid edge on the two L-shape routings, we assign demand 0.5 to it. This gives us the very first congestion map. Finally, in order to make the congestion map more accurate, we perform a fast rip-up and reroute using L-shaped pattern routing. For each 2-pin net, we first remove its routing demand from the congestion map. Then we perform routing based on the current congestion map by taking the L-shape which accumulates least number of overflow. After a full round of L-shaped pattern routing for all 2-pin nets, we obtain a routing solution and its corresponding congestion information. We use it as the congestion map to guide the following congestion-driven via-aware Steiner tree generation.

2) Congestion-driven and Via-Aware Steiner Tree Generation: Traditionally, global routing just uses tree structure like RMST or RSMT while RSMT is becoming more popular due to its minimal wirelength to connect a multi-pin net together.

Because congestion and via minimization are not taken into account, simply adopting RSMT as the tree topology becomes insufficient. To address this problem, FastRoute generates routing topologies with consideration of reducing routing congestion and vias. The congestion-driven via-aware Steiner tree topology construction technique has great impact on the routing solution quality. It explores the solution space out of the scope of pattern routing and maze routing.

Routing congestion happens when there is more routing demand than the capacity of grid edges. We find that the congestion in horizontal direction and vertical direction can vary a lot. Due to different routing demand and capacity, it is very common that one direction is highly congested but the other direction is abundant of routing resources. If routing demand can be transferred between two directions, a lot of congestion problems can be easily resolved. However, we notice that neither pattern routing or maze routing is able to shift routing demand in between horizontal and vertical directions once the tree topology is fixed.

In addition, the local routing demand and resource always vary so that local congestion differs a lot. Pattern routing and maze routing have the ability to even out the routing demand but their effectiveness is limited because both techniques are applied only to 2-pin nets obtained after breaking the routing tree.

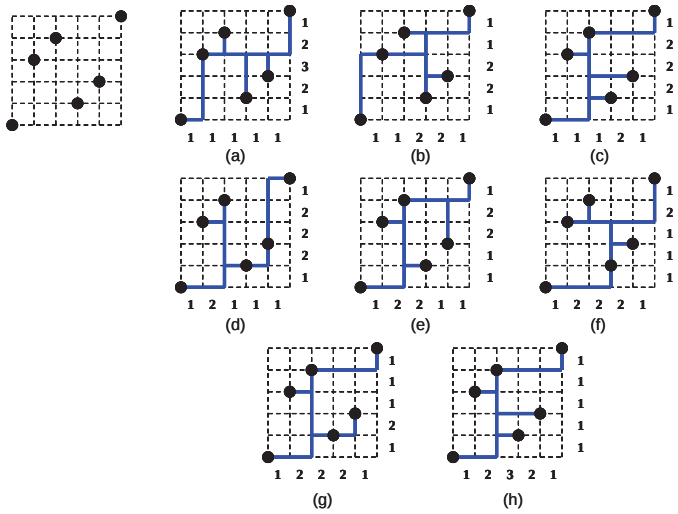


Fig. 3. Different Steiner trees topologies for a 6-pin net.

One important observation we make is that Steiner tree topologies can provide more flexibility to avoid routing congestion. For a multi-pin net, there are many different Steiner tree topologies to connect all the pins. Each topology corresponds to some specific routing demand and affects congestion differently. For example, in Fig. 3, we show 8 different Steiner tree topologies for a 6-pin net. For each topology, we only show one of the possible embeddings on the routing grids. The number below each column of grid edges is the routing demand over all the grid edges in that column. The number right to each row of grid edges is the routing demand over all the grid edges in that row. Although all these Steiner trees in Fig. 3 have the same wirelength, they have very different routing demand distribution, hence result in very different congestion. Therefore, we make use of this flexibility in topology and try to find good topology for each net in terms of congestion metric. For example, for the net shown in Fig. 3, if it is congested in horizontal direction, we want to pick topology (a) which has less routing demand in horizontal direction. On the contrary, if it is congested in vertical direction, (h) would be the best choice. In addition to transferring routing demand between two directions, shifting local routing demand in the same direction is also enabled by changing topology. Comparing topology (b) with (e), instead of having more routing demand in the 2nd row (from left) and 2nd column (from top) of grid edges as in (e), topology (b) has more routing demand in the 4th row and 4th column of grid edges. So whether using topology (b) or (e) depends on the congestion of these rows and columns of grid edges.

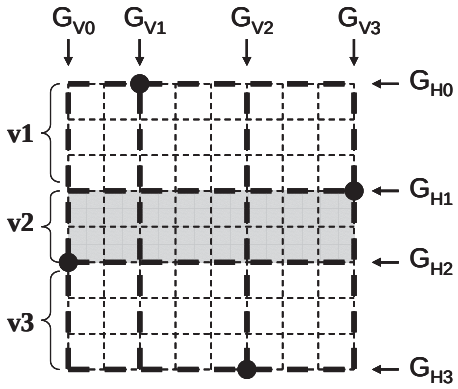
With this flexibility of topology in mind, our main idea is to construct good Steiner tree for each net according to the congestion map. We encourage using the topology with less routing demand in the congested direction and congested regions. To achieve this goal, we construct Steiner tree topologies in the following way. First, we define the row/column region between two Hanan grid lines for a net as the rectangular region between the two grid lines and the bounding rectangle of the net. As illustrated in Fig. 4, the shaded region in (a) is the row region between the Hanan grid lines G_{H1} and G_{H2}, and the distance between G_{H1} and G_{H2} is $v2$. Similarly, the shaded region in (b) is the column region between the Hanan grid lines G_{V1} and G_{V2}, and the distance between G_{V1} and G_{V2} is $h2$. For each column region x or row region y between two Hanan grid lines of the original net, we compute their corresponding "average congestion" AC_x or AC_y as

$$AC_x = \frac{\sum_{i=1}^{n} \frac{usg_{Vi}}{cap_{Vi}}}{n} \quad (1)$$

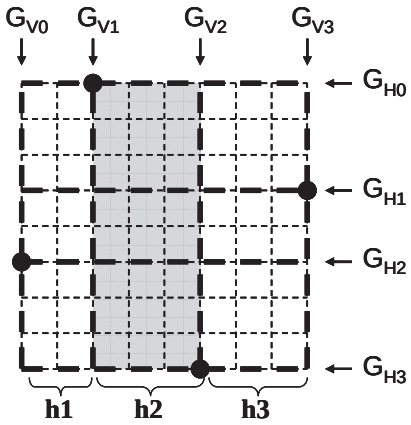
$$AC_y = \frac{\sum_{j=1}^{m} \frac{usg_{Hj}}{cap_{Hj}}}{m} \quad (2)$$

where m and n are the numbers of vertical/horizontal Hanan grid lines within the bounding box, and V_i/H_j are the vertical/horizontal grid edges at (x, i)/(j, y). Then, the distance between the corresponding two Hanan grid lines is scaled according to the "average congestion" (the higher the "average congestion", the bigger the scaling factor). In other words, we warp the Hanan grid according to the congestion map. Finally, we apply FLUTE to find the RSMT for this warped Hanan grid. In this way, we maintain a balance between wirelength and congestion when constructing the Steiner tree other than just minimizing wirelength.

In addition, we also notice that most global routers merely start to consider via usage only in the R&R stages. Since the majority of nets are in congestion free regions and not involved in R&R, their via usage will stay as the solution before R&R and is not optimized in consideration of via usage. After analysis of net topologies, we find that different tree topologies would have significant impact on the number of vias. As shown in Fig. 5, three topologies are generated for a 5-pin net. Assume that horizontal segments are routed on metal



(a) The row region between G_{H1} and G_{H2}



(b) The column region between G_{V1} and G_{V2}

Fig. 4. Hanan grid region.

layer 1 and vertical segments are routed on metal layer 2, and assume the pins are at metal layer 1. The three topologies will generate 5, 8 and 7 vias respectively. Here we define two special topologies: Horizontal Tree (H Tree) and Vertical Tree (V Tree). H tree is defined as a rectilinear tree with only one vertical trunk and all the other trunks connecting pin nodes are horizontal. Similarly, vertical tree is defined as a tree with only one horizontal trunk and all the other trunks coming out of pin nodes are vertical. If each net is assigned onto two adjacent metal layers, which our layer assignment algorithm tries to achieve by keeping segments in one net close to each other, H Tree and V Tree are two extremes with respect to the number of vias. Other trees, like the RSMT with smaller wirelength shown in Fig. 5, have via counts in between. However, it is not always the case that H Tree would have less number of vias than V Tree. If the resources on metal layer 1 is used up and the net has to go onto layer 2 and 3, it is obvious that V Tree is a better choice.

To include via usage into the picture of Steiner tree topology generation, we adjust the net topology by the usage/capacity ratio between horizontal metal layers and vertical ones, as defined in Equation (3).

$$\left(\frac{\sum cap_{(h)}}{\sum usg_{(h)}} \Big/ \frac{\sum cap_{(v)}}{\sum usg_{(v)}}\right) \tag{3}$$

In the equation, $\sum cap_{(h)}$ and $\sum usg_{(h)}$ is the sum of

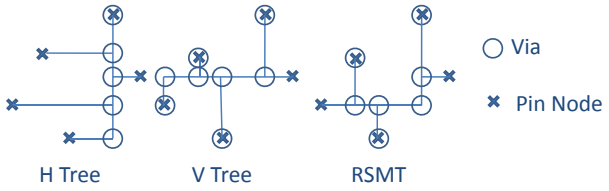


Fig. 5. Via-aware Steiner tree.

horizontal capacity and usage in the bounding box of each net. Similarly, $\sum cap_{(v)}$ and $\sum usg_{(v)}$ is the sum of vertical capacity and usage in the bounding box of each net. We use this factor in concatenation to the congestion driven factor in Equations (1) and (2) to extend or shrink the horizontal distances between the pin nodes and use FLUTE to generate adjusted topology for each tree. In this way, we can achieve 3% less via count after pattern routing stage with less than 1% overhead in wirelength and overflow.

3) Segment Shifting: The Steiner tree topology only specifies the connections between the pins and Steiner nodes in a net. After fixing the topology, there is still flexibility left for congestion optimization. For instance, in , we can focus on the segment location in the Steiner tree shown as the bold line in Fig. 6. We define a segment as a straight concatenation of routing edges that cannot be further extended. With different congestion scenarios, the segment should be shifted to different positions to avoid congested regions.

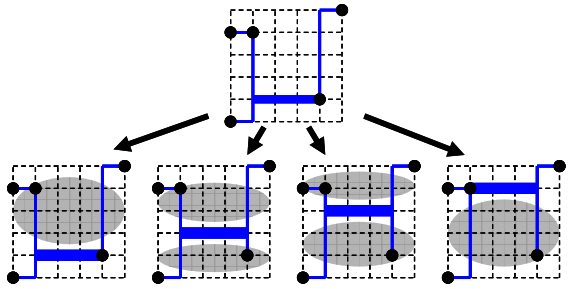


Fig. 6. Segment shifting for less congestion. (Shaded regions in bottom four cases are congested.)

Our idea is to move some segments out of the congested regions without increasing the Steiner tree wirelength. We observe that if the two endpoints of a horizontal or vertical segment are both Steiner nodes, we can shift this segment freely within a "safe range" without increasing the Steiner tree length. For a horizontal/vertical segment between a pair of Steiner nodes S_1 and S_2, the "safe range" is defined as the shifting range of y/x coordinates for S_1 and S_2 so that the Steiner tree length will not be increased when shifting the tree edge S_1-S_2. As illustrated in Fig. 7, the "safe range" of (a) a horizontal segment, or (b) a vertical segment S_1-S_2 is R_{12}. We only consider shifting segment S_1-S_2 when both S_1 and S_2 have degree 3. A Steiner node can only have degree 3 or 4, but degree 4 Steiner node has no flexibility for moving. The way to get this "safe range" is as follows. We consider the two neighbors for S_1/S_2 which are not S_2/S_1. If S_1-S_2 is horizontal, the range for safely moving S_1 and S_2 is between

the y coordinates of two neighboring nodes in the tree (R_1 and R_2 in Fig. 7(a)). Otherwise, the range for safely moving is between the x coordinates of two neighbor nodes (R_1 and R_2 in Fig. 7(b)). The "safe range" of S_1-S_2 is the common part of R_1 and R_2, which is R_{12} in Fig. 7. In R_{12}, the segment S_1-S_2 can be shifted freely without increasing the tree wirelength.

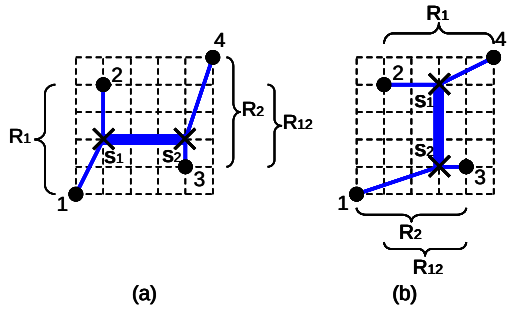


Fig. 7. "Safe range" to shift a segment.

If we allow topology change, the "safe range" can be extended in some cases. For example, in Fig. 8(a), the "safe range" for segment S_1-S_2 is R_{12}. Hence, S_1-S_2 can at most shift to the same y grid as Steiner node S_3. But we notice that S_1-S_2 can be shifted higher than S_3 without changing the Steiner tree length. The only problem here is that the topology of the tree needs to be changed. This happens when two Steiner nodes S_2 and S_3 overlap with each other (as illustrated in Fig. 8(b)). In this case, we will exchange the two Steiner nodes S_2 and S_3 to enable further shifting, which is shown in Fig. 8(c). Notice that by exchanging S_2 and S_3, we change the $topology1$ into $topology2$. In Fig. 8(c), the new "safe range" is R_{13}. Therefore, now we can explore the full range R_{123} for S_1-S_2.

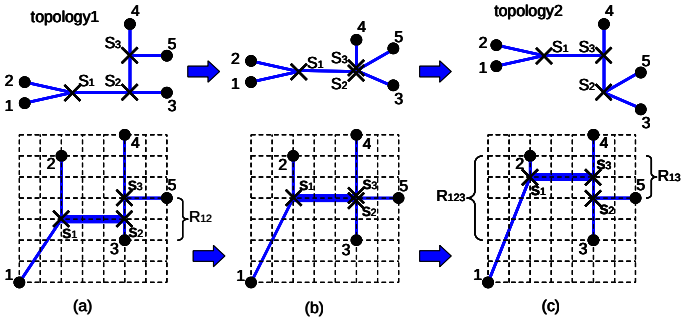


Fig. 8. Modification of tree topology during segment shifting.

After we find the "safe range" for a segment S_1-S_2, we need to decide the best position to shift it within the "safe range". The criterion for the best position is that the total congestion of all the grid edges on the Steiner tree is minimized. Hence, for every possible position, we evaluate the cost for the tree by adding up the cost on all grid edges used by the tree. Note that we only need to evaluate the demand on the grid edges affected by shifting S_1-S_2, which are all tree edges E adjacent to S_1 and S_2. Note that some tree edges $e \in E$, could be a diagonal tree edge (e.g., tree edge 1-S_1 in Fig. 7(a)). We do not know which grid edges this tree edge will use. In this case, we consider the two possible L-shape route for it and pick the one resulting in smaller congestion. For these diagonal tree edges, later stages will try to minimize the total demand of grid edges on their routing path.

We apply this segment shifting technique iteratively on all the horizontal/vertical segments in a Steiner tree until the total cost of the tree cannot be improved. After segment shifting, the Steiner tree structures for multi-pin nets are determined and the nets are broken into a set of 2-pin nets. Later the 3-bend routing and maze routing will focus on these 2-pin nets.

B. Routing Techniques

With the congestion-driven and via-aware topology, the next stage of FastRoute is actual routing. We find that there exists significant potential to improve traditional routing techniques in terms of via control and congestion reduction. The most commonly used routing techniques in global routing include L/Z/U pattern routing, monotonic routing and maze routing, as shown in Fig. 9. L/Z/U pattern routing generates limited number of via, has fast speed but cannot effectively reduce congestion. Monotonic routing and traditional maze routing, on the contrary, do better job in solving congestion problem but cannot control via count effectively. Besides, maze routing and U routing allow detour to strengthen the congestion reduction capability. Traditional maze routing is most powerful but suffers from long runtime. So all traditional routing techniques sacrifice one or several quality to improve some others.

To address this problem, we propose 3-bend routing, a fast routing technique with enhanced congestion reduction capability than traditional pattern routing and much less via than maze routing. Even with 3-bend routing, FastRoute has to use maze routing as the last resort for highly congested area. In order to enhance the usefulness of maze routing, we propose a new multi-source and multi-sink maze routing technique which greatly improves the flexibility and performance of traditional maze routing.

Fig. 9. L/Z/U, monotonic and maze routing.

1) 3-Bend routing: A 3-bend route is a 2-pin rectilinear connection that has at most three bends and possible detour. It is much more flexible than L/Z/U route on solving congestion problem. Comparing to monotonic route [17] and maze route, 3-bend route has advantage on having less vias. Fig. 10 shows two possible 3-bend routes for a tree edge, $S \rightarrow B \rightarrow T$ and $S \rightarrow B' \rightarrow T$. No L/Z/U routing can avoid the congested area marked as shades. However, the 3-bend route $S \rightarrow B \rightarrow T$ can achieve congestion free routing with least bends possible.

To find the best 3-bend routing path for a 2-pin net, we assume one pin to be the source ($S = (x_s, y_s)$) and the other one to be the sink ($T = (x_t, y_t)$). Without loss of generality, we assume S is at the lower-left corner and T is at the upper-right corner. We define the possible detouring region as an

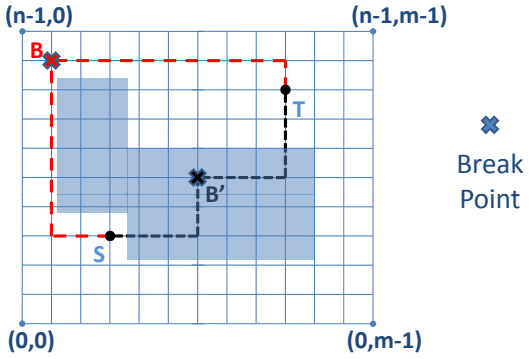


Fig. 10. 3-bend routing.

expanding box for each net. It is calculated depending on the size, location and congestion of each net. The larger net with more congestion will have a larger expanding box. The pseudo code to compute the best 3-bend path for an S-T bounding box of size $p \times q$ and an expanding box of $m \times n$ nodes is given in Fig. 11.

Algorithm 1: 3-Bend Routing

1. $C_{best} = +\infty$
2. **for** $y = 0$ to $n-1$
3. $\quad d_h(0, y) = 0$
4. $\quad$ **for** $x = 1$ to $m-1$
5. $\quad\quad d_h(x, y) = d_h(x-1, y) + cost_h(x-1, y)$
6. **for** $x = 0$ to $m-1$
7. $\quad d_h(x, 0) = 0$
8. $\quad$ **for** $y = 1$ to $n-1$
9. $\quad\quad d_v(x, y) = d_v(x, y-1) + cost_v(x, y-1)$
10. **for** $y = 0$ to $n-1$
11. $\quad$ **for** $x = 0$ to $m-1$
12. $\quad\quad B = (x, y)$
13. $\quad\quad d_{L1}(B) = |d_h(S) - d_h(x, y_s)| + |d_v(x, y_s) - d_v(B)|$
14. $\quad\quad d_{L2}(B) = |d_h(S) - d_h(x_s, y)| + |d_v(x_s, y) - d_v(B)|$
15. $\quad\quad d_{L3}(B) = |d_h(T) - d_h(x, y_t)| + |d_v(x, y_t) - d_v(B)|$
16. $\quad\quad d_{L4}(B) = |d_h(T) - d_h(x_t, y)| + |d_v(x_t, y) - d_v(B)|$
17. $\quad\quad$ Compute the cost of four possible 3-bend paths (i.e., L1-L3, L1-L4, L2-L3, L2-L4) from the four L-paths above plus via cost and compare them to C_{best}. If better, update the best 3-bend path.

Fig. 11. 3-bend routing algorithm.

In the algorithm, $d_h(x, y)$ and $d_v(x, y)$ denote the costs for a path going from the point (x, y) horizontally to the left boundary and vertically to the bottom boundary respectively. $cost_h(x-1, y)$ is the cost for using the horizontal grid edge between $(x-1, y)$ and (x, y) while $cost_v(x, y-1)$ is the cost for using the vertical grid edge between $(x, y-1)$ and (x, y). To balance wirelength and congestion, we use the same cost function as in maze routing, which will be discussed in Sec. 3.3.2. Line 2 to Line 9 create two tables that have the cost for a bend-free edge between any points in the expanding box and the left or bottom boundary, from which the cost of a 3-bend path between any two nodes in the expanding box could be easily calculated. A 3-bend path could be concatenated from two L-shaped paths, like using $S \rightarrow B$ and $B \rightarrow T$ to form $S \rightarrow B \rightarrow T$. So we add a break point in the expanding box, calculate the cost of the induced L-shaped paths in Lines 13 to 16, from which we can compute the cost of all the possible 3-bend paths and find the best solution. Lines 2 to 9 take $O(mn)$ time. Lines 10 to 19 also take $O(mn)$ time. So the complexity of 3-bend routing algorithm for a 2-pin net is $O(mn)$, the same as Z routing. It is worth noticing that the algorithm shown in Fig. 11 may compute some paths with overlapping segments. But they will be automatically excluded because of their higher cost.

The small via count, short runtime and good congestion solving capability let 3-bend routing to become an alternative to maze routing. In the past, only a small percentage of nets would be routed by maze routing but the statement fails to hold as the benchmarks become more complex. We apply 3-bend routing for congested nets before maze routing, which leads to runtime and via count reduction.

2) Multi-source Multi-sink maze routing: Maze routing is used as the last resort to solve congestion in global routing. Originally, maze routing algorithm is designed to find the shortest path connecting two pins in the presence of routing blockages. Later, it has been extended to find a path connecting two pins in such a way that it favors a path that passes through less congested area according to some cost function. It is a very powerful technique to find paths avoiding congestion.

However, traditional maze routing only finds the shortest path between two pins. For multi-pin nets, a typical way is to break the routing tree into 2-pin nets, and route each 2-pin nets by maze routing. We find that this kind of independent edge-by-edge routing scheme for each net fails to generate good routing solutions for the multi-pin nets. Fig. 12 illustrates three different scenarios. The shaded areas denote the congested regions.

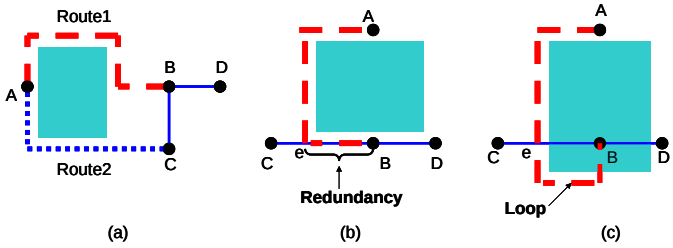


Fig. 12. Maze routing scenarios.

- *Unnecessary detour:* Consider the scenario in Fig. 12 (a). The dashed route "Route1" is the maze routing result for tree edge (A, B). However, if the path does not need to go from A to B, "Route2" is a better choice in terms of cost.
- *Redundant routing:* Consider the scenario in Fig. 12 (b). The dashed route is the maze routing result for tree edge (A, B). However, the (e, B) part on the path is already part of the routing tree, and it is redundant to repeat it.
- *Unintentional loop:* Consider the scenario in Fig. 12 (c). The dashed route is the maze routing result for tree edge (A, B). A loop is created in the routing tree. It is obvious

that this loop is not needed and only the part from A to e is necessary on the path.

As we can see in these three scenarios, unnecessary wires are used to route the multi-pin nets. This results in using more routing resources than necessary and causes extra routing congestion. The major defect of this edge-by-edge routing scheme for each net is that the topology information is neglected. When routing a tree edge for multi-pin nets, global router just needs to rejoin the two disconnected subtree generated by rip-up procedure, no matter where the rejoining path ends.

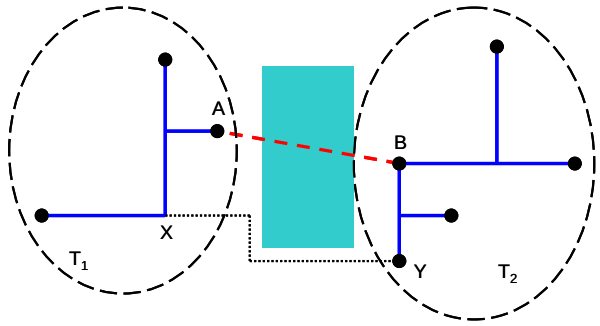


Fig. 13. Multi-source multi-sink maze routing.

Aware of the problem, we propose a multi-source multi-sink maze routing algorithm. The main idea is that the existing routing tree is respected when we route a tree edge for a multi-pin net. We do not constrain the two endpoints of the routing path to be the original endpoints of the tree edge being routed. As illustrated in Fig. 13, suppose we are routing a tree edge (A, B) in the routing tree T for a multi-pin net N. We first remove (A, B) from T and obtain two subtrees T_1 and T_2. (Note that T_1 and T_2 can be just a point.) We treat all the grid points on T_1 as sources, and all the grid points on T_2 as sinks. Then, we apply the multi-source multi-sink maze routing to find the best path connecting T_1 and T_2 to form a tree. In Fig. 13, the dotted line from X to Y is the best path to connect T_1 and T_2.

Our multi-source multi-sink maze routing algorithm is shown in Fig. 14. In the algorithm, $d(g)$ is the distance from T_1 to g, defined as the total cost of all grid edges passed by the temporary shortest path from T_1 to g. The algorithm follows the framework of Dijkstra's algorithm [27]. Lines 1-5 initialize the distance d, priority queue Q and destination points. Lines 6-17 are the loop similar to Dijkstra's algorithm. Line 18 just traces back to find the shortest path from T_1 to T_2.

Our algorithm finds the least cost routing path from T_1 to T_2. Theorem 1 gives the optimality of the algorithm.

Theorem 1 The path found by multi-source multi-sink maze routing algorithm is the least cost routing path from T_1 to T_2.

Proof: First of all, note that the cost function cost(u, v) is a positive function in our problem. In Line 3, $d(u) = 0$ for all the grid points on T_1. Hence, we can assume a super source which replaces all the grid points on T_1, and all grid points adjacent to T_1 are its neighbor. Similarly, we can assume a super sink which replaces all the grid points on T_2, and all adjacent grid points to T_2. Then the problem is transformed

Algorithm 2: Multi-source Multi-sink Maze Routing

1. $d(g) = \inf$ for all grid points g
2. Find subtree T_1 (contains A) and T_2 (contains B) after removing tree-edge (A, B)
3. Set $d(u) = 0$ and $\pi(u) = nil$, for all grid points u on T_1
4. Set up a priority queue Q with all grid points on T_1
5. Mark all grid points on T_2 as sink point
6. $u \leftarrow$ Extract-Min(Q)
7. **while** u is not sink point
8. **for** each neighbor grid point v of u
9. **if** $d(v) > d(u) + cost(u, v)$
10. $\pi(v) = u$
11. **if** v is in Q
12. Update Q
13. **else**
14. Insert v into Q
15. $u \leftarrow$ Extract-Min(Q)
16. Trace back from u using π to find the shortest path from T_1 to T_2

Fig. 14. Multi-source multi-sink maze routing algorithm.

to a single-source, single-sink shortest path problem. The optimality follows the optimality of Dijkstra's algorithm.

The only thing left is to prove the stopping criterion is correct. Recall that we stop when a destination point on T_2 is extracted from Q. Assume u is the first destination point extracted from Q. For the purpose of contradiction, let w be the destination point which is on the shortest path from T_1 to T_2. Hence, we have $d(w) < d(u)$. However, when we extract u from Q, w is still in Q, which means $d(w) \geq d(u)$. Because the cost function is positive, $d(w)$ will never decrease in later updating. Therefore, we obtain a contradiction that $d(w) \geq d(u)$. ■

Now we analyze the complexity of the algorithm. Assume there are V grid points in the search region. Lines 1-5 takes time $O(V)$. Each Extract-Min operation on the priority queue Q takes time $O(lgV)$. There are at most V iterations for the while loop. For each u, there are at most 4 neighbors adjacent to it. The insertion and updating of Q takes time $O(lgV)$. The total complexity is therefore $O(VlgV)$.

We apply this multi-source multi-sink maze routing algorithm on the tree edges of multi-pin nets. The runtime of maze routing algorithm is highly related to the size of the search region. In order to speed up the algorithm, we do not search the whole grid graph to find the least cost path. Instead, we use expanding box in the same way as 3-bend routing to significantly reduce the runtime while maintaining good solution quality. In our implementation, the enlarge value is proportional to the size and level of congestion of the original bounding box.

We want to point out one issue for the multi-source multi-sink maze routing technique. It can totally change the routing tree structure because the endpoints of new routing path do not need to be the endpoints of the tree edge being routed. For example, in Fig. 15, the Steiner tree structure is changed

from (a) to (b) because of the new routing of tree edge (A, B). Hence, we need to update the Steiner tree structure accordingly after routing each tree edge by multi-source multi-sink maze routing.

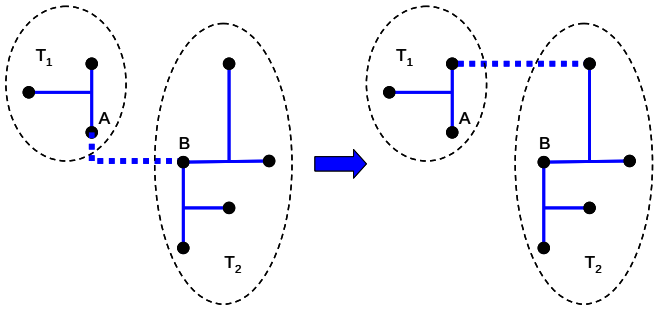


Fig. 15. Steiner tree topology changed by maze routing.

C. Convergence Enhancement Techniques

In addition to new topology and routing techniques, FastRoute integrates several performance enhancement techniques to further improve routing quality and reduce run time. In the 2007 and 2008 ISPD global routing contests, we find that traditional global routing framework may easily get trapped in local minimal of solution space and require significant runtime and control to jump out. In order to solve such problem, we propose two new enhancement techniques to improve the convergence of global routing.

1) Virtual Capacity Technique: Other recently published academic global routers, including BoxRouter [7], Archer [8], NTHU-R [9][10], NTUgr [11] and FGR [12] employ negotiation based maze routing technique, which increments the maze routing cost for consistently congested grid edges. However, such negotiation based cost adjustment lacks theoretical basis and requires significant tuning before it can work properly.

Instead of negotiation based maze routing technique, we propose virtual capacity, a systematic alternative to handle congestion problem. Virtual capacity tries to use adjusted "virtual capacity" instead of original capacity to guide maze routing. Given a global routing solution, consider any congested grid edge e. With capacity u_e and capacity c_e, overflow would be $o_e = u_e - c_e$. We denote the virtual capacity as vc_e. The basic idea of virtual capacity is to reduce the capacity of e by o_e units (i.e., set the virtual capacity to $c_e - o_e$) for the next round of maze routing. Because of the reduction in capacity, grid edge e becomes more expensive to use and hence some of its routing demand will hopefully be pushed away. In the ideal situation, exactly o_e units of routing demand will be pushed away in order to bring the congestion back to the level of the previous round, if we measure the overflow using virtual capacity. Thus, the new routing demand will be $u_e - o_e = u_e - (u_e - c_e) = c_e$, i.e., the same as the original capacity. In order words, grid edge e will not be congested in the second round of global routing. In reality, more or less than o_e units of routing demand might be pushed away because other grid edges cannot absorb or will absorb more than the pushed routing demand. So it is necessary to update the virtual capacities and apply maze routing again to further reduce the overflow.

In Sec. IV-C1a, we discuss the initialization of virtual capacities. In Sec. IV-C1b, we describe the updating of virtual capacities during the routing process.

*a) Virtual Capacity Initialization by Alternative Congestion Estimation (*ACE*):* Virtual capacity is initialized by subtracting the overflow of last round of routing from the actual grid edge capacity. But for the first round of routing, we want to predict the overflow in order to use virtual capacity to speed up the convergence. We use adaptive congestion estimation (*ACE*) technique to predict the overflow. *ACE* assigns net usage to proper grid edge in a more realistic manner and can estimate overflow much more accurately than traditional probabilistic estimation. We implement the estimation using the following two assumptions: (1) Routing region of each 2-pin net is confined within the bounding box. (2) Fractional usage assignment is allowed. The first assumption suggests that we only consider the grid edges inside the bounding box. The second assumption allows breaking the integer usage into fractional values. The fractional value models the behavior of global router that evenly distributes the routing usage in congested region.

The notation of problem formulation of *ACE* is shown in Table I. *ACE* designs a more realistic usage assignment method to estimate routing demand. In general, it allocates the new routing demand to regions where routing demands are previously low.

TABLE I
ACE USAGE ASSIGNMENT NOTATION

N	number of 2-pin nets
$BBox_k$	bounding box of net_k
r_k	number of rows inside $BBox_k$
c_k	number of columns inside $BBox_k$
$left_k$	left coordinate of $BBox_k$
$right_k$	right coordinate of $BBox_k$
top_k	top coordinate of $BBox_k$
$bottom_k$	bottom coordinate of $BBox_k$
$c_{i,j}^{V/H}$	capacity of the $e_{i,j}^{V/H}$
$p_{i,j}^{V/H}$	current assigned usage of $e_{i,j}^{V/H}$

Consider the usage assignment of one single 2-pin net, the usage ready to be assigned within the bounding box is 1. Without loss of generality, here we just discuss the assignment for vertical grid edges. The usage assignment algorithm for vertical grid edges is shown in Fig. 17. Each row is processed independently. Inside one row, grid edges are sorted in a decreasing order according to the value of $cost_{i,j}^V$, which is equal to $p_{i,j}^V + m_i^V - c_{i,j}^V$. m_i^V is the value of maximum grid edge capacity of row i. The algorithm compares the average potential assigned usage with largest current assigned usage. It iteratively excludes the grid edge with largest current assigned usage until an even assignment is possible. The time complexity required for processing single 2-pin net net_k is $O(r_k c_k \cdot log(c_k))$. Fig. 16 illustrates the assignment process.

Due to the sequential manner of usage assignment, the net processing order may significantly affect accuracy. *ACE* processes smaller span nets with higher priority. The net

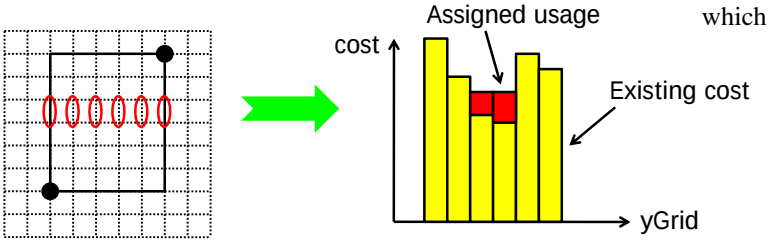


Fig. 16. 2-pin net usage assignment(vertical case).

Algorithm 3: ACE 2-Pin Net Assignment Vertical(net_k)

1. **for** $(i = top_k \cdots bottom_k + 1)$
2. $m_i^V = max(c_{i,j}^V), j \in [left_k, right_k]$
3. $\Delta = right_k - left_k + 1$
4. **for** $(j = left_k \cdots right_k)$
5. $cost_{i,j}^V = p_{i,j}^V + m_i^V - c_{i,j}^V$
6. $C_{sum} = \sum cost_{i,j}^V \ (j \in [left_k, right_k])$
7. Sort $cost_{i,j}^V (j \in [left_k, right_k])$ by decreasing order
8. Copy sorted grid edge index into queue Q
9. **for** $(t = 1 \cdots \Delta)$
10. **if** $\frac{1+C_{sum}}{\Delta - t + 1} > cost_{i,Q(t)}^V$
11. **for** $(n = t \cdots \Delta)$
12. $p_{i,Q(n)}^V = \frac{1+C_{sum}}{\Delta - t + 1} - m_i^V + c_{i,Q(n)}^V$
13. break out of the second for loop
14. **else**
15. $C_{sum} = C_{sum} - cost_{i,Q(t)}^V$

Fig. 17. The ACE 2-pin net assignment algorithm for vertical grid edges

span represents width of bounding box in vertical grid edge assignment or height of bounding box in horizontal grid edge assignment. Nets with larger spans offer more choices to distribute the net usage. Therefore we order the nets by net span and perform usage assignment for nets with smaller span first. Fig. 18 shows the detail of the whole *ACE* algorithm.

Algorithm 4: ACE Usage Assignment

1. $p_{i,j}^V = 0 \ \ \forall i, j$
2. $p_{i,j}^H = 0 \ \ \forall i, j$
3. Sort 2-pin nets by BBox width with increasing order
4. Copy sorted nets into queue Q^V
5. **for** $(t = 1 \cdots m)$
6. ACE 2-pin net assignment vertical($Q^V(t)$)
7. Sort 2-pin nets by BBox height with increasing order
8. Copy sorted nets into queue Q^H
9. **for** $(t = 1 \cdots m)$
10. ACE 2-pin net assignment horizontal($Q^H(t)$)

Fig. 18. The ACE usage assignment algorithm

Now we apply *ACE* to solve the routing example in Fig. 19. Consider vertical grid edges along the line T . Due to the permutation, the net processing order becomes A→B→C→D. After assigning net A, current assigned usage becomes(1,0,0,0). And we will get (1,1,0,0) after assigning net B. As it goes on, the final assigned usage will be (1,1,1,1). So the estimation will not generate any potential congestion, which matches exactly with the optimal routing solution.

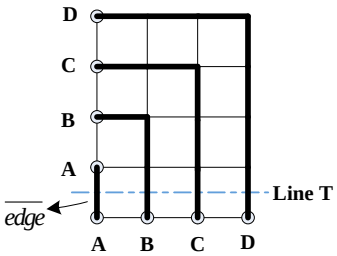


Fig. 19. 2-pin net assignment example for *ACE*.

After the estimation, virtual capacity will be initialized by Equation (4).

$$vc_e = c_e - max(0, p_e - c_e) \quad \forall e \tag{4}$$

In Equation (4), c_e denotes actual grid edge capacity and p_e is the estimated usage obtained by *ACE*. The new capacity after subtraction is named virtual capacity, which is vc_e in abbreviation.

The time complexity of *ACE* technique is $O(m \cdot log(m) + mn^2 \cdot log(n))$ and following is the detailed analysis. To sort out the order of m 2-pin nets, it will take $O(m \cdot log(m))$. For each net, the worst case time complexity is $O(n^2 log(n))$, where n is the maximum number of horizontal and vertical grids. Hence, in general, the overall worst case time complexity is $O(m \cdot log(m) + mn^2 \cdot log(n))$. But the bounding box of a 2-pin net is generally small. Therefore, on average, *ACE* accounts for around 2% of total global routing runtime.

b) Virtual Capacity Updating: After the virtual capacity initialization, we use virtual capacity instead of the actual grid edge capacity to guide 3-bend routing and maze routing. As the rip-up and reroute proceeds, FastRoute updates virtual capacity at the end of each routing iteration.

The update method is presented in Equation (5) and (6). Existing overflow o_e is calculated as the difference between grid edge usage u_e and actual grid edge capacity c_e. Virtual capacity will be monotonically decreased for the grid edges that are consistently congested.

$$o_e = u_e - c_e \quad \forall e \tag{5}$$

$$vc_e = vc_e - o_e \tag{6}$$

It is worth noticing that the overflow calculated in Equation (5) can be negative. When we use negative overflow to adjust virtual capacity, it will go up. Thus, virtual capacity adjustment will automatically reduce the cost for grid edges that were previously congested but currently not. In this way, virtual capacity technique can better utilize grid edges and further enhance the convergence of global router.

2) Adaptive Maze Cost Function: With virtual capacity, we have a systematic way to model congestion. But we still need a cost function to put it into use when maze router evaluates alternative routes.

We propose a logistic function [28] based adaptive cost function, as shown in Equation (7). In the function, k is the

coefficient controlling the function curve slope when u_e is below c_e. k is adaptively adjusted in different maze routing phases. In the initial phase, k is set small to preserve good wirelength. Normally in the first few iterations, many nets need rip-up and reroute. If a large k coefficient is applied, those nets would be rerouted with huge detour. While in the final stage of maze routing, the cost function curve is made steep to aggressively drive down the residual overflow. There are two other coefficients in the function: S determines the slope when u_e is over c_e. H is the cost height which controls the trade off between converging speed and wirelength and would be increased each maze routing iteration.

$$cost_e = \begin{cases} 1 + H/(1 + exp(-k(u_e - vc_e))) & if \quad 0 < u_e \leq c_e \\ 1 + H + S \times (u_e - vc_e) & if \quad u_e > c_e \end{cases} \tag{7}$$

D. Spiral Layer Assignment

There are generally two ways to generate solutions for 3D global routing benchmarks. One is running routing techniques and layer assignment concurrently. It overly complicates the problem and is rarely used. The other more popular way first projects the 3D benchmarks from aerial view, finds a solution for the 2D problem and expands the solution to multiple layers. This expansion is called layer assignment, which has significant impact on the number of vias for the final solution. To keep FastRoute fast, we propose a sequential layer assignment algorithm that would assign the 2D solution into routing layers, from lower layers to higher ones. The layer assignment algorithm will not change the aerial view of 2D solution and thus keep the total wirelength. Besides, our algorithm keeps total number of overflow unchanged. Thus, if we can find a congestion-free solution for the 2D global routing problem, we can find a valid solution for the original 3D problem.

In the algorithm, we first sort the nets considering their total wirelength and number of pin nodes. Then we order the segments in each net according to their locations in the net. Finally, we assign layers using dynamic programming, segment by segment, net by net.

Due to the competition of different nets in the assigning sequence and greedy nature of layer assignment, careless early assignment causes later nets switching among the layers and thus generates a large number of unnecessary vias. Smaller nets connecting nearby global cells are considered relatively local and should use lower metal layers. On the contrary, longer nets assigned to upper layers will encounter less switching between layers and will use wider tracks on top layers to achieve better timing. Furthermore, we observe that nets with higher number of pins tend to cause more vias. So we order nets by an increasing order of $\sum wl/\#Pins$, where $\sum wl$ is the total wirelength for a net. Thus, we keep nets with smaller total wirelength and higher pin count on the lower layers.

For each net, we order segments for the following reason. The only layer information for a net is that the pin nodes must go up to at least metal layer 1 to have metal connections. So we order the segments in each net in increasing order of their distance to the pin nodes. Here, the distance is defined as the number of segments the two nodes in a segment have to traverse to reach the nearest pin node. We first assign layers to the segments with 0 distance i.e., segments that have at least one pin node and move onto segments with larger distance. By such an order, we are sure that at least one end of each segment has the information that which layers the pin node ranges between. Thus, we start assigning segments on the periphery of a net and continue inwardly.

As shown in Fig. 20, we create a "via grid graph" to assign each segment to metal layers. We call each node on the graph a "via node". Vertical grid edges represent the possible places to add via while the horizontal grid edges are constructed from the actual 2D path in the "via grid graph". We pull straight the original zigzagged 2-pin net to form the horizontal grid edges in the via grid graph and copy the capacity and usage of corresponding grid edge from the original grid graph. We break the segments in a tree into the size of grid edges and assign them to layers one by one. Such breakdown enables us to keep the total number of wirelength and overflow of the 2D solution unchanged. Without loss of generality, we assume sources S_i on the very left column and targets T_j on the right. If we do not know the layer information about the ending node, layer 1 to L are all considered to be targets. Here, L is the number of metal layers in a benchmark. Otherwise, the target is set to be the spanning range of the ending node.

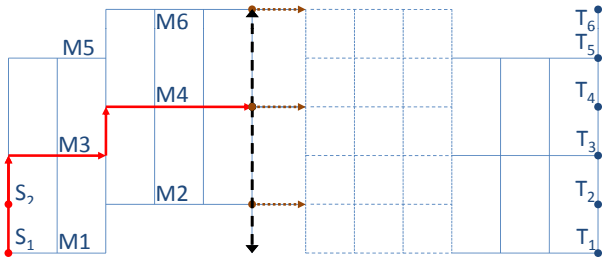


Fig. 20. Dynamic programming layer assignment.

We associate every via node with a cost, which represents the least number of vias on the paths from the node to any source nodes. Since we do not change the aerial view of a net, a 3D path must and must only use the horizontal segments between two adjacent columns once. Thus, the cost for a node is the same as its left neighbor if there is still routing resource or one plus the cost associated with the upper or lower neighbor nodes, whichever is smaller. The pseudo-code to process each segment with wirelength n is shown in Fig. 21.

In the algorithm, Line 1 uses $O(nL)$ time and Line 2 takes $O(L)$ time. The update of costs from vertical neighbors involves with a series of sorting, comparison and update, which takes $O(LlgL)$ time. However, because of the small number of L (typically less than 10 depending on the semiconductor process), we use an $O(L^2)$ implementation. Hence, Line 4 to Line 8 take $O(nL^2)$. So the complexity of layer assignment for each segment is $O(nL^2)$.

Algorithm 5: Layer Assignment for Segment

1. Initialize the cost for all the via nodes to $+\infty$
2. For every source s_j, $C(j,0) = 0$
3. Update the cost for other via nodes on the first column
4. **for** $x = 1$ to $n-1$
5. **for** $j = 1$ to L
6. **if** $cap(j, x-1) > usg(j, x-1)$
7. $C(j,x) = C(j, x-1)$
8. Update the cost from vertical neighbors.
9. Find the least cost for any sink node and trace back using $C(j,x)$

Fig. 21. Layer assignment algorithm for segment.

V. Experimental Results

We implemented FastRoute in C with Steiner tree package FLUTE and the current version is FastRoute 4.1. All the experiments are performed on a Linux machine with 2.8 GHz Intel processor and 32GB RAM. We run experiments on ISPD08 global routing contest benchmarks [30]. The benchmark statistics are shown in table II. It is worth mentioning that FastRoute 4.1 now adopts a single set of tuning and avoids specific benchmark tuning to demonstrate the effectiveness of global routing framework and techniques presented in this work. On the contrary, all the participants in ISPD 08 contest use benchmark specific tuning.

The 2008 set of benchmarks has 8 new benchmarks and 8 benchmarks inherited from 2007. However, when ISPD08 global routing contest considers one unit of via at the same cost of one unit of wirelength, the one held in 2007 charges via at a cost three times of the cost for wirelength. In our experiment, we use the rules set by the 2008 contests which treats wire segments and vias equally.

In Table III., we compare the performance of FastRoute 4.1 on the ISPD08 global routing contest benchmarks with the top 4 routers besides FastRoute 3.0. Again, FastRoute 4.1 is the fastest router. For the four benchmarks that no one can successfully finish routing without incurring any overflow, FastRoute achieves lowest overflow for two benchmarks. Due to the fact that other groups do not disclose the details about the metal wirelength part and via part of the total wirelength, we only compare the total wirelength. Since no newer data is available for BoxRouter2.0 after the ISPD08 contest, we quote the results for BoxRouter2.0 from ISPD08 global routing contest results. All runtime are scaled to 2.8GHz.

Comparing to NTHU-R2.0, the 2008 ISPD global routing contest winner, FastRoute achieves 0.01% and 74% improvement for total wirelength and runtime respectively on the 12 routable benchmarks. Comparing to the 2^{nd} place winner, NTUgr, FastRoute 4.1 can finish routing one more benchmark without overflow and can achieve 3.8% less wirelength in 15X faster speed for 11 benchmarks that the two routers both successfully finished.

[1]Segment wirelength, via and total wirelength are in unit of $10K$.

[2]Wirelength and runtime comparisons are based on overflow-free benchmarks.

Via accounts for 26% to 47% of the total wirelength of FastRoute solutions to the contest benchmarks. Although via has higher resistivity and larger process variation which makes it much more important than before, we still believe that congestion reduction is the most important function for global router. Both of the two recent global routing contests held by ISPD gave highest priority to the overflow of solutions for evaluating the performance of global routers.

Even though most global router that participated in the 2008 ISPD global routing contests have greatly improved over their earlier version in the 2007 contest, we observe that some routers still face two challenges. One is how to handle the congestion left in the final stages. Even though FastRoute 4.1 and NTHU-R2.0 successfully finished routing for newblue1, they both failed newblue4, newblue7 and bigblue4, with a residue overflow of just less than 150. The huge runtime spent by NTUgr and BoxRouter2.0 on newblue1 showed the inability to solve the few final overflow. Another challenge is the effectiveness for the global routers to balance between reducing the number of overflow and extending wirelength. The conflict incurs due to the fact that one of the most efficient method to reduce congestion is detour, i.e. extending wirelength, which could however induce congestion in other areas. One important way to effectively control the trade-off is through cost function used in maze routing. Although cost functions evolve from step function to logistic function and the variants of logistic functions, the fact that global routers that generates shorter wirelength or longer wirelength can only reduce congestion to a similar level demonstrates that there is considerable potential for the academic global routers to improve in this area.

To demonstrate the effectiveness of the global routing techniques proposed in this paper, we turn off certain techniques to see the performance degradation as shown in Table IV. In the column "No Tree Adj", we turn off the congestion-driven via-aware Steiner tree generation and use unadjusted tree topology directly generated from FLUTE. This configuration of FastRoute leads to 38% more congestion and 23% run time overhead. The "No 3-Bend" column shows the performance of FastRoute without 3-Bend routing. We observe degradation for all three qualities we focus on, though the degradation are not very significant. However, FastRoute spends 55% more runtime for the four unroutable benchmarks without 3-Bend routing, which has explanation in the fact that 3-bend routing is much more efficient than maze routing. The "No VCA" column shows results generated by FastRoute without Virtual Capacity Adjustment. Without convergence assisting techniques, FastRoute only finishes 5 benchmarks without overflow. This configuration also dramatically increase total wirelength and runtime because FastRoute spends much more time running maze routing to try to eliminate overflow. For the last configuration, we turn off net ordering and segment ordering used in the spiral layer assignment and it shows that the two ordering saves 11% of wirelength, which would translate into significantly more percentages of via.

[3]Wirelength and runtime comparisons are based on overflow-free benchmarks.

TABLE II
EXPERIMENTAL BENCHMARKS STATISTICS

Name	Grids	#Layers	#Nets	#Routed Nets	Max Deg	Avg Deg
adaptec1	324×324	6	219K	177k	340	4.2
adaptec2	424×424	6	260K	208k	153	3.9
adaptec3	774×779	6	466K	368k	82	4.0
adaptec4	774×779	6	515K	401k	171	3.7
adaptec5	465×468	6	867K	548k	121	4.1
newblue1	399×399	6	332K	271k	74	3.5
newblue2	557×463	6	463K	374k	116	3.6
newblue3	973×1256	6	552K	442k	141	3.2
newblue4	455×458	6	636K	531k	152	3.6
newblue5	637×640	6	1.26M	892k	258	4.1
newblue6	463×464	6	1.29M	835k	123	3.8
newblue7	488×490	8	2.64M	1.65M	113	3.6
bigblue1	227×227	6	283K	197k	74	4.1
bigblue2	468×471	6	577K	429k	260	3.5
bigblue3	555×557	8	1.12M	666k	91	3.4
bigblue4	403×405	8	2.23M	1.13M	129	3.7

TABLE III
FASTROUTE 4.1 RESULTS ON 3D VERSION OF ISPD08 GLOBAL ROUTING CONTEST BENCHMARKS

	FastRoute 4.1					NTHU-R2.0 [10]			NTUgr [11]			BoxRouter2.0 [30]		
name	ovfl	swl[1]	via[1]	twl[1]	cpu(s)	ovfl	twl[1]	cpu(s)	ovfl	twl[1]	cpu(s)	ovfl	twl[1]	cpu(s)
adaptec1	0	36.4	17.4	53.8	193	0	53.4	568	0	57.4	270	0	52.9	1227
adaptec2	0	33.3	18.9	52.2	51	0	52.3	98	0	53.7	66	0	52.7	162
adaptec3	0	96.7	34.5	131.2	183	0	131.0	510	0	135.0	264	0	131.8	1635
adaptec4	0	89.9	31.4	121.3	61	0	121.7	121	0	123.7	72	0	122.1	403
adaptec5	0	104.1	51.7	155.8	407	0	155.4	1077	0	159.9	918	0	156.9	1889
newblue1	0	24.5	21.8	46.3	361	0	46.5	290	6	49.3	58650	44	47.5	74488
newblue2	0	46.7	28.5	75.2	40	0	75.7	57	0	76.9	36	0	75.9	109
newblue3	31532	76.5	31.3	107.8	1353	31454	106.5	5728	31024	188.3	53040	38958	109.1	82615
newblue4	142	82.9	47.6	130.5	2140	138	130.5	4525	142	143.8	67086	200	129.5	78225
newblue5	0	148.8	82.1	230.9	565	0	231.6	908	0	244.9	1230	0	232.9	1700
newblue6	0	103.7	73.8	177.5	598	0	176.9	847	0	186.6	1278	0	179.8	1785
newblue7	54	186.1	166.8	352.9	16888	62	353.5	6734	310	372.2	86730	208	358.6	84743
bigblue1	0	37.9	18.7	56.6	257	0	56.0	641	0	60.0	918	0	56.9	1147
bigblue2	0	49.3	41.6	90.9	457	0	90.6	397	0	91.2	14898	0	90.4	2346
bigblue3	0	78.9	51.1	130.0	114	0	130.7	235	0	133.5	240	0	131.3	380
bigblue4	138	121.3	108.9	230.2	2144	162	231.0	6159	188	242.8	24786	472	231.6	52644
Comparison[2]	1	\	\	1	1	0.998	1.001	1.75	0.994	1.038	23.99	1.25	1.0007	26.55

TABLE IV
CONTRIBUTIONS FROM TECHNIQUES IN FASTROUTE

	FastRoute 4.1			No Tree Adj			No 3-Bend			No VCA			Input Order LA		
name	ovfl	twl	cpu(s)	ovfl	twl	cpu(s)	ovfl	twl	cpu(s)	ovfl	twl	cpu(s)	ovfl	twl	cpu(s)
adaptec1	0	53.8	193	0	54.1	211	0	54.2	192	0	54.4	941	0	58.6	176
adaptec2	0	52.2	51	0	52.4	56	0	52.4	58	126	52.8	343	0	56.7	37
adaptec3	0	131.2	183	0	131.8	195	0	132.2	189	0	130.5	384	0	140.5	155
adaptec4	0	121.3	61	0	121.5	60	0	121.3	62	0	121.3	70	0	129.4	30
adaptec5	0	155.8	407	0	156.5	448	0	157.1	457	0	171.6	315	0	171.6	315
newblue1	0	46.3	361	0	46.4	335	0	46.4	313	1362	46.6	2431	0	52.2	299
newblue2	0	75.2	40	0	75.4	46	0	75.1	40	0	75.2	41	0	83.0	15
newblue3	31532	107.8	1353	33628	107.8	3895	38563	107.5	4345	34528	108.5	4679	31532	114.3	3737
newblue4	142	130.5	2140	146	130.9	2211	144	130.9	2644	1352	130.5	12940	142	139.9	2189
newblue5	0	230.9	565	0	231.9	646	0	232.2	670	166	235.6	20244	0	254.8	446
newblue6	0	177.5	598	0	179.0	703	0	179.2	668	42	180.2	4755	0	202.1	514
newblue7	54	352.9	16888	80	354.8	17376	86	353.3	22284	1124	357.1	42598	54	403.3	12633
bigblue1	0	56.6	257	0	57.2	300	0	57.0	383	84	57.2	2179	0	63.6	231
bigblue2	0	90.8	457	0	91.1	516	0	91.1	762	48	91.6	2355	0	99.1	434
bigblue3	0	130.1	114	0	130.2	115	0	130.4	242	268	131.3	3195	0	150.0	73
bigblue4	138	230.2	2144	144	232.1	4059	142	231.5	5529	648	228.9	7177	138	265.5	2807
Comparison[3]	1	1	1	1.38	1.005	1.23	1.07	1.004	1.10	1.25	1.02	11.33	1	1.11	0.83

The source code of lastest FastRoute 4.1 could be requested for download at http://home.eng.iastate.edu/cnchu/FastRoute.html. If the reader is interested, one can find all the algorithm and tuning factors inside. The latest FastRoute 4.1 uses a single set of tuning factors. The major factors are bounding box sizes, maze routing iterations and the factor sused in formula 7 in the cost funciton. Due to space limit, we only present the philosophy in how to set them and user can refer to the source code to find the exact value. For bouding box sizes, Fastroute starts at a small value to limit detour at the beginning of routing process. It increases as maze routing iterations proceed but is capped at 20% of the entire grid graph size because a larger bounding box would not help to further eliminate congestion and would increase runtime in vain. FastRoute run maze routing for at most 100 iterations or as soon as it elimiates all violation. For the coefficient in formula 7, H keeps growing to increase the strength to push away nets from congested edges. S is set to 10

VI. Conclusion

In this paper, we develop a new global routing tool that focuses on reducing routing congestion and the number of vias. If the runtime bonus used in ISPD08 is considered, FastRoute 4.1 outperforms every single academic global router. In addition, it reduces the via count significantly during global routing.

Our future work will focus on how to control maze routing so that it can make more effective balance between reducing congestion and keeping wirelength small.

References

[1] [1] J. Hu and S. S. Sapatnekar, "A survey on multi-net global routing for integrated circuits," Integration, the VLSI Journal, vol. 31, no. 1, pp. 1-49, 2001.

[2] [2] M. D. Moffitt, J. A. Roy and I. L. Markov, "The coming of age of (academic) global routing," invited paper *Proc. of Intl. Symp. on Physical Design*, pp 148-155, 2008.

[3] [3] C.Albrecht, "Global routing by new approximation algorithms for multicommodity flow," *IEEE trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 5, pp. 622-631, 2001.

[4] [4] M. Cho and D. Z. Pan, "BoxRouter: A new global router based on box expansion and progressive ILP," *Proc. of Design Automation Conference*, pp. 373-378, 2006.

[5] [5] R. Kastner, E. Bozorgzadeh and M. Sarrafzadeh, "Pattern Routing: Use and Theory for Increasing Predictability and Avoiding Coupling," *IEEE trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 7, pp. 777-790, 2002.

[6] [6] R.T. Hadsell and P.H. Madden, "Improved global routing through congestion estimation," *Proc. of Design Automation Conference*, pp. 28-31, 2003.

[7] [7] M. Cho, K. Lu, K. Yuan and D. Z. Pan, "BoxRouter 2.0: Architecture and Implementation of a hybrid and robust global router," *Proc. of Intl. Conf. on Compuer-Aided Design*, pp. 503-508, 2007.

[8] [8] M. M. Ozdal and M. D.F. Wong, "ARCHER: a history-driven global routing algorithm," *Proc. of Intl. Conf. on Computer-Aided Design*, pp. 481-487, 2007.

[9] [9] J.-R. Gao, P.-C. Wu, and T.-C. Wang, "A new global router for modern designs," *Proc. Asia and South Pacific Design Automation Conf.*, 2008.

[10] [10] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang, "NTHU-Route 2.0: A Fast and Stable Global Router," *Proc. IEEE/ACM Intl. Conf. on Compuer-Aided Design*, 2008.

[11] [11] Y. Chen, C. Hsu and Y. Chang"High-Performance Global Routing with Fast Overflow Reduction," *Proc. Asia and South Pacific Design Automation Conf.*, 2009.

[12] [12] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," *Proc. IEEE/ACM Intl. Conf. on Compuer-Aided Design*, pp. 496-502, 2007.

[13] [13] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," *Proc. of Int'l Symp. on Field-Programmable Gate Arrays*, pp. 111-117, 1995.

[14] [16] M. D. Moffitt, "MaizeRouter: Engineering an effective global router," *Proc. Asia and South Pacific Design Automation Conf.*, 2008.

[15] [17] J. A. Roy and I. L. Markov, "High-performance Routing at the Nanometer Scale," *IEEE trans. on Computer-Aided Design of Integrated Circuits and Systems*, VOL. 27, NO. 6, pp. 1066-1077, 2008.

[16] [14] M. Pan and C. Chu, "FastRoute: A step to integrate global routing into placement," *Proc. of Intl. Conf. on Computer-Aided Design*, pp. 464-471, 2006.

[17] [15] M. Pan and C. Chu, "FastRoute 2.0: A high-quality and efficient global router," *Proc. of Asia and South Pacific Design Automation Conf.*, 2007.

[18] [18] Y. Zhang, Y. Xu and C. Chu, "FastRoute 3.0: a fast and high quality global router based on virtual capacity," *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, pp. 344-349, 2008.

[19] [18] Y. Xu, Y. Zhang and C. Chu, "FastRoute 4.0: global router with efficient via minimization," *Proc. Asia and South Pacific Design Automation Conf.*, 2009.

[20] [19] T. Wu, A. Davoodi and J. Linderoth, "GRIP: scalable 3D global routing using integer programming," *Proc. of Design Automation Conference*, pp. 320-325, 2009.

[21] [20] C. Chu, "FLUTE: Fast lookup table based wirelength estimation technique," *Proc. of Intl. Conf. on Compuer-Aided Design*, pp. 696-701, 2004.

[22] [21] C. Chu and Y.-C. Wong "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70-83, 2008.

[23] [22] J. Cong, M. Xie and Y. Zhang, "MARS - A multilevel fullchip gridless routing system," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 3, pp. 382-394, 2005.

[24] [23] J. Westra, C. Bartels and P. Groeneveld, "Probabilistic congestion prediction," *Proc. of Intl. Symp. on Physical Design*, pp 204-209, 2004.

[25] [24] J. Westra and P. Groeneveld, "Is probabilistic congestion estimation worthwhile?" *Proc. Intl. Workshop on System-Level Interconnect Prediction(SLIP)*, pp. 99-106, 2005.

[26] [25] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM Journal of Applied Mathematics*, vol. 14, pp. 255-265, 1966.

[27] [26] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik, vol. 1*, pp. 269-271, 1959.

[28] [27] D. von Seggern. CRC Standard Curves and Surfaces. Boca Raton, FL: CRC Press, 1993.

[29] [28] http://www.sigda.org/ispd2007/rcontest/.

[30] [29] http://www.ispd.cc/contests/ispd08rc.html.