A Timing-constrained Algorithm for Simultaneous Global Routing of Multiple Nets [*]

Jiang Hu *Sachin S. Sapatnekar*

Department of ECE, University of Minnesota, Minneapolis, MN 55455, USA

ABSTRACT

In this paper, we propose a new approach for VLSI interconnect global routing that can optimize both congestion and delay, which are often competing objectives. Our approach provides a general framework that may use any single-net routing algorithm and any delay model in global routing. It is based on the observation that there are several routing topology flexibilities under timing constraints. These flexibilities are exploited for congestion reduction through a network flow based hierarchical bisection and assignment process. Experimental results on benchmark circuits are quite promising.

1. INTRODUCTION

As interconnect is becoming one of the dominant factors affecting VLSI performance in deep submicron era, the requirements on the quality of interconnect routing are becoming stricter, and the routing problem is consequently growing more difficult to solve. In global routing, a given set of global nets are routed coarsely, in an area that is conceptually divided into small regions called routing cells. For each net, a routing tree is specified only in terms of the cells through which it passes. The number of allowable routes across a boundary between two neighboring cells is limited. One fundamental goal of global routing is to route all the nets without overflow, i.e., the number of wires across each boundary does not exceed its supply. Various works have been proposed, for example, sequential approach [1], rip-up-and-reroute technique [2], multicommodity flow based algorithm [3] and hierarchical methods [4, 5].

When interconnect becomes a performance bottleneck in deep submicron technology, merely minimizing congestion is not adequate. In later works [6, 7, 8], interconnect delays are explicitly considered during global routing. In [6], each net is initially routed in SERT-C [9], after which the congested area is ripped up and rerouted by locally applying a multicommodity flow algorithm. In [7], beginning with a set of routing trees satisfying timing constraints for each net, a multicommodity flow method is applied to choose a single routing tree for each net, such that the congestion is minimized. At places where overflow occurs, the wires are ripped up and rerouted through maze routing in which the timing objective is combined with wirelength and congestion. For global routing on standard cell designs, the work of [8] incorporates the timing issue with an iterative deletion technique. In [10], timing constraints are combined with a top-down hierarchical bisection and assignment method for FPGA routing where the switch delay dominates and wire delays are neglected.

In global routing, congestion and delay are often competing objectives. In order to avoid congestion, some wires must make detours, and the signal delay may consequently suffer. In this paper, we propose a new approach to global routing such that both congestion and timing objectives can be optimized at the same time. One key observation is that there are several routing topology flexibilities that can be traded into congestion reduction while ensuring that timing constraints are satisfied. We express these flexibilities through the concepts of a soft edge and a slideable Steiner node and exploit them in global routing through hierarchical bisection and assignment as in [5, 10]. However, due to interdependence on timing slack consumption and the presence of slideable Steiner nodes, the assignment is not straightforward as in [5, 10]. We propose a network flow formulation so that the timing slack consumptions are adaptive to the congestion distributions. Finally, a timing-constrained rip-up-and-reroute process is performed to overcome any inabilities of the hierarchical approach in satisfying congestion constraints. Since the timing performance of initial routing solution can be preserved, our method provides a general framework that can accommodate any single-net routing scheme and can be applied on any delay model.

2. PRELIMINARIES

2.1. Problem background and congestion metrics

We are given a set of nets $\mathcal{N} = \{N^1, N^2, ...\}$, with each net N^i being defined by a set of pins $V^i = \{v_0^i, v_1^i, ...\}$, where the source or driver is denoted as v_0^i. We consider routing in two layers, one for horizontal wires and the other for vertical wires. As in conventional global routing, we tessellate the entire routing region into an array of uniform rectangular cells. We represent this tessellation as a grid graph $G(V_G, E_G)$, where $V_G = \{g_1, g_2, ...\}$ corresponds to the set of grid cells, and a grid edge $b_k \in E_G$ corresponds to the boundary between two adjacent grid cells. We will refer to a grid edge simply as a *boundary*. The number of wires that are allowed to cross a boundary is limited by an upper bound, which is called the *supply* of the boundary and expressed as $s(b)$. During the routing, the number of wires that are routed across a boundary b is designated as the *demand* $d(b)$. The *overflow* $f_{ov}(b)$ at boundary b is $\max(d(b) - s(b), 0)$. The *demand density* for a boundary b is defined as $D(b) = d(b)/s(b)$. We use the metrics of the maximum demand density $D_{max} = \max_{b \in E_G}\{D(b)\}$ and the total overflow $F_{ov} = \sum_{\forall b \in E_G} f_{ov}(b)$ to evaluate the congestion reduction.

2.2. Soft edges

A routing tree T is described by a set of nodes $V = \{v_0, v_1, v_2...\}$ and a set of edges $E = \{e_1, e_2...\}$. The location for a node v_i is specified by its coordinates x_i and y_i. An edge in E is uniquely identified by the node pair (v_i, v_j) or the notation e_{ij} interchangeably, where v_i is the upstream end of this edge.

Routing in the rectilinear space requires that each edge has a fixed orientation, either horizontal or vertical. For example, when we consider the connection between v_0 and v_3 in Fig. 1(b), we usually choose an upper L-shaped or a lower L-shaped connection, both of which are indicated in the dotted lines. In each case, a bend (degree-two Steiner) node v_4 or v_4' is induced. Since there are many uncertainties at the global routing stage, i.e., the detailed routes are not determined, the specifications on delays need to capture the nature of the delay functions without being completely exact. In this spirit, these two routes and many multi-bend monotone

[*]This work is supported in part by the NSF under contract CCR-9800992 and the SRC under contract 98-DJ-609.

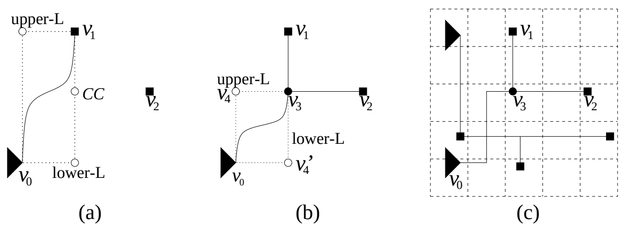

Figure 1. Routing with soft edges.

routes connecting v_0 and v_3 can be regarded to have same delay performance, if the extra delay from a small number of vias can be neglected for the same reason.[1] However, these routes may have different influences on the congestion distribution when we consider multiple nets in global routing. Before these different influences become clear, it is better to keep the flexibilities on routes rather than to embed them into the rectilinear space prematurely. Based on this observation, we may connect v_0 and v_3 with a *soft edge*, which is defined as follows.

Definition 1: A *soft edge* is an edge connecting two nodes $v_i, v_j \in V$, such that: 1. $x_i \neq x_j$ and $y_i \neq y_j$, 2. its edge length l_{ij} is fixed, 3. the precise edge route between v_i and v_j is not determined.

We will refer to the traditional edges in a rectilinear tree with fixed orientations as *solid edges*. The soft edge connection between v_0 and v_3 is shown as a solid curve in Figure 1(b). By keeping edge e_{03} soft, we can maintain the flexibility on routes connecting v_0 and v_3 until we consider congestion in global routing with other nets. In Figure 1(c), in the presence of another net, a Z-shaped route for e_{03} is chosen to reduce congestion without hurting the delay.

In fact, the concept of soft edge is also useful in single-net routing. Consider the process of constructing the Steiner minimum tree in Fig. 1(b) in a manner similar to Prim's minimum spanning tree algorithm. If we begin by connecting sink v_1 to source v_0, and arbitrarily choose the upper-L connection, the Steiner minimum tree will not be reached. Instead of fixing the edge orientation immediately, we connect them with soft edge e_{01}, as shown in Fig. 1(a). Then, node v_2 is joined to edge e_{01} at the closest connection (CC) point. The *closest connection* (CC) *point* between a node v_k and an edge e_{ij} is defined by its coordinates x_{CC} and y_{CC} such that $x_{CC} = median(x_i, x_j, x_k)$ and $y_{CC} = median(y_i, y_j, y_k)$. In Figure 1(b), Steiner node v_3 is introduced at the CC point and the Steiner minimum tree is obtained.

2.3. Delay properties and slideable Steiner nodes

To measure the signal delay of an interconnect, we employ the Elmore delay model. Although occasional large errors make Elmore delay unsuitable for critical nets [11], it has a role in global routing because of its fidelity [9] and simplicity, and is a reasonable model considering that the routing in global stage is coarse and the number of nets may be very large.

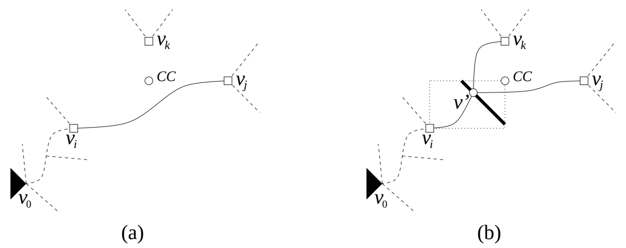

Figure 2. Connection between node v_k and soft edge e_{ij}.

[1] Later in our algorithm, we will penalize the excessive use of vias.

For a general form of a partially constructed routing tree, shown in Figure 2(a), let us consider the process of obtaining an optimal connection between node v_k and edge e_{ij}. The dashed lines are other nodes and edges of this routing tree, and CC represents the closest connection point between v_k and e_{ij}. We wish to search for an optimal connection point within the bounding box defined by v_i and CC. Suppose we connect v_k to e_{ij} at point $v'(x', y')$, as indicated in Figure 2(b). Let $z = |x' - x_i| + |y' - y_i|$. If the delay at an arbitrary sink v_a is $t(v_a)$ and the its required arrival time is $RAT(v_a)$, then the *delay slack* $s(v_a) = RAT(v_a) - t(v_a)$. The *timing slack* $S(T^i)$ for a routing tree T^i on the net N^i is the minimum delay slack among all the sinks in this net. If the objective is to minimize wire cost subject to timing constraints, the optimal connection (Steiner) point here is a point with a non-negative net timing slack, lying as close to CC as possible. The optimal connection point is likely to be a non-Hanan point such as v' in Fig. 2(b). A similiar conclusion is derived in [12].

A careful observation tells us that there are often many Steiner node locations for a specific value of z. The set of locations for a given value of z form a locus as illustrated by the thickened segment in Figure 2(b). When we slide the Steiner node v' along this locus, the lengths of its incident edges are preserved and so is the delay at each sink. Similar to the rationale for soft edges, we only specify this locus instead of a point for this Steiner node and call it as *slideable Steiner node* (SSN).

3. ALGORITHM

3.1. Algorithm overview

This algorithm includes three phases: (1) performance driven routing for each net, (2) **HBA**: hierarchical bisecting of routing regions and assigning soft edges to boundaries along the bisector, and (3) **TRR**: timing-constrained rip-up-and-reroute.

In phase 1, each net is routed to meet its timing constraints without considering congestion. Any single-net performance driven routing method, e.g., P-tree [13], RATS tree [14] or MVERT [12], can be applied here. Besides satisfying timing constraints, each routing tree should be soft. This can be achieved through utilizing soft edges during routing as in the example of Fig. 1 or replacing L-shaped connections in the results with soft edges. Thus, at the end of phase 1, timing-constrained routing trees are generated along with topology flexibilities to be exploited in the subsequent phases.

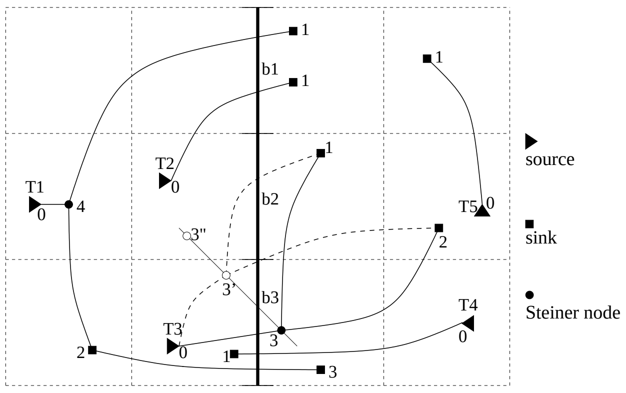

Figure 3. An example of bisection.

In phase 2, a routing region is recursively bisected into subregions in a top-down manner. At the topmost level, the whole routing region is bisected into left(upper) and right(lower) halves with the same or similar size by a bisector line which is formed by a column(row) of consecutive vertical(horizontal) grid cell boundaries. For example, in Fig. 3, the thickened bisector line is composed of three boundaries, b_1, b_2 and b_3. Each soft edge that intersects this bisector is assigned to a boundary. After the assignment, a pseudo-pin is inserted into the soft edge at the assigned boundary, and therefore this soft edge is split into two new soft edges that belong to two separate subregions. One assignment for the example in Fig. 3 is shown in Fig. 4. In the next hierarchical level, bisections and assignments are applied on the left(upper) and right(lower) half region along an orthogonal orientation. This process is repeated until the subregion is a single grid cell or a pair of neighboring grid cells. Thus, at the end of this process, the route for each soft edge is specified to the detailed level of grid cells it goes through.

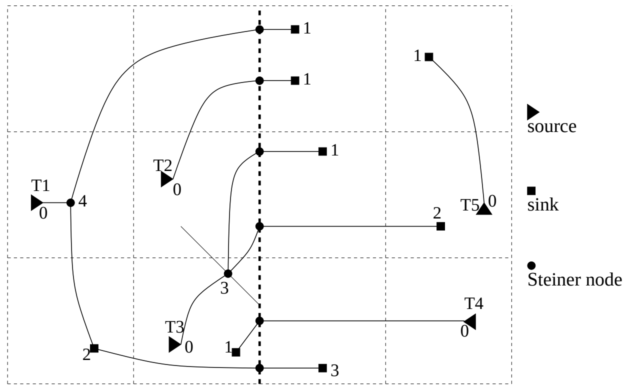

Figure 4. An assignment result from network flow solution.

The crucial part is to determine how to assign the soft edges to the boundaries on the bisector line. The basic goal is to assign *all* of the soft edges without exceeding any boundary supply and without causing any delay violations. In order to make the assignment feasible, sometimes it is necessary to allow some wires to detour, which inevitably increases delay, i.e., some timing slack is consumed. In addition to ensuring absence of delay violations, it is naturally desirable that the consumption of the timing slack is minimized, since the timing slack may be needed in the subsequent levels of bisection and assignment. These objectives are achieved through a min-cost network flow formulation.

The hierarchical bisection and assignment in phase 2 is a method of divide-and-conquer that has the advantage of simplifying the problem nature. It reduces a two-dimensional problem into one dimension. However, a decision at a higher hierarchical level may overlook the needs at a lower level. In phase 2, any soft edge that could not be assigned in the network solution is temporarily assigned to a boundary such that the maximum demand density is minimized and no delay violation is incurred. These residual overflows will be cleaned in phase 3.

The third phase is a timing-constrained rip-up-and-reroute process. It is similar to traditional rip-up-and-reroute except that a constraint on edge length is imposed to ensure no timing violation. It rips up the edges on a set of most congested boundaries and reroutes them through maze routing. The cost in maze routing is defined as the summation of demand densities over all boundaries that a soft edge passes through.

3.2. Basic network formulation

After one bisection, the assignment problem is formulated as:

Assignment problem: Given a bisector line B composed of a set of consecutive boundaries $\{b_1, b_2, ...\}$, and a set of soft edges $E_X = \{e^i_{jl} | e^i_{jl}$ intersects $B\}$, assign each soft edge to a boundary $b_k \in B$ such that there is no overflow on any boundary $b_k \in B$ or no delay violation on any routing tree T^i which has at least one soft edge $e^i_{jl} \in E_X$, and the timing slack consumption is minimized.

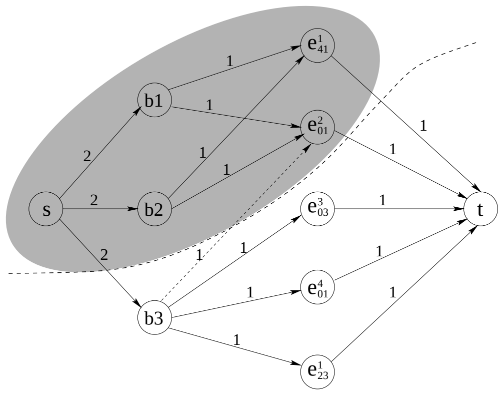

Figure 5. Network formulation of the example in Fig. 3 without considering SSN. The number on each arc is its capacity.

We solve this problem through a formulation of the network flow problem and applying a min-cost max-flow algorithm [15] on it. The network $G_F(V_F, A_F)$ is a directed graph consisting of a set of vertices V_F and arcs A_F. The vertex set V_F includes all boundaries in B and soft edges in E_X, plus a source s and target t. For the bisection in Fig. 3, its corresponding network is illustrated in Fig. 5. We do not use SSN at this moment for simplicity and only e^3_{03} in T^3 is included in the network. The usage of SSN will be introduced in section 3.4.. There are three types of arcs: (1) from source s to every boundary vertex, (2) from some boundary vertices to some soft edge vertices, (3) from every soft edge vertex to the target t. Each arc has a cost and a capacity associated with it. For each type 1 arc, its cost is 0 and its capacity is the corresponding boundary supply. In this example, we assume that each boundary has a supply of 2. For each type 2 arc, its capacity is 1 and its cost will be defined later. For each type 3 arc, its capacity is 1 and its cost is 0.

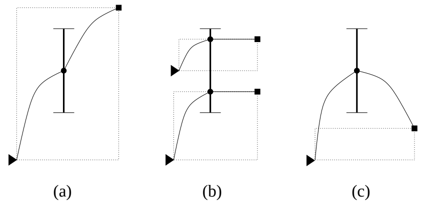

Figure 6. Relative positions of a boundary and a soft edge.

An arc from a boundary vertex to a soft edge vertex implies a candidate assignment between them. Not every pair of boundary and soft edge vertices is automatically qualified for constructing a type 2 arc between them. For any boundary and any soft edge, there are three relative positions between them as shown in Fig. 6. In Fig. 6(a), the boundary lies entirely within (the bounding box of) the soft edge. If we choose an assignment of the soft edge to this boundary, there will be no change in the length of the soft edge, and two vias are induced. If a boundary lies partially within the bounding box of a soft edge, as in Fig. 6(b), we have an *L-intersection* between the boundary and the soft edge, where no change in the soft edge length is required and one via is induced. In either of these two cases, we can always set up an arc between them without affecting the delay. These arcs are called *basic arcs*, and they are the solid type 2 arcs in Fig. 5. The third situation is shown in Fig. 6 (c), where the soft edge does not intersect with the boundary. In this case, an assignment on this pair will require a wire detour, and we need to check whether or not this may cause any delay violation. An arc can be constructed for such a pair only if the assignment on this pair will not cause any delay violation. For the example in Fig. 3, if the timing slack of T^2 remains non-negative when the soft edge e_{01}^2 goes through boundary b_3, then an arc (a dashed line) between them is constructed in Fig. 5. We call such a construction as a *soft edge expansion* and each expansion implies a timing slack consumption.

We categorize the trees across the bisector line B into single-crossing trees and multi-crossing trees, which are the trees that cross B only once (such as T^2 in Fig. 3) and more than once (such as T^1 in Fig. 3), respectively. Initially, we construct all the basic arcs for all the soft edges in E_X and perform an expansion for all the soft edges that belong to single-crossing trees. The expansions of edges in multi-crossing trees will be discussed in the next section.

The cost of a type 2 arc is defined according to the timing slack of its corresponding tree, since one major objective is to minimize timing slack consumption. If the timing slack of tree T^i is $S_{old}(T^i)$ before the assignment, and is $S_{new}(T^i)$ if its soft edge e_{jl}^i is assigned to boundary b_k, then we define the arc cost as:

$$cost(b_k, e_{jl}^i) = \frac{S_{old}(T^i)}{S_{new}(T^i)}. \tag{1}$$

It can be seen that if a soft edge intersects with a boundary entirely or partially, its corresponding type 2 arc has a cost of unity. As a secondary objective, we hope to reduce the number of vias in the wiring. Therefore, for the situation in Fig. 6(b), we reduce its cost by a small user-specified offset $\epsilon, 0 < \epsilon < 1$.

3.3. Construction of arcs for multi-crossing trees

Generally speaking, adding a type 2 arc between a boundary vertex and a soft edge vertex may increase the likelihood of obtaining a feasible network flow solution. Hence, a soft edge expansion is usually desired as long as no delay violation is incurred. One issue that was not discussed in the last section is the procedure for those soft edges that belong to multi-crossing trees, such as T^1 in Fig. 3. The difficulty here is that the timing slack consumptions for the soft edges are correlated. For some specified timing constraints, whether a soft edge can be expanded, or how far it can be expanded, depends on whether other crossing edges in the same tree are expanded, and how far they have been expanded. For example, in Figure 3, the expansion of e_{41}^1 depends on whether e_{23}^1 has been expanded and how far, i.e., to b_2 or to b_1. In fact, these soft edges compete with each other on a common timing slack resource, which must be allocated properly.

We solve this difficulty by identifying the necessary expansions through the min-cut method. It is well known that the max-flow equals the forward capacity of the $s-t$ min-cut in a network flow problem[15]. In the beginning, we run a max-flow algorithm on the partially constructed network to obtain an $s-t$ min-cut $(X, \bar{X}), s \in X, t \in \bar{X}$. The forward capacity of this cut is denoted by $U_{min}(X, \bar{X})$. If $U_{min}(X, \bar{X}) \geq |E_X|$, then it is guaranteed that every soft edge can be assigned to a boundary without any overflow, and thus, no more expansion is necessary. Otherwise, the maximum feasible flow is less than the number of soft edges to be assigned, thus we need to increase the capacity of the min-cut through additional soft edge expansions. In the example for Fig. 3, before the expansion for multi-crossing trees, the min-cut is indicated in the dashed curve in Fig. 5, where the vertices in X are in the shaded region and vertices in $\bar{X}$ are unshaded. We can see that the forward capacity $U_{min}(X, \bar{X}) = 4$ while there are 5 soft edges that need to be assigned, thus, we need to expand some soft edge(s) from the multi-crossing tree T^1 if possible.

The min-cut result shows us not only whether more expansions are necessary but also the congestion distribution information or where to make the expansion. Every forward arc in the min-cut must be saturated [15], e.g., (s, b_3), (e_{41}^1, t) and (e_{01}^2, t) are saturated. If a soft edge vertex e_{jl}^i is in X, its downstream arc must be saturated and therefore, it can always be assigned to a boundary without inducing overflow, i.e., it is not in a congested area. On the other hand, if a boundary vertex b_k is in $\bar{X}$ (and not all of its downstream arcs are saturated), its upstream arc must be saturated and the soft edges corresponding to its downstream vertices are located in a congested area. Adding an arc from a boundary vertex $b_k \in X$ to a soft edge vertex $e_{jl}^i \in \bar{X}$ matches a soft edge in a congested area to an uncongested boundary.

Lemma 1: *The necessary and sufficient condition to increase the max-flow f_{max} of a network is to add a forward arc between X and $\bar{X}$ for every min-cut $(X, \bar{X})$ with $U_{min}(X, \bar{X}) = f_{max}$.*

We make a sweep among all the soft edges in multi-crossing trees and pick at most one soft edge from each tree to expand in order to increase the capacity of min-cut. More precisely speaking, for each multi-crossing tree T^i, from all the $b_k \in X$ and $e_{jl}^i \in \bar{X}$ pairs, we choose one with minimum cost to add an arc between them if no delay violation is induced. After one iteration of expansions, we run the max-flow min-cut algorithm again to repeat this process until $U_{min}(X, \bar{X}) \geq |E_X|$ or no more feasible arc can be found. Note that the timing slack computation in a later iteration of expansions should account for any wire detour in other soft edges of the same tree in previous expansions. In the example in Fig. 5, We can make an expansion between $b_2 \in X$ and $e_{23}^1 \in \bar{X}$ if no delay violation is induced, and then the network problem becomes feasible. The iterative min-cut and expansion technique makes the allocation of timing slack in multi-crossing trees adaptive to the congestion distribution, and expansions are made only when necessary, without waste.

3.4. Utilization of slideable Steiner nodes (SSN)

In phase 1, if we use the MVERT algorithm together with soft edges, we can have a slideable Steiner node that provides extra flexibility in routing. The appealing feature of SSN is that when we slide it along its locus, the timing performance is preserved.

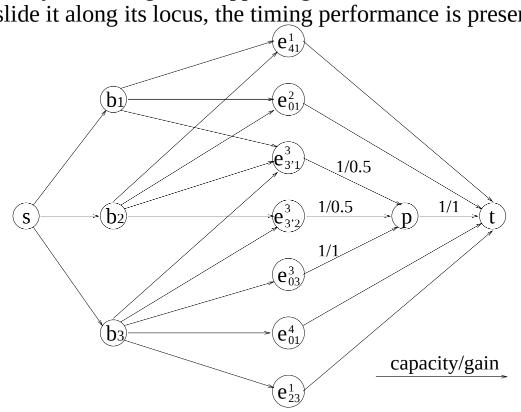

Figure 7. Network formulation considering SSN.

The positions of a SSN within a grid cell do not affect wire congestion distributions, hence we can consider one arbitrary position for a SSN within a grid cell. For each SSN whose locus intersects with B, we consider only two candidate positions, each on a different side of the bisector line B, such as v_3^3 and $v_{3'}^3$ in Fig. 3. We need to consider candidate positions on both sides of B, since they result in remarkably different intersections between their incident soft edges and the bisector line B. On each side of B, we only consider the grid cell that has a boundary in B such that this boundary intersects the locus of the SSN, since the SSN position in this grid cell can provide the maximum overlap between its incident soft edge(s) and B. For example, in Fig. 3, $e_{3'2}^3$ intersects with two boundaries b_2 and b_3, while $e_{3''2}^3$ would intersect only with b_2. It is evident that a larger overlap implies a larger number of basic arcs which are preferred as they will not consume timing slacks. For v_3^3 and $v_{3'}^3$, all three associated soft edges $e_{3'1}^3$, $e_{3'2}^3$ and e_{03}^3 are included in the vertices in the network as shown in Fig. 7. Obviously, e_{03}^3 cannot be assigned simultaneously with $e_{3'1}^3$ or $e_{3'2}^3$. This exclusiveness constraint can be instantiated through adding a pseudo-vertices p and formulating a generalized network flow model [15], where each arc has a gain factor associated with it. For example, the amount of flow will reduce 50% after passing through an arc with gain factor of 0.5. We solve this generalized network flow problem through Wayne's algorithm [16].

After the assignment, only one of the candidate SSN positions is selected. The locus of the SSN is truncated at the intersection with B, and the part where the selected position located would be retained, as shown in Fig. 4.

4. EXPERIMENTAL RESULTS

The experiments aim to test the effect of the proposed algorithm on both timing and congestion. Traditional rip-up-and-reroute(RR) and timing-constrained rip-up-and-reroute(TRR) methods are tested together with our algorithm(HBA+TRR) on the same set of circuits. The circuits that we tested belong to the CBL/NCSU benchmark suite whose statistics are shown in Table 1. The initial routing trees are obtained through MVERT[12] algorithm so that they must satisfy timing constraints. The results are listed in Table 2.

Table 1. Benchmark circuits.

Circuit	# modules	# nets	# pins
apte	9	45	162
ami33	33	85	480
ami49	49	390	913
xerox	10	203	696

The congestion results are expressed in terms of total overflow F_{ov} and the maximum demand density D_{max}. The congestion results from rip-up-and-reroute(RR) are generally good. The congestion results from TRR are much worse than RR, because the algorithm may get stuck in a deadlock and fail to find a solution under timing constraints. A naive combination of timing constraints with rip-up-and-reroute does not work, and a crafted approach is necessary to optimize these two competing objectives simultaneously. Table 2 also shows the congestion results from HBA for reference. Even though they are usually better than TRR, they are not ideal yet and not comparable with those of RR, and should be considered as intermediate results. When we combine TRR with HBA, the congestion results are found to be good and are mostly better than even RR, which does not satisfy the timing constraints. Since hierarchical approach is better at a global planning level while rip-up-and-reroute is specialized to find local and more detailed routes, it is natural that a combination of these two complementary approaches can yield a good result on congestion reductions. When we compare the timing results, it is not surprising that only RR causes delay violations while there is no delay violation in the results from TRR or our algorithm. The percentage of nets with delay violations from RR are listed in column 6, and ranges from $6-32\%$.

The total CPU time for three phases of our algorithm on each circuit are listed in the rightmost column in seconds. Since each circuit has different number of nets and the number of pins on one net may be between two and several dozens, it would be more interesting to evaluate the average CPU time on each 2-pin net as a normalized comparison. The third column, $|E|$ gives the total number of soft edges from the initial routing trees in each circuit. It is conceivable that the formulation of soft edges is equivalent to a decomposition to 2-pin nets. Based on this data, the average CPU time is found to be 0.06 second/2-pin-net in the worst case.

5. CONCLUSION

We propose a new approach to timing-constrained global routing. We formalize the routing tree topology flexibilities under timing constraints through the concepts of a soft edge and a slideable Steiner node, and trade these flexibilities into congestion reduction while the timing constraints are satisfied. Experimental results show that our proposed algorithm can achieve good congestion results while satisfying timing constraints.

REFERENCES

[1] C. Chiang, C. K. Wong and M. Sarrafzadeh, "A weighted Steiner tree-based global router with simultaneous length and density minimization," *IEEE Trans. on CAD*, Vol. 13, No. 12, pp. 1461-1469, Dec., 1994.

[2] B. S. Ting and B. N. Tien, "Routing techniques for gate array," *IEEE Trans. on CAD*, Vol. CAD-2, No. 4, pp. 301-312, Oct., 1983.

[3] R. C. Carden IV, J. Li and C.-K. Cheng, "A global router with a theoretical bound on the optimal solution," *IEEE Trans. on CAD*, Vol. 15, No. 2, pp. 208-216, Feb., 1996.

[4] M. Burstein and R. Pelavin, "Hierarchical wire routing," *IEEE Trans. on CAD*, Vol. CAD-2, No. 4, pp. 223-234, Oct., 1983.

[5] M. Marek-Sadowska, "Route planner for custom chip design," *Proc. ICCAD*, pp. 246-249, 1986.

[6] D. Wang and E. S. Kuh, "Performance-driven interconnect global routing," *Proc. of the IEEE Great Lakes Symp. on VLSI*, pp. 132-136, 1996.

[7] J. Huang, X.-L. Hong, C.-K. Cheng and E. S. Kuh, "An efficient timing-driven global routing algorithm," *Proc. DAC*, pp. 596-600, 1993.

[8] J. Cong and P. H. Madden, "Performance driven global routing for standard cell design," *Proc. ISPD*, pp.73-80, 1997.

[9] K. D. Boese, A. B. Kahng, B. A. McCoy and G. Robins, "Near-optimal critical sink routing tree constructions," *IEEE Trans. on CAD*, Vol. 14, No. 12, pp. 1417-36, Dec. 1995.

[10] K. Zhu, Y.-W. Chang and D. F. Wong, "Timing-driven routing for symmetrical-array-based FPGAs," *Proc. ICCD*, pp.628-633, 1998.

[11] J. Hu and S. S. Sapatnekar, "FAR-DS: full-plane AWE routing with driver sizing," *Proc. DAC*, pp. 84-89, 1999.

[12] H. Hou, J. Hu and S. S. Sapatnekar, "Non-Hanan routing", *IEEE Trans. on CAD*, Vol. 18, No. 4, pp. 436-444, Apr., 1999.

[13] J. Lillis, C. K. Cheng, T. T. Lin and C. Y. Ho, "New performance driven routing techniques with explicit area/delay tradeoff and simultaneous wire sizing," *Proc. DAC*, pp. 395-400, Jun. 1996.

[14] J. Cong and C. K. Koh, "Interconnect layout optimization under higher-order RLC model," *Proc. ICCAD*, pp. 713-720, 1997.

[15] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network flows: theory, algorithms, and applications.* Prentice Hall, Upper Saddle River, NJ, 1993.

[16] K. D. Wayne and L. Fleischer, "Fast and simple approximation schemes for generalized flow," *Proc. Symposium on Discrete Algorithms*, pp. 981-982, 1999.

Table 2. Experimental results.

			Rip-up-and-reroute			TRR		HBA		HBA+TRR		
Circuit	Grid	$\|E\|$	F_{ov}	D_{max}	DV %	F_{ov}	D_{max}	F_{ov}	D_{max}	F_{ov}	D_{max}	CPU
apte	45×55	145	0	1.00	9	56	1.50	19	1.83	0	1.00	7.1
ami33.1	19×31	489	0	1.00	21	15	1.29	10	1.57	0	1.00	30.3
ami33.2	19×27	497	0	1.00	12	19	1.25	0	1.00	0	1.00	26.6
ami49.1	43×45	594	2	1.09	17	11	1.18	16	1.36	1	1.09	19.6
ami49.2	44×44	594	0	1.00	12	61	1.56	0	0.94	0	0.94	12.2
xerox.1	24×24	583	2	1.06	6	41	1.38	1	1.06	0	1.00	12.8
xerox.2	28×32	569	0	1.00	11	17	1.22	1	1.05	0	1.00	16.3
xerox.3	41×34	569	3	1.11	32	43	1.33	0	1.00	0	1.00	23.8