A Timing-Constrained Simultaneous Global Routing Algorithm

Jiang Hu and Sachin S. Sapatnekar

***Abstract*—Proposed in this paper is a new approach for VLSI interconnect global routing that can optimize both congestion and delay, which are often competing objectives. The authors' 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 that can be exploited for congestion reduction under timing constraints. These flexibilities are expressed through the concepts of a soft edge and a slideable Steiner node. Starting with an initial solution where timing-driven routing is performed on each net without regard to congestion constraints, this algorithm hierarchically bisects a routing region and assigns soft edges to the cell boundaries along the bisector line. The assignment is achieved through a network flow formulation so that the amount of timing slack used to reduce congestions is adaptive to the congestion distributions. Finally, a timing-constrained rip-up-and-reroute process is performed to alleviate the residual congestions. Experimental results on benchmark circuits are quite promising and the run time is between 0.02 s and 0.15 s per two-pin net.**

***Index Terms*—Global routing, interconnect, layout, performance optimization, physical design, VLSI.**

I. Introduction

As INTERCONNECT is becoming one of the dominant factors affecting very large scale integration (VLSI) performance in the 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. Most commonly, the routing problem is solved in two separate stages: global routing and detailed routing. 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. This problem is NP-complete even if each net has only two pins. Since minimizing congestion is very hard to achieve and is essential for global routing, it has long been a focus of research [1]–[13] in global routing. Most of these works belong to one or a combination of the following genres: the sequential approach, hierarchical methods, linear programming or multicommodity flow based algorithms, and rip-up-and-reroute techniques.

In the sequential approach, the nets are routed one after another. In [1], for each net, a Steiner tree on the grid graph that minimizes the maximum edge weight is sought to minimize the congestion, with the weights being proportional to the density of wires in each routing cell. For any sequential approach, it is hard to decide which net ordering is better than others [14], i.e., each ordering has its own weakness. As a solution to avoid this ordering problem, the hierarchical method [2]–[4] recursively splits the routing region into successively smaller parts. At each hierarchical level, all of the nets are routed simultaneously (often through linear programming) and refined in the next hierarchical level until the lowest level of the hierarchy is reached. Sometimes the whole global routing is formulated and solved through linear programming followed by a randomized rounding [5]. Another method is the application of multicommodity flow model [6]–[8], in which the fractional solutions are rounded to obtain the routing solutions. For global routing on standard cell designs, the work of [9] proposed an iterative deletion technique to avoid the net ordering problem. The works of [10]–[12] first route each net independently, then rip up the wires in congested areas and reroute them to spread out the routing density. The rip-up-and-reroute technique is very practical and popular in industrial applications.

When interconnect becomes a performance bottleneck in deep submicron technology, merely minimizing congestion is not adequate. In later works [15]–[19], interconnect delays are explicitly considered during global routing. In [15], each net is initially routed in SERT-C [20], after which the congested area is ripped up and rerouted by locally applying a multicommodity flow algorithm. In [16], 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. The work of [17] is similar to [16] except that path-based timing constraints are satisfied instead of net-based timing constraints. For global routing on standard cell designs, the work of [18] and [19] incorporates the timing issue with an iterative deletion technique. In [21], 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

Manuscript received December 18, 2000; revised September 7, 2001 and January 21, 2002. This work was supported in part by the National Science Foundation under Contract CCR-9800992 and by the SRC under Contract 98-DJ-609. This paper was recommended by Associate Editor M. Pedram.

J. Hu is with the Department of Electrical Engineering, Texas A&M University, College Station, TX 77840 USA (e-mail: jhu@ece.tamu.edu).

S. S. Sapatnekar is with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: sachin@ece.umn.edu).

Publisher Item Identifier 10.1109/TCAD.2002.801083.

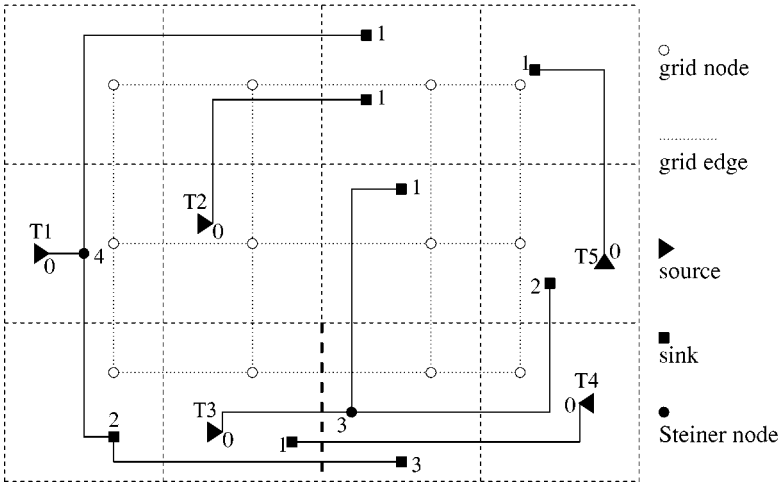

Fig. 1. Tessellation in global routing.

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. These flexibilities include the use of: 1) soft edges; 2) slideable Steiner nodes; and 3) edge elongation, all of which are described later in this paper.

In our algorithm flow, each net is initially routed individually to satisfy its timing constraints, and these routes are used to obtain the timing-constrained routing flexibilities. Next, these flexibilities are traded into congestion reduction through a hierarchical bisection and assignment process followed by a timing-constrained rip-up-and-rerouting. The hierarchical bisection and assignment process here is similar to the works in [21]–[23]. However, due to interdependence of the timing slack consumption among the nets, the assignment is not straightforward as in [21]–[23]. We propose a network flow formulation so that the timing slack consumptions are adaptive to the congestion distributions in the assignment. We further extend the model to be a generalized network flow problem, in order to exploit the flexibility from slideable Steiner nodes. Finally, the timing-constrained rip-up-and-reroute process is performed to overcome any inabilities of the hierarchical approach in satisfying congestion constraints. This method has the advantage that it does not depend on any net ordering. Moreover, it provides a general framework that can accommodate any single-net routing scheme and can be applied on any delay model, since the timing performance of any initial routing solution can be preserved in subsequent stages.

The remainder of the paper is as follows. Section II introduces background knowledge for this work, and Section III briefly shows an overview of our algorithm. The network flow based assignment algorithm is described in Section IV and the computational complexity of the hierarchical bisection and assignment algorithm is analyzed in Section V. The timing-constrained rip-up-and-rerouting method is introduced in Section VI. Experiments are presented in Section VII, and we conclude in Section VIII.

II. Preliminaries

A. Problem Background and Congestion Metrics

We are given a set of nets $\mathcal{N} = \{N^1, N^2, \ldots\}$, with each net N^i being defined by a set of pins $V^i = \{v_0^i, v_1^i, \ldots\}$, 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, as shown in the dashed lines in Fig. 1. We represent this tessellation as a grid graph $G(V_G, E_G)$, where $V_G = \{g_1, g_2, \ldots\}$ corresponds to the set of grid cells, and a grid edge $b \in E_G$ 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 $D(b) = d(b)/s(b)$. In Fig. 1, if the supply for each boundary is 2, there is an overflow of 1 on the thickened boundary and the corresponding demand density is 1.5. 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.

B. Soft Edges

The concept of a soft edge is proposed in [24] for single net routing and buffer insertion with location restrictions. We will show that this concept can also be exploited as a routing flexibility under timing constraints in multinet global routing.

A routing tree T is described by a set of nodes $V = \{v_0, v_1, v_2 \cdots\}$ and a set of edges $E = \{e_1, e_2 \cdots\}$. 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, i.e., v_i is closer to the source node and v_j is closer to the leaf nodes of the tree.

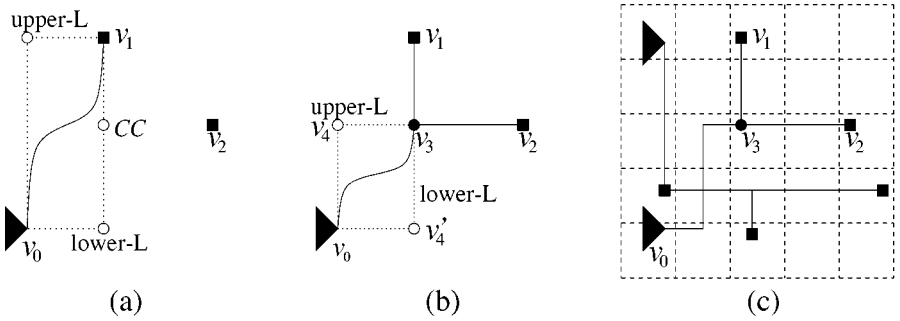

Fig. 2. Routing with soft edges.

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_1 in Fig. 2(a), or v_0 and v_3 in Fig. 2(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 is induced, for example, v_4 or v_4' may be induced in Fig. 2(b). 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 multibend monotone 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 Fig. 2(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 Fig. 2(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. 2(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 can use a soft edge e_{01}, as shown in Fig. 2(a). In order to minimize wirelength, when we consider connecting v_2 to the routing tree, we choose the closest connection (CC) point between v_2 and e_{01}. 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 Fig. 2(b), Steiner node v_3 is introduced at the CC point and the Steiner minimum tree is obtained. The concept of soft edges is especially useful for nets with a large number of pins, where the decision-making process is much more complicated.

C. 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 [25], it has a role in global routing because of its fidelity [20] 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. The works of [20] and [26] describe delay properties with respect to connection location along a maximal segment.[2] The work in [24] shows that these properties hold for soft edges and we will briefly describe them as follows.

For a general form of a partially constructed routing tree, shown in Fig. 3(a), let us consider the process of obtaining an optimal connection between node v_k and edge e_{ij}. 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 $y_{CC} = median(y_i, y_j, y_k)$. 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}. Any connection that is downstream of CC cannot lead to an optimal solution [20]. More specifically, 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 Fig. 3(b). Let z be the Manhattan distance from v' to v_i, i.e., $z = |x' - x_i| + |y' - y_i|$. 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* $s(v_a) = RAT(v_a) - t(v_a)$. We can obtain the following conclusion.[3]

Lemma 1: Under the Elmore delay model, the delay slack at any sink in the routing tree is a convex function with respect to z.

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

[2] A maximal segment is a maximal set of consecutive edges that are either all horizontal or all vertical.

[3] A detailed derivation is available in [24].

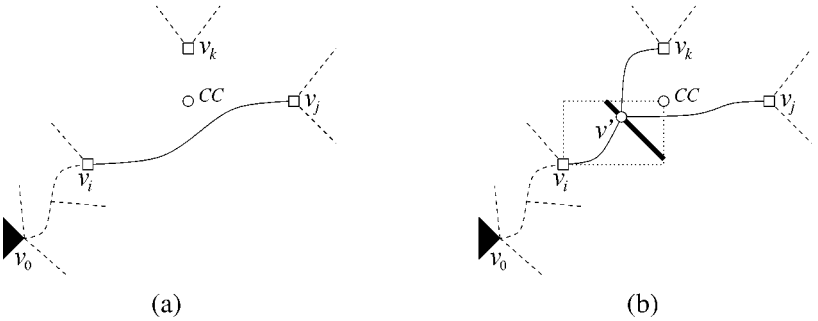

Fig. 3. General case, node v_k is to be connected to edge e_{ij}.

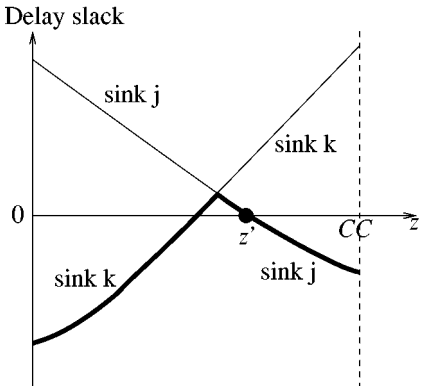

Fig. 4. Delay slack function versus distance z of connection point. Here we overload CC as its Manhattan distance to v_i.

For the example in Fig. 3, if only sink v_j and sink v_k are timing critical, we depict their delay slack functions in Fig. 4. 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; this is illustrated by the thickened contour in Fig. 4. 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; for this particular example, this corresponds to z'. As in this example, the optimal connection point is, in general, likely to be a non-Hanan point. The work of [26] showed this advantage of using non-Hanan points and proposed the MVERT algorithm to perform non-Hanan optimization globally for a routing tree. Based on properties similar to *Lemma 1*, MVERT finds the optimal connection point through a quasi-binary search and obtains significant wire cost reductions.

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 Fig. 3(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 a *slideable Steiner node* (SSN). This is similar to the merging segment in the deferred-merge embedding algorithm [27] for zero skew clock net routing. The concept of a slideable Steiner node provides extra flexibility for the routes of its incident edges and can again be used to reduce the congestion in global routing without degrading timing performance or area.

III. 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 [28], RATS tree [29] or MVERT [26], can be applied here. Besides satisfying timing constraints, each routing tree should be soft, i.e., should not contain any degree-two Steiner node. This can be achieved through utilizing soft edges during routing as in the example of Fig. 2 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. For a net with k sinks, the computational complexity of MVERT is about $O(k^4)$ [26].

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 by a bisector line which is formed by a column (row) of consecutive vertical (horizontal) grid cell boundaries. For example, in Fig. 5, 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 pseudopin 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. 5 is shown in Fig. 6. In the next hierarchical level, bisections and assignments are applied on the left (upper) and right (lower) half regions. 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.

When we make a bisection, we always choose a direction to make the region as close to a square as possible. For example, if a region has more rows (columns) than columns (rows), we will bisect along the horizontal (vertical) direction. At each direction, the bisection could be at different locations. We choose a location such that the ratio of the number of crossing soft edges to the total capacity along the bisector line is the maximum, i.e., we make bisection at the most congested place. Since our hierarchical approach proceeds in a top-down manner, more favor

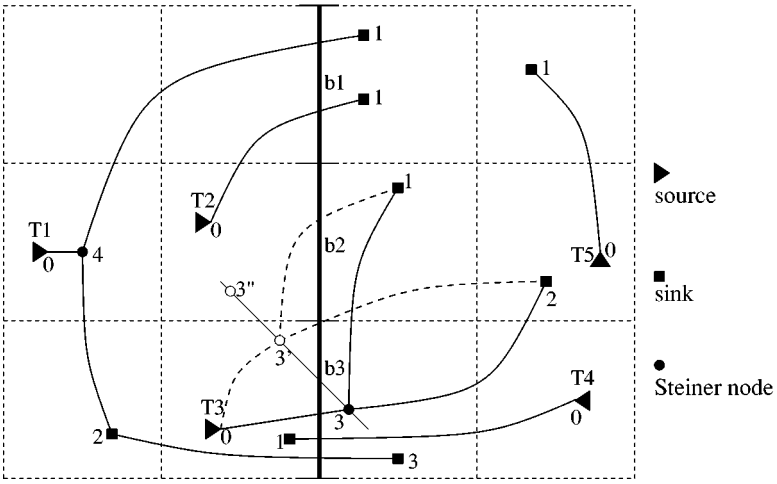

Fig. 5. An example of bisection.

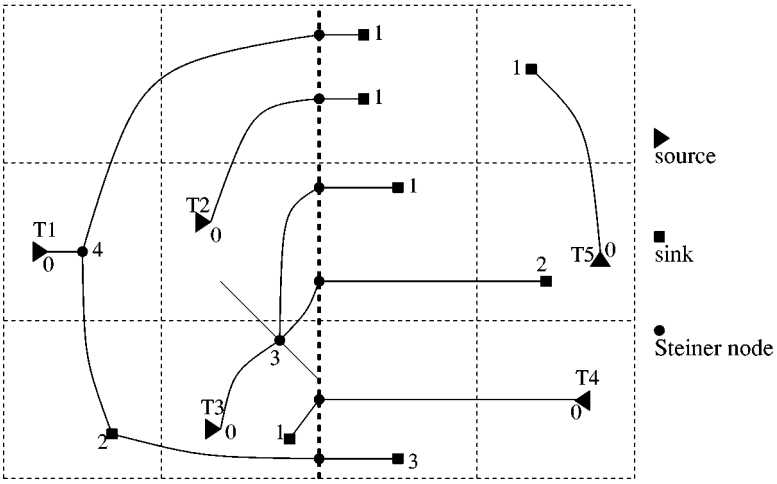

Fig. 6. An assignment result from network flow solution.

is given to the higher hierarchical level and we try to solve the most difficult part at a higher level. Similar bisection strategy is employed in the work of [22]. Although quadrisection as in [4] is better at handling congestion, integrating it with timing constraints is very difficult.

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. The absence of delay violation implies that the delay slack for each net is nonnegative. 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 to reduce congestion. 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. Because of the involvement of timing issues, this formulation is not as straightforward as that in [21]–[23]. We run a min-cost max-flow algorithm [30] to solve this network flow problem. The min-cost flow algorithm we employed in practice is the capacity scaling algorithm [30], which can give an optimal solution in a pseudopolynomial time.

The hierarchical bisection and assignment in phase 2 is a method of divide-and-conquer that has the advantage of simplifying the problem nature. In this global routing approach, it reduces a two-dimensional (2-D) problem into one dimension. The price that this simplification inevitably pays is on congestion reduction, since 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 and the location of each SSN is readjusted to minimize the congestion. 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 the square of demand densities over all boundaries that a soft edge passes through, and these densities are dynamically updated. The edge

length can be elongated to the extent that no delay violation is incurred. The procedure for transforming timing slack into an edge length slack is described in Section VI.

IV. Network Flow-Based Assignment Algorithm

A. Basic Network Formulation

After one bisection, the assignment problem is formulated as follows.

Assignment Problem: Given a bisector line B composed of a set of consecutive boundaries $\{b_1, b_2, \ldots\}$, 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 $b_k \in B$ and 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.

We solve this problem through a formulation of the network flow problem and applying a min-cost max-flow algorithm 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. 5, its corresponding network is illustrated in Fig. 7. We do not use SSNs 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 IV-C. There are three types of arcs: 1) from source s to every boundary vertex; 2) from some boundary vertices to some soft edge vertices; and 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.

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. 8. In Fig. 8(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. 8(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, i.e., if a boundary is within or has an L-intersection with a soft edge, 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. 7. The third situation is shown in Fig. 8(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. 5, if the timing slack of T^2 remains nonnegative when the soft edge e^2_{01} goes through boundary b_3, then an arc (a dashed line) between them is constructed in Fig. 7. We call such a construction a *soft edge expansion* and each expansion implies a timing slack consumption.

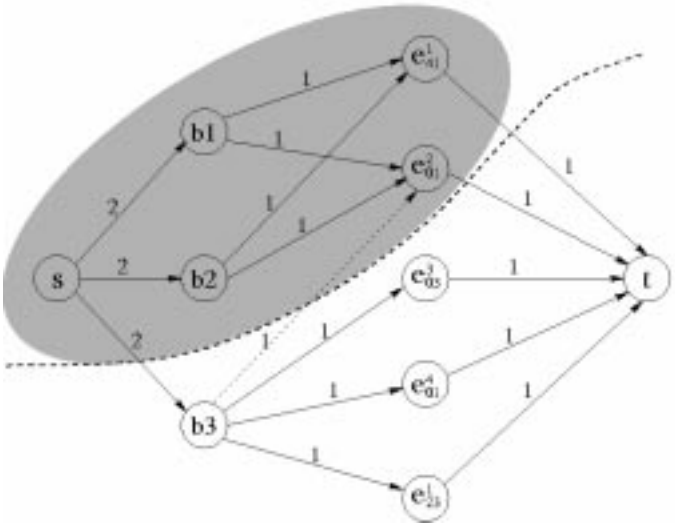

Fig. 7. Network formulation of the example in Fig. 5 without considering SSN. The number on each arc is its capacity.

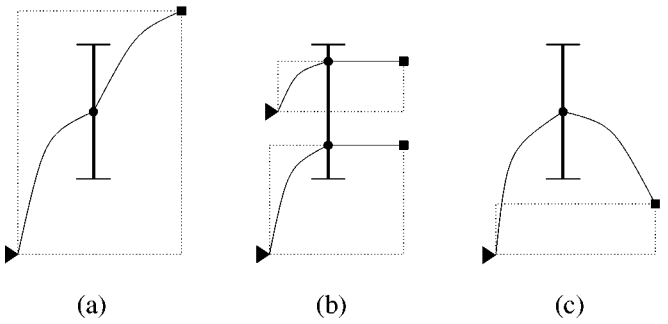

Fig. 8. Relative positions of a boundary and a soft edge.

We categorize the trees across the bisector line B into single-crossing trees and multicrossing trees, which are the trees that cross B only once (such as T^2 in Fig. 5) and more than once (such as T^1 in Fig. 5), 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 multicrossing 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 $S_{\text{old}}(T^i)$ before the assignment, and is $S_{\text{new}}(T^i)$ if its soft edge e^i_{jl} is assigned to boundary b_k, then we define the arc cost as

$$cost(b_k, e^i_{jl}) = (S_{\text{old}}(T^i) - S_{\text{new}}(T^i) + 1)^2. \quad (1)$$

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, otherwise the cost is larger than one. As a secondary objective, we hope to reduce the number of vias in the wiring. Therefore, for the situation in Fig. 8(b), we reduce its cost by a small user-specified offset θ, $0 < \theta < 1$. In our implementation, we let $\theta = 0.5$.

B. Construction of Arcs for Multicrossing 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 multicrossing trees, such as T^1 in Fig. 5. The difficulty here is that the timing slack computations 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 Fig. 5, the expansion of e^1_{41} depends on whether e^1_{23} 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. A uniform allocation may overlook local congestion distribution and result in some unnecessary expansions while some necessary expansion is not performed.

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 [31]. In the beginning, we run a max-flow algorithm on the partially constructed network to obtain an $s-t$ min-cut $(X, \overline{X})$, $s \in X$, $t \in \overline{X}$. The forward capacity of this cut is denoted by $U_{\min}(X, \overline{X})$. If $U_{\min}(X, \overline{X}) \geq |E_X|$, 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. 5, before the expansion for multicrossing trees, the min-cut is indicated in the dashed curve in Fig. 7, where the vertices in X are in the shaded region and vertices in $\overline{X}$ are unshaded. We can see that the forward capacity $U_{\min}(X, \overline{X}) = 4$ while there are five soft edges that need to be assigned, thus, we need to expand some soft edge(s) from the multicrossing 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 [31], e.g., (s, b_3), (e^1_{41}, t) and (e^2_{01}, t) are saturated. If a soft edge vertex e^i_{jl} is in X, e.g., e^1_{41} in Fig. 5, 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 $\overline{X}$ (and not all of its downstream arcs are saturated), e.g., b_3 in Fig. 5, 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 $e^i_{jl} \in \overline{X}$ matches a soft edge in a congested area to an uncongested boundary.

Lemma 2: The necessary and sufficient condition to increase the max-flow $f_{\max}$ of a network is to add a forward arc between X and $\overline{X}$ for every min-cut $(X, \overline{X})$ with $U_{\min}(X, \overline{X}) = f_{\max}$.

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 multicrossing tree T^i, from all the $b_k \in X$ and $e^i_{jl} \in \overline{X}$ 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 $U_{\min}(X, \overline{X}) \geq |E_X|$ 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. 7, we can make an expansion between $b_2 \in X$ and $e^1_{23} \in \overline{X}$ 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 multicrossing trees adaptive to the congestion distribution, and expansions are made only when necessary, without waste.

C. Utilization of 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. i.e., no timing slack is consumed. Again, we integrate this flexibility into the formulation of the network flow problem so that it can be exploited in a unified network flow solution.

The positions of an 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. 5. 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. 5, $e^3_{3'2}$ intersects with two boundaries b_2 and b_3, while $e^3_{3''2}$ 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. It is possible to include SSN locations in other grid cells, such as $v^3_{3''}$ in Fig. 5, into the generalized network flow model. However, their incident soft edges has less overlap on B which implies less timing-conserved flexibility and including them will increase the size of the network flow model. For v^3_3 and $v^3_{3'}$, all three associated soft edges $e^3_{3'1}$, $e^3_{3'2}$ and e^3_{03} are included in the vertices in the network as shown in Fig. 9. Obviously, e^3_{03} cannot be assigned simultaneously with $e^3_{3'1}$ or $e^3_{3'2}$. This exclusiveness constraint can be satisfied through adding a pseudo-vertex p and formulating a generalized network flow model [30], 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 min-cost flow problem using the Fleischer–Wayne algorithm [32], which is currently the fastest approximation algorithm for the generalized network flow model.

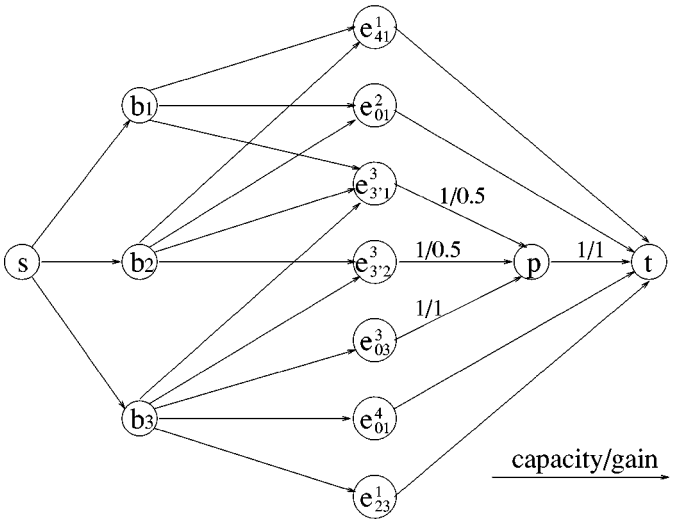

Fig. 9. Network formulation considering SSN.

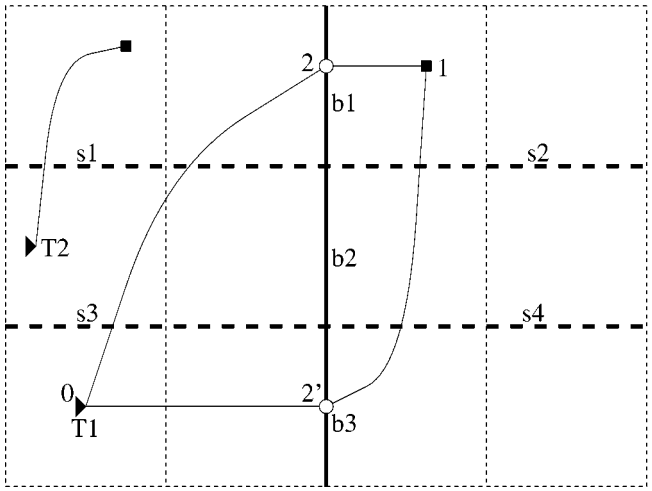

Fig. 10. Two options of routing T_1 may have different impacts on the congestions along the four thick dashed segments, even when both of them satisfy the congestion constraints along the vertical bisector line.

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. 6.

D. Post Processing

In our top-down hierarchical approach, the assignment at each hierarchical level is performed along one dimension (either horizontal or vertical). An assignment along one direction at one hierarchical level may be unfavorable to the congestion along the other direction at a lower hierarchical level. In order to alleviate this weakness, we apply simple post processing on each network flow solution. After performing the min-cost network algorithm, each soft edge is assigned to a grid cell boundary. Sometimes there are multiple assignment solutions (corresponding to degenerate solutions) that all satisfy congestion constraint at the same cost in terms of consumption on timing slack and the number of vias. The network flow algorithm can provide only one of the solutions, even though they may imply different impacts to congestion at the subsequent lower hierarchical level. For example, the assignment of T_1 across the vertical bisector line in Fig. 10 may affect the congestions along the four thick dashed segments. We define the density over a segment as the ratio of the number of intersecting wires to the total number of tracks along this segment. For example, if there are two wiring tracks across each grid cell boundary and we let the route of T_1 pass through b_1 in Fig. 10, then the density over segment s_1 will be 0.5. We define the cost over a segment in the same way as we define the boundary cost in maze routing in Section VI. The summation of the cost, over all segments that a route passes through, is employed as a secondary cost in the post processing, while the cost defined in the network flow formulation is treated as the primary cost. In Fig. 10, the secondary cost for assigning T_1 to b_3 is the summation of the cost over segments s_2 and s_4. In the post processing, we reassign each soft edge to another boundary when there is a reduction in the secondary cost and no degradation in either the primary cost or the congestion

Algorithm: Assignment

Input: Bisector line B
Soft edges E_X intersects B
Routing trees T^i that has soft edge in E_X
Output: Assignment of soft edges to boundaries in B

1. Set vertices and type-1, type-3 arcs in network
2. Set basic type-2 arcs
3. For each single-crossing tree
4. Do soft edge expansion
5. Min-cut $(X, \bar{X}) \leftarrow$ max-flow algorithm
6. While max-flow $< |E_X|$
7. For each multi-crossing tree T^j
8. $\forall$ pairs from $b_k \in B, \in X$ to $e \in T^j, \in \bar{X}$
Insert arc between min cost pair
9. Min-cut $(X, \bar{X}) \leftarrow$ max-flow algorithm
10. Run (generalized) min-cost max-flow algorithm
11. Truncate locus of SSN
12. Do post processing
13. Make assignment according to flow result

Fig. 11. Algorithm of assignment.

along the bisector line. The complete assignment algorithm is summarized in Fig. 11.

V. Computational Complexity of the Hierarchical Bisection and Assignment Algorithm

We will roughly analyze the complexity of the assignment algorithm at each hierarchical level and then give the complexity of the whole hierarchical bisection and assignment (HBA) algorithm.

The assignment algorithm consists of the dynamic network construction stage and the min-cost flow algorithm stage. The dynamic network construction is composed by several iterations of the max-flow algorithm whose complexity is dominated by the min-cost flow algorithm. The number of iterations of the max-flow algorithm is bounded by the number of cell boundaries along the cut line, because each soft edge is expanded to

cover at least one more boundary in each iteration. Thus, the complexity of the assignment algorithm is dominated by the complexity of the min-cost flow algorithm. If there is SSN involved, we will use the Fleischer–Wayne min-cost flow algorithm [32] for the generalized network. Otherwise, we will use the capacity scaling min-cost flow algorithm [30] for the conventional network, which is faster than the Fleischer–Wayne algorithm.

For a network with $|V|$ vertices and $|A|$ arcs, the capacity scaling algorithm has a complexity of $O((|A|\log U)\cdot(|A|+|V|\cdot\log|V|))$ [30], where U is the maximum arc capacity. If there are k cell boundaries along the bisector line and l soft edges across the bisector line, $|V|$ is bounded by $O(k+l)$ and $|A|$ is bounded by $O(k\cdot l)$. Thus, the capacity scaling algorithm has a complexity of $O((k\cdot l\cdot\log U)\cdot(k\cdot l+(k+l)\cdot\log(k+l)))$.

The Fleischer–Wayne algorithm is an ϵ-approximate algorithm[4] with complexity of $O(\epsilon^{-2}\cdot|A|\log|A|\cdot(|A|+|V|\cdot\log|A|)\cdot(\log\epsilon^{-1}+\log\log(UB/LB)))$ [32], where UB and LB are the upper bound and the lower bound of the total cost of a max-flow solution. If the cost upper bound for each arc is C, then $UB = C\cdot|A|$. Since each soft edge is assigned to at most one cell boundary, the cost lower bound LB approximately equals l. Thus, the complexity of the Fleischer–Wayne algorithm is $O(\epsilon^{-2}\cdot k\cdot l\log(kl)\cdot(kl+(k+l)\cdot\log(kl))\cdot(\log\epsilon^{-1}+\log\log(Ck)))$.

For a grid graph with m grid cells, the number of bisections is bounded by m and the number of cell boundaries k along each bisector line is bounded by $\sqrt{m}$, assuming that the number of rows roughly equals the number of columns. Usually the number of soft edges for a routing tree is bounded by a constant times the number of pins, hence the number of soft edges across a bisector line is bounded by n which is the total number of pins for a circuit. Then we can conclude the following.

Theorem 1: The HBA algorithm without using the SSN has a complexity of $O((m^{3/2}\cdot n\cdot\log U)\cdot(m^{1/2}\cdot n+(m^{1/2}+n)\cdot\log(m^{1/2}+n)))$ for a circuit with n pins on a grid graph with m grid cells and the maximum wire capacity across a cell boundary to be U.

Theorem 2: The HBA algorithm using the ϵ-approximate the Fleischer–Wayne algorithm has a complexity of $O(\epsilon^{-2}\cdot m^{1/2}\cdot n\log(mn)\cdot(m^{1/2}n+(m^{1/2}+n)\cdot\log(mn))\cdot(\log\epsilon^{-1}+\log\log(Cm)))$, for a circuit with n pins on a grid graph with m grid cells and the maximum arc cost in the network to be C.

Note that the above bounds are very loose bounds, since only on the topmost hierarchical level the values of k and l are close to $\sqrt{m}$ and n, respectively, and the actual values of k and l are usually much smaller on lower hierarchical levels.

VI. Timing-Constrained Rip-Up-and-Reroute

The last phase of our method is the TRR process. For each cell boundary with wiring overflow, we rip up every wire across this boundary and reroute it through maze routing. For each wire, we rip up the part corresponding to the soft edge. For example, in Fig. 1, we rip up edge (v_0, v_3) for tree T^3 across the thickened boundary. In the maze routing, we define the cost across a boundary b to be $D^2(b)$ if $D(b)<1$; otherwise $KD^2(b)$, where K is any large number greater than $D_{\max}$. Such quadratic cost can give a much heavier penalty to the congested path in a continuous manner. Similar cost definition is also employed in the work of [33] and a good discussion on the cost definition in maze routing can be found in [19]. We keep an arbitrary constant boundary ordering and repeat this process until there is no wire overflow or no improvement on congestion. On each boundary with wiring overflow, the rip-up-and-reroute also follows an arbitrary constant net ordering. Because of the iterative nature, the net ordering is not important. A net rerouted earlier in an iteration may have a result poorer than those rerouted later in the same iteration, since its rerouting is based on a poorer routings of other nets. Therefore, it should be rerouted earlier in the next iteration to make larger corrections. This explains why we use a constant net ordering [11].

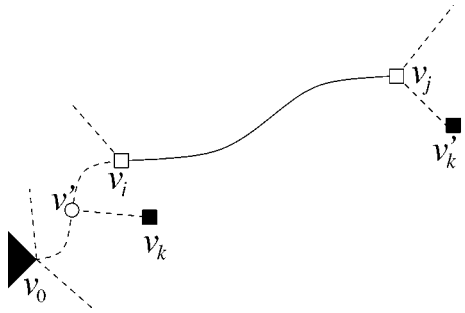

Fig. 12. Elongation for edge e_{ij}.

In the TRR method, we need to transform the delay constraints into physical constraints on edge length, i.e., we need to compute the maximum allowed elongation δ_{ij} for routing edge e_{ij} such that no delay violation is caused. We use C_j to represent the load capacitance seen from node v_j. The subtree rooted at v_i is denoted as T_i. For the interconnect wire, the resistance and capacitance per unit length is $\hat{r}$ and $\hat{c}$, respectively. The length of a routing path from driver v_0 to a node $v_i \in V$ is denoted as $p_{0,i}$, and the length of the common path for two nodes v_i, $v_j \in V$ from the driver is expressed as $p_{0,ij}$. For example, in Fig. 12, $p_{0,ik}$ is the path length from v_0 to v'. For any sink $v_k \in V$, we can compute the maximum δ_{ij} such that the delay slack $s(v_k)$ is nonnegative. If $v_k \notin T_i$, i.e., v_k is not downstream of v_i, as shown in Fig. 12, then

$$\delta_{ij} = \frac{s(v_k)}{(R_d + \hat{r}p_{0,ik})\,\hat{c}} \tag{2}$$

where R_d is the driver resistance, since the elongation of e_{ij} affects only the load capacitance seen from v'. If $v_k \in T_i$, such as v_k' in Fig. 12, δ_{ij} satisfies the following:

$$s(v_k) = (R_d + \hat{r}p_{0,i})\,\hat{c}\delta_{ij} + \tfrac{1}{2}\,\hat{r}\hat{c}(2l_{ij}\delta_{ij} + \delta_{ij}^2) + \hat{r}\delta_{ij}C_j \tag{3}$$

where l_{ij} is the original length of edge e_{ij}. This equation can be solved to obtain the δ_{ij}. In the case of double roots for this equation, we choose the one where the slope of function is negative, since the delay slack should be monotonically decreasing with respect to the allowed elongation. We compute δ_{ij} for all the sinks in the routing tree and choose the minimum value as a

[4]An ϵ-approximate algorithm can provide at least $(1-\epsilon)$ times the maximal flow with at most the optimal cost.

safe value. In the case where a delay without closed form expression is employed, the actual delay needs to queried each time a detour may occur in the maze routing.

The SSN can be exploited in TRR as well. For an SSN, we rip up the three incident soft edges all together and compute the minimal congestion paths to the locus of the SSN at each grid cell. For the example in Fig. 5, we rip up edges e^3_{03}, e^3_{31}, and e^3_{32}. Then, a maze routing search is performed from nodes v^3_3, $v^3_{3'}$, and $v^3_{3''}$ simultaneously toward nodes v^3_0, v^3_1 and v^3_2. Note that v^3_3, $v^3_{3'}$, and $v^3_{3''}$ are the three candidate locations for the SSN and moving any of them within its grid cell will not affect congestion. We choose one of the candidate locations based on the total cost of the three paths connected to it in the maze routing. For example, the cost of choosing $v^3_{3'}$ is the summation of the following paths found in maze routing: $(v^3_0, v^3_{3'})$, $(v^3_{3'}, v^3_1)$ and $(v^3_{3'}, v^3_2)$. We finally select a candidate location with the minimum total paths cost and route its incident soft edges according the paths found by the maze routing.

VII. Experimental Results

We implemented our algorithm in C++ and performed experiments on an Sun Ultra-10 workstation with 2 Gb of memory. The experiments are performed on ten benchmark circuits provided by the VLSI CAD Lab at UCLA. These circuit's characteristics are summarized in Table I. 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 results are listed in Table II. The initial routing trees are obtained through MVERT [26] algorithm. In the implementation of MVERT, we replace the SERT [20] algorithm in the initial routing by the AHHK [34] algorithm which can give similar routing tree performance at a faster speed. As a reasonable way of specifying timing constraints, after constructing the AHHK trees, we randomly assign a positive slack to each sink as a timing constraint. The subsequent non-Hanan optimization stage in MVERT will keep the routing tree to satisfy the timing constraints and minimize its wirelength at the same time. The second column in Table II gives the number of soft edges $|E|$ generated by the MVERT algorithm.

The congestion results are expressed in terms of total overflow F_{ov} and the maximum demand density $D_{\max}$. In order to see the impact of exploiting SSNs, two versions of our algorithm (HBA + TRR) are tested. The results without using SSNs are listed in columns 7–9 of Table II while the results exploiting SSNs are in columns 10–12. The SSNs are exploited through the Fleischer–Wayne algorithm [32], which is a relatively computationally expensive method. The value of ϵ in the Fleischer–Wayne algorithm determines the tradeoff of runtime and the solution quality of the generalized network flow problem. However, we found that the final routing quality is not sensitive to the value of ϵ in our experiment. Therefore, we empirically let $\epsilon = 0.2$, which is a relatively large value so that the Fleischer–Wayne algorithm can converge at a reasonable speed. Another implementation strategy is to enable the Fleischer–Wayne algorithm only at lower hierarchical level, i.e., we will not exploit SSNs at higher hierarchical levels. Since a decision at a higher hierarchical level may be unfavorable to subsequent lower levels, it is not worthwhile to invest computational resources on the expensive Fleischer–Wayne algorithm. On the other hand, exploiting SSNs at lower hierarchical levels will yield a more definite impact on final solution quality. Because a top-down approach is inherently in favor to higher hierarchical levels, it is reasonable to provide SSNs as additional leverage to lower hierarchical levels as a compensation. Moreover, it is more economical to apply the Fleischer–Wayne algorithm to lower hierarchical levels where the problem size is smaller. Based on our experience, we enable the Fleischer–Wayne algorithm only when $k \cdot l < 1000$, where k is the number of cell boundaries along a bisector line and l is the number of wires across the bisector line.

TABLE I
Benchmark Circuits

Circuit	# modules	# nets	# pins	grid
a9c3	147	1148	2674	45 × 42
ac3	27	200	609	48 × 46
ami33	33	112	480	50 × 46
ami49	49	368	861	45 × 45
apte	9	77	218	40 × 47
hc7	77	430	1748	49 × 56
hp	11	68	255	38 × 42
playout	62	1294	2957	37 × 32
xc5	59	975	3124	45 × 41
xerox	10	171	561	40 × 38

Comparing the results from with and without exploiting SSNs, we can see that SSNs help to improve the congestion quality in a few circuits ($a9c3$, $hc7$, and $xerox$). Several conditions need to be satisfied to let the SSNs taking effect: 1) The existence of SSNs depends on timing constraints and cannot be guaranteed. 2) The locus of an existing SSN should intersect the bisector line. 3) The SSN should be in a congested region so that sliding its location makes a difference. If it is not in a congested region, its location will not affect the congestion result. 4) Even in a congested region, the original location of an SSN must be at an inferior point that can be improved. It is common that one of these four conditions is not satisfied, so that enabling the use of SSNs does not make a difference on congestion results. The extra CPU time from exploiting SSNs is limited due to our careful application strategy on the Fleischer–Wayne algorithm.

The TRR method is a naive combination of timing constraints with RR in an effort to minimize the congestion subject to the timing constraints. Note that the SSNs are not exploited in the TRR here. The congestion results of TRR are in columns 5 and 6. We can observe that our approach always gives significant lower congestion in terms of both total overflow and the maximum demand density. Since the RR is good at congestion reduction only in a local region and lacks a global view, it is more likely to get stuck in a deadlock and fail to find a better solution under timing constraints. On the other hand, the hierarchical approach is better at a global planning level, and therefore, a combination of these two complementary approaches can yield a good result on congestion reductions subject to timing constraints.

TABLE II
EXPERIMENTAL RESULTS FOR RR (UNCONSTRAINED RR), TRR, AND HBA + TRR WITHOUT/WITH USING SSN. IN THE RESULTS OF RR, $F_{ov} = 0$, $D_{\max} = 1$ FOR ALL THE CIRCUITS EXCEPT THAT $F_{ov} = 1$, $D_{\max} = 1.13$ ON $xerox$

		RR		TRR		HBA+TRR(noSSN)			HBA+TRR(SSN)		
Circuit	$\|E\|$	#neg	slack(ps)	F_{ov}	D_{max}	F_{ov}	D_{max}	CPU(s)	F_{ov}	D_{max}	CPU(s)
a9c3	1727	191	-40345	11	1.12	10	1.12	216	4	1.06	261
ac3	499	105	-18448	25	1.29	2	1.14	12	2	1.14	16
ami33	393	32	-12130	5	1.60	1	1.20	8	1	1.20	11
ami49	570	156	-31771	24	1.22	0	1.00	20	0	1.00	26
apte	163	32	-9986	11	1.71	0	1.00	3	0	1.00	3
hc7	1623	155	-47274	11	1.33	5	1.22	36	3	1.22	63
hp	232	27	-6667	7	1.17	1	1.17	2	1	1.17	5
playout	1816	337	-31211	20	1.26	0	1.00	257	0	1.00	265
xc5	2390	174	-19813	32	1.14	0	1.00	151	0	1.00	177
xerox	437	61	-11391	6	1.25	4	1.13	9	3	1.13	14

TABLE III
RUNTIME IN SECONDS FOR THREE PHASES OF THE ALGORITHM AND THE MAXIMUM NUMBER OF ITERATIONS ON MAX-FLOW ALGORITHM IN EACH NETWORK CONSTRUCTION IN PHASE 2

	Phase 1	Phase 2		Phase 3
Circuit	CPU	CPU	#iter	CPU
a9c3	< 1	247	3	11
ac3	< 1	13	2	1
ami33	< 1	8	3	< 1
ami49	< 1	22	4	< 1
apte	< 1	3	0	< 1
hc7	< 1	56	5	2
hp	< 1	3	2	< 1
playout	< 1	261	1	3
xc5	1	172	2	3
xerox	< 1	9	4	1

For reference, we also performed the RR on the same circuits without imposing timing constraints during the congestion reduction process. The unconstrained approach is able to eliminate the wiring overflow for almost every circuit except $xerox$. Obviously, the congestions from RR are always better than the timing-constrained approaches. In order to see how much we may lose on timing performance if we ignore it in congestion reduction, we computed the number of nets with negative slack and the worst slack among all of the nets from the results of RR and listed them in columns 3 and 4 in Table II. We can see that every circuit has a very high negative slack and up to half of the nets could have timing violations for some circuits. Our proposed timing-constrained method results in no timing violations at all. The congestion results from our method are not sensitive to small changes on timing constraints. However, if we relax the timing constraints sufficiently, we can reach congestion results similar to those from RR, i.e., results with less congestions. Since our approach strictly obeys the timing constraints, the change on wiring capacities will not affect the timing performance.

The total runtime for three phases of our algorithm (with the exploitation of SSNs enabled) on each circuit are listed in the rightmost column in Table II. In Table III, we decompose the runtime for each phase. As we can see, Phase 2-HBA is the dominating part of the run time. In Phase 2, the adaptive network construction process includes several iterations of max-flow algorithm. In the column 4 of Table III, we listed the maximum number of iterations among all of the network constructions. 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 runtime on each two-pin net as a normalized comparison. It is conceivable that the formulation of soft edges is equivalent to a decomposition to two-pin nets. Based on this data, the average runtime is found to be between 0.02 and 0.15 s per two-pin net.[5]

VIII. CONCLUSION AND FUTURE WORK

In this work, we have proposed 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 an SSN and trade these flexibilities into congestion reduction while the timing constraints are satisfied. Experimental results show that the traditional RR method may cause significant delay violations and is poor on congestion when timing constraints are imposed directly. Our proposed algorithm can achieve good congestion results while satisfying timing constraints.

One limitation of our work is that only local timing-constrained routing flexibilities are employed compared with the global flexibilities used in [17]. A combination of the global and local flexibilities is expected to yield more timing-constrained congestion reduction. We assume that the timing constraints for each net is given and the consumption of the positive slack on each net will not cause timing violation along any path in the timing graph. Obviously this assumption depends on a good slack budgeting for each net along a timing path. If we can utilize a path-based slack directly, we will be able to avoid the slack budgeting and potentially obtain more routing flexibilities as in [17]. Therefore, including global flexibilities and path-based timing constraints into our current method will be a good direction of future research.

ACKNOWLEDGMENT

The authors are grateful to J. Cong, T. Kong, and D. Pan for providing the benchmark circuits and the floorplan. The authors would like to thank the anonymous reviewers for the insightful comments.

[5]The worst runtime is from $a9c3$, which has a 261 runtime on 1727 soft edges (1727 two-pin nets) and has an average runtime of 0.15 s/two-pin-net.

REFERENCES

[1] C. Chiang and M. Sarrafzadeh, "Global routing based on Steiner min–max trees," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 1318–1325, Dec. 1990.

[2] M. Burstein and R. Pelavin, "Hierarchical wire routing," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, pp. 223–234, Oct. 1983.

[3] M. Marek-Sadowska, "Global router for gate array," in *Proc. IEEE Int. Conf. Computer Design*, 1984, pp. 332–337.

[4] J. D. Cho and M. Sarrafzadeh, "Four-bend top-down global routing," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 793–802, Sept. 1998.

[5] P. Raghavan and C. D. Thompson, "Multiterminal global routing: A deterministic approximation scheme," *Algorithmica*, vol. 6, pp. 73–82, 1991.

[6] E. Shragowitz and S. Keel, "A global router based on a multicommodity flow model," *Integration: VLSI J.*, vol. 5, pp. 3–16, Mar. 1987.

[7] R. C. Carden, J. Li, and C.-K. Cheng, "A global router with a theoretical bound on the optimal solution," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 208–216, Feb. 1996.

[8] C. Albrecht, "Provably good global routing by a new approximation algorithm for multicommodity flow," in *Proc. ACM Int. Symp. Phys. Design*, 2000, pp. 19–25.

[9] J. Cong and B. Preas, "A new algorithm for standard cell global routing," *Integration: VLSI J.*, vol. 14, no. 1, pp. 49–65, 1992.

[10] B. S. Ting and B. N. Tien, "Routing techniques for gate array," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, pp. 301–312, Oct. 1983.

[11] R. Nair, "A simple yet effective technique for global wiring," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 165–172, Oct. 1987.

[12] K. W. Lee and C. Sechen, "A global router for sea-of-gate circuits," in *Proc. Euro. Design Automation Conf.*, 1991, pp. 242–247.

[13] Q. Yu, S. Badida, and N. Sherwani, "Algorithmic aspects of three dimensional mcm routing," in *Proc. ACMIIEEE Design Automation Conf.*, 1994, pp. 397–401.

[14] L. C. Abel, "On the ordering of connections for automatic wire routing," *IEEE Trans. Computers.*, vol. G-21, pp. 1227–1233, Nov. 1972.

[15] D. Wang and E. S. Kuh, "Performance-driven interconnect global routing," in *Proc. Great Lake Symp. VLSI*, 1996, pp. 132–136.

[16] J. Huang, X. L. Hong, C.-K. Cheng, and E. S. Kuh, "An efficient timing-driven global routing algorithm," in *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp. 596–600.

[17] X. Hong, T. Xue, J. Huang, C.-K. Cheng, and E. S. Kuh, "TIGER: An efficient timing-driven global router for gate array and standard cell layout design," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 1323–1331, Nov. 1997.

[18] J. Cong and P. H. Madden, "Performance driven global routing for standard cell design," in *Proc. ACM Int. Symp. Phys. Design*, 1997, pp. 73–80.

[19] ——, "Performance driven multi-layer general area routing for PCB/MCM designs," in *Proc. ACM/IEEE Design Automation Conf.*, 1998, pp. 356–361.

[20] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, "Near-optimal critical sink routing tree constructions," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 1417–1436, Dec. 1995.

[21] K. Zhu, Y.-W. Chang, and D. F. Wong, "Timing-driven routing for symmetrical-array-based FPGAs," in *Proc. IEEE Int. Conf. Computer Design*, 1998, pp. 628–633.

[22] U. P. Lauther, "Top down hierarchical global routing for channelless gate arrays based on linear assignment," in *Proc. IFIP Int. Conf. VLSI*, 1987, pp. 141–151.

[23] M. Marek-Sadowska, "Route planner for custom chip design," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1986, pp. 246–249.

[24] J. Hu and S. S. Sapatnekar, "Algorithms for non-Hanan-based optimization for VLSI interconnect under a higher order AWE model," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 446–458, Apr. 2000.

[25] ——, "FAR-DS: Full-plane AWE routing with driver sizing," in *Proc. ACM/IEEE Design Automation Conf.*, 1999, pp. 84–89.

[26] H. Hou, J. Hu, and S. S. Sapatnekar, "Non-Hanan routing," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 436–444, Apr. 1999.

[27] T. H. Chao, Y. C. Hsu, and J. M. Ho, "Zero skew clock net routing," in *Proc. ACM/IEEE Design Automation Conf.*, 1992, pp. 518–523.

[28] 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," in *Proc. ACM/IEEE Design Automation Conf.*, 1996, pp. 395–400.

[29] J. Cong and C. K. Koh, "Interconnect layout optimization under higher-order RLC model," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1997, pp. 713–720.

[30] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ: Prentice Hall, 1993.

[31] L. R. Ford, Jr. and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton Univ. Press, 1962.

[32] L. K. Fleischer and K. D. Wayne. (2001, Sept.) Fast and simple approximation schemes for generalized flow. *Math. Programming, DOI 10.1007/s101 070 100 238* [Online]. Available: http://link.springer.de/link/service/journals/10 107/first/tfirst.htm.

[33] H.-M. Chen, H. Zhou, F. Y. Yang, H. H. Yang, and N. Sherwani, "Integrated floorplanning and interconnect planning," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1999, pp. 354–357.

[34] C. J. Alpert, T. C. Hu, J. H. Huang, A. B. Kahng, and D. Karger, "Prim-dijkstra tradeoffs for improved performance-driven routing tree design," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 890–896, July 1995.

Jiang Hu received the B.S. degree in optical engineering from Zhejiang University, Hangzhou, China, in 1990, the M.S. degree in physics from the University of Minnesota, Duluth, in 1997, and the Ph.D. degree in electrical engineering from the University of Minnesota, Minneapolis, in 2001.

From 2001 to 2002, he has been with IBM Microelectronics Division working on VLSI CAD tools development. He will join the Department of Electrical Engineering at the Texas A&M University as an Assistant Professor in the fall of 2002. His current research interests include VLSI physical design automation, interconnect routing, optimization, and planning.

Dr. Hu received a Best Paper Award at the Design Automation Conference in 2001.

Sachin S. Sapatnekar received the B.Tech. degree from the Indian Institute of Technology, Bombay, in 1987, the M.S. degree from Syracuse University, Syracuse, NY, in 1989, and the Ph.D. degree from the University of Illinois at Urbana-Champaign, in 1992.

From 1992 to 1997, he was an Assistant Professor in the Department of Electrical and Computer Engineering, Iowa State University. He is currently an Associate Professor in the Department of Electrical and Computer Engineering at the University of Minnesota, Minneapolis. He has coauthored two books, *Timing Analysis and Optimization of Sequential Circuits*, and *Design Automation for Timing-Driven Layout Synthesis*, and is a co-editor of *Layout Optimizations in VLSI Designs*, all published by Kluwer.

Dr. Sapatnekar has been an Associate Editor for the IEEE TRANSACTIONS ON VLSI SYSTEMS and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: ANALOG AND DIGITAL SIGNAL PROCESSING, has served on the Technical Program Committee for various conferences, as Technical Program and General Chair for Tau and ISPD, and is currently a Distinguished Visitor for the IEEE Computer Society and a Distinguished Lecturer for the IEEE Circuits and Systems Society. He is a recipient of the NSF Career Award and best paper awards at DAC 1997, ICCD 1998, and DAC 2001.