Creating and Exploiting Flexibility in Rectilinear Steiner Trees

Elaheh Bozorgzadeh, Ryan Kastner, and Majid Sarrafzadeh

Abstract— **The global routing problem decomposes the large, complex routing problem into a set of more manageable subproblems. The high correlation between the output of the global router and the detailed router enables the designer to efficiently use the global route to refine the design quickly before running full detailed route. Hence,** *routability* **of the global routing solution is the key factor. The routability of the circuit depends on the congestion of the routing. In this paper, we study Steiner trees in terms of routability. We introduce the notion of** *flexibility* **– a geometric property associated with Steiner trees. We show that the flexibility of a Steiner tree is related to its routability. The main contribution of this paper is an algorithm which takes a stable Steiner tree as an input and maps it to a more flexible Steiner tree. Any existing Steiner tree algorithm can be used for the initial construction of the Steiner tree. Experiments with a global router on a subset of nets show that routing congestion is improved by approximately 20% locally throughout the region where those nets are routed.**

I. INTRODUCTION

WITH today's silicon technology, not only has the effect of interconnect delay on total path delay become significant, but also dominant. The router must have good controllability and predictability on congestion and timing issues, so that the design can converge on timing and interconnect without excessive routing iterations.

A standard cell router needs to handle a very large number of small standard cells. Current standard cell routers are mainly area-based gridded routers. These routers extensively employ rip-up and reroute (search and repair) algorithms to clean up design rule violations. The advent of DSM (Deep Sub-Micron) technology has greatly complicated the standard cell routing procedure.

Routing usually involves two steps: global routing and detailed routing. Global routing creates an approximate path for every interconnect, even though this path is not yet physically exact. Detailed routing then follows, determining the width and spacing needed for each different metal layer. The global router should consider a number of different metrics including – but not limited to – minimizing the delay of the each net, reducing the size of the chip, minimizing the number of vias and distributing the routes equally across the routing area [19]. The important problem facing global routing is to develop sub-solutions such that they can be solved by detailed router. A global routing solution with minimum delay and chip size accomplishes nothing if the detailed router can not find a solution. Hence, *routability* of the global routing solution is the key factor. Routability of the circuit depends on the congestion of the routing. Most global routers use congestion maps to help designers understand the results of global routing. If the design appears congested after the global route, designers alter the top level floorplan or placement of cells inside the block to get a better congestion before starting a detailed routing job.

Steiner tree construction is an essential part of global routing (the problem is formally defined later in Section II). The early set of Steiner tree algorithms focused on minimizing the total wire length as the objective (see [19] for an overview). As wire delay has become increasingly more important, Steiner tree algorithms have focused on minimizing the delay of the net. Recently, there has been much focus on algorithms which simultaneously consider buffer insertion, wire sizing and/or Steiner tree construction [11], [15], [18].

Typically, critical nets are a small subset of the overall number of nets. However, the number of critical nets are increasing with today's DSM technology. The delay of non-critical nets is not crucial to the performance on the circuit. Therefore, these nets should be routed in a manner which increases the likelihood that the detailed router can find a final, valid solution. Yet, we do not know of any previous Steiner tree algorithms which explicitly consider routability. Mostly routability is referred to as the density of routing edges in global routing. There exist many different global routing algorithms both in industry and research. In most of these techniques, the congestion is handled iteratively. There are other techniques which are not based on net ordering. In [5], a Steiner-tree based global router is presented. The proposed algorithm finds a weighted Steiner tree for a net while the weight of a region represents the demand of the region. The aim is to optimize the length and density simultaneously. In [6], a performance-driven Steiner tree algorithm is proposed. The objective is to minimize the delay function of Steiner tree. Global routing problem is formulated as multiterminal multicommodity flow problem. The objective is to maximize the overall routability by minimizing the edge congestion for a set of nets. In [3], an approximation algorithm for multi-commodity flow is used for global routing which can reduce the congestion significantly. In that paper, it is assumed that the algorithm can query a subroute to give a Steiner tree of a net with fixed length.

In this work, we study Steiner trees in terms of routability. We introduce the notion of *flexibility* – a geometric property associated with Steiner trees. Flexibility in routing edges has been exploited in many of the abovementioned existing research to improve the routing congestion. To the best of our knowledge, there has not been any work which tries to maximize the flexibility in routing trees. In this work, our focus is not to propose a new global routing algorithm or to develop a new algorithm to construct a new Steiner tree but to introduce and understand the property of flexibility for Steiner trees. With no deep understanding of flexibility, this property has been exploited (not fully) in several global routing techniques. For example, flexibility in Steiner trees has been exploited in the timing-driven global routing proposed in [10]. The global router tries to reduce routing congestion by taking advantage of different possible routes for edges called *soft edges*. We show that the flexibility of a Steiner tree is related to its routability. Specifically, a flexible tree is less prone to being routed through a congested region. We demonstrate this concept by constructing flexible trees in a state-of-the-art global router. Global routers (e.g. [10], [8]) can take advantage of flexibility in flexible routing trees to reduce the congestion locally.

The main contribution of this paper is an algorithm, which takes a Steiner tree as an input and produces a more flexible Steiner tree. The only restriction on the initial tree is that it must be stable (stability is defined in Section II-B). The new, flexible tree is guaranteed to have the same total length. Any existing Steiner tree algorithm can be used for the initial construction of the Steiner tree. As an example, our technique to map one Steiner tree to another tree with same length but more flexibility can be used in querying the subroutine to obtain new Steiner trees for the nets in the approximation algorithm described in [3]. Here it means that new Steiner tree might be found by changing the route of flexible edges for the net only. As a consequence, our algorithm can be applied or integrated into global routing algorithms to reduce the congestion locally. The preliminary version of this work has been published in DAC 2001 [7]. In [7], only the concept of flexibility

in global routing is proposed. The detailed analysis of creating and exploiting flexibility is presented here in this work.

The paper proceeds as follows: In Section II, we give some basic routing definitions, define the minimum Steiner tree problem and some associated properties of Steiner trees including flexibility. In Section III, the problem of generating flexibility in Steiner trees is formulated. Then our algorithm is described, which takes an existing Steiner tree as input and increases its flexibility. In Section IV, experimental results which study the impact of flexibility during global routing are presented. We conclude and give some possibilities for future work in Section V.

II. PRELIMINARIES

A. Global Routing Definitions

A *grid graph* is a graph G(V,E) such that each vertex corresponds to a point in a plane (an example is shown in Figure 1). A *net* $= \{(x_1, y_1,), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n)\}$ is an unordered set of points on a grid graph. A single point of the net is referred to as a *terminal*. A *routing* of a net is a set of grid edges such that the terminals are fully connected. Integer grid routing is assumed. The *route edges* of a net are the set of edges used in the routing of that net.

A *global bin* is a rectangular partition of the chip. By partitioning the chip into many rectangular regions and placing the cells into these regions, we have a placement using global bins. The boundaries of the global bins are *global bin edges*.
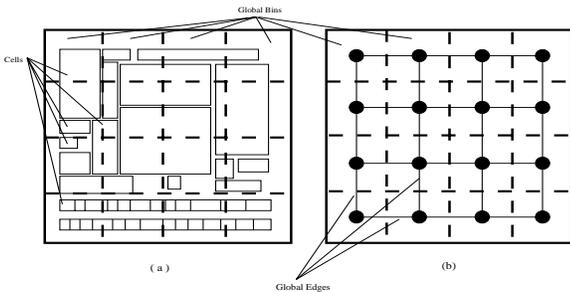

Fig. 1. **(a)** Placement of Cells into Global Bins. **(b)** The Corresponding Grid Graph.

In this paper, we assume that a global placement of cells and their interconnections are given by some placement engine. Each cell is assumed to be placed in the center of the global bin. The global bins and edges can be transformed into a grid graph (see Figure 1). The interconnections between the cells can be modeled by nets.

The *capacity* (also known as *supply*) of edge e is c_e. It is the maximum number of nets that can be routed over e. c_e is a fixed value that is based on the length of the edge and the technology used in creating the chip. The routing demand of e, specified as d_e, is defined as the number of route edges crossing edge e. Similarly, the demand of a vertex v is d_v. Here the demand corresponds to the number of routes that pass though the vertex v (equivalently the global bin v). An edge is overflown if and only if the $d_e > c_e$. Formally, the overflow of an edge is:

$$overflow_e = \begin{cases} d_e - c_e - t & \text{if } d_e > c_e \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

t is a threshold value which allows d_e to go above c_e without an overflow penalty. t is used since you can often route up to t nets though neighboring bins without affecting the congestion of those bins. t is usually a small constant (between 2-5). Using the global bin and global edge notation, the total overflow of a routing is:

$$overflow = \sum_{e=1}^{n} overflow_e, \quad (2)$$

where n is the number of bin edges. The total overflow reflects the shortage of routing resources for a particular set of edge capacities. The objectives of our global router is a linear function of the overflow of a route and the route length. Our industrial experience shows that total overflow is a good measure of congestion [8].

B. Rectilinear Steiner Tree

Given a net N (set of nodes called *terminals* of the net N) embedded in a grid graph $G(V, E)$, a *rectilinear Steiner tree* (RST) is a tree T interconnecting the terminals of N and some arbitrary points $P \in V$. The points P are known as the *Steiner points* (also called Steiner nodes) of T. The rectilinear Steiner tree problem is to find a rectilinear Steiner tree of minimum length (Steiner Minimal Tree). The length of a Steiner tree is the sum of the length of the edges of the tree. In rectilinear Steiner tree problem, the distance is defined in L_1 metric (rectilinear metric), in which the distance between two nodes is the sum of the difference of their x-coordinate and y-coordinates. The construction of a Rectilinear Steiner Minimal Tree (RSMT) has been given much attention and shown to be NP-Complete. [14].

Formally, a *segment* is a horizontal or vertical line segment connecting its two end points in the plane. Degree of each point is at most 4. Steiner points have the edge degree of at least 3. *corner-point* is a point with degree 2 with non-collinear segments that is not a terminal. Two points are *directly connected* if they are connected by a sequence of one or more segments including only corner-points. Such a route has staircase shape.

For any given rectilinear Steiner tree, there is a corresponding tree graph ($G(T)$), which is called *topology*. Vertices of $G(T)$ correspond to the terminals and Steiner points of RST T. Note that the corner-points are not vertices of the graph. Edge (i, j) connects vertices i and j iff the corresponding points are directly connected in T. As long as the edges between the vertices remain consistent, the topology is considered unchanged. In a given RST, location of Steiner nodes can change such that topology does not change (see Figure 3). Similarly in graph $G(T)$, the degree of vertices is bounded by 4. If the Steiner point p has edge degree of 4, it is called *cross-point*. *T-point* is a node with degree 3 in which one of incident edges is perpendicular to the other two incident edges.

In [2], Hwang *et al.* define two different transformations for rectilinear Steiner trees: *flipping* and *sliding*. Each transformation maps one such tree to another without moving the positions of the terminals and without increasing the length of the tree. Depending on the direction of transformation, there are 4 different slides and flip moves. In slide, a segment e perpendicular and incident to parallel segments is replaced by new segment which is still perpendicular and incident to the parallel segments. Slide transformation and Flip transformation are shown in Figure 3.
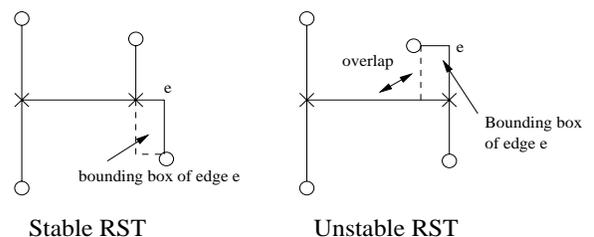

Stable RST   Unstable RST

Fig. 2. Examples of Stable and Unstable RSTs.

Ho *et al.* introduced the notion of *stability* under re-routing in an RST [16]. A RST is stable if there is no pair of edges such that their bounding boxes intersect or overlap except at a common endpoint (if any) of the two edges. Equivalently, a stable RST will not have overlaps when the edges are routed with minimum length. From this point forward, any reference to a RST assumes that the tree is stable. Figure 2 shows examples of stable and unstable RSTs.

C. Pattern Routing

Since we are focusing on routability, we define the cost of the Steiner tree as a linear function of the overflow and route length. We need to assign a routing (rectilinear route) to the edges which are neither vertical nor horizontal (e.g. edge e in Figure 2). We choose to *pattern route* such edges. Pattern routing is the notion of using pre-defined patterns to embed an edge on the grid graph. Usually these are simple patterns such as a L-shaped – 1-bend – or a Z-shaped pattern – 2-bends, route restricted within bounding box. In this paper, we restrict our pattern route definition to L-shaped or Z-shaped patterns.

Pattern routing has many benefits. First, it has a lower runtime complexity when compared to maze routing [13] – a commonly used routing algorithm. Second, a pattern routed net has small delay since it introduces a constant predetermined number of vias and routes the net with minimum wirelength. Finally, by properly choosing a subset of nets to pattern route, the global routing solution is more predictable without hindering the overall quality of the routing solution; this gives high level CAD tools (e.g. placement engine, floorplanner) higher accuracy when estimating routing metrics such as congestion, routing area and wirelength [8].

D. Flexibility

In this section, we define the property of flexibility. We argue that flexibility relates to the routability of the Steiner tree.

A *flexible edge* is an edge by which two nodes are directly connected with more than one segment. Therefore the flexible edge has one or more corner-points (staircase shape, L-shaped or Z-shaped). Flexible edges allow more than one minimum length route. If an edge e is routed with minimum rectilinear length, the route is contained within the bounding box of e. An inflexible edge has only one minimum length routing. Given a flexible edge with coordinates of its two endpoints as (x_1, y_1) and (x_2, y_2), X_e and Y_e are defined as follows:

$$X_e = \mid x_1 - x_2 \mid, \tag{3}$$

$$Y_e = \mid y_1 - y_2 \mid. \tag{4}$$

X and Y are called the *x-coordinate* and *y-coordinate* of edge e, respectively. The number of distinct two-bend routes (equivalently Z-shaped) with minimum length is $X + Y$ and the number of distinct one-bend (equivalently L-shaped) routes with minimum length is 2. In this paper, the flexible edges are shown as diagonal edges in the figures to show that there are more than one possible rectilinear routes for such an edge.

Looking at Figure 3, we show two rectilinear Steiner trees for the same net. One tree has some flexible edges while the other has only inflexible edges. Both trees have the same rectilinear length and topology. Assume that the shaded area is congested. The flexible tree is able to avoid the congested region while the inflexible tree has no choice but to be routed through the congested area if each edge is routed with minimum rectilinear length.

We need to define a function which evaluates the flexibility of a given edge. The function should reflect the routability of the edge.
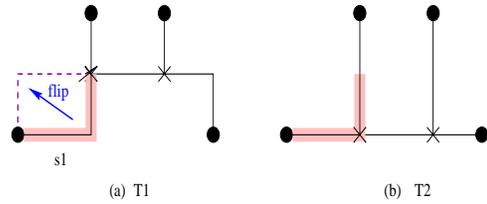

Fig. 3. **(a)**RST With Flexible Edges. **(b)** RST With No Flexible Edges.

Additionally, a function $g(e)$ corresponding to edge e should have the following properties:

- $g(e) = 0$ when $X_e = 0$ or $Y_e = 0$.
- $g(e)$ is a monotonically increasing function of X_e and Y_e.

Two suggested flexibility functions, g_1 and g_2, are shown in Equations 5 and 6.

$$g_1(e) = g_1(X_e, Y_e) = \begin{cases} 0 & \text{if } X_e = 0 \text{ or } Y_e = 0 \\ X_e + Y_e & \text{Otherwise} \end{cases} \tag{5}$$

$$g_2(e) = g_2(X_e, Y_e) = X_e \times Y_e \tag{6}$$

Function $g_1(e)$ corresponding to flexible edge e returns the value equal to half of the length of the bounding box of edge e. The value of function $g_2(e)$ corresponding to edge e is the area of the bounding box of edge e. The flexibility of a Steiner tree is computed by summing the flexibility over all the edges. Equation 7 defines the flexibility on a subtree T. A number of other functions can be introduced.

$$G(T) = \sum_{e \in T} g(e) = \sum_{e \in T} g(X_e, Y_e) \tag{7}$$

III. CREATING FLEXIBILITY IN RECTILINEAR STEINER TREES

In this section, we present an algorithm which maps a given rectilinear Steiner tree to another RST with maximum increase in the flexibility of the Steiner tree. First, we formally define the problem. Next, we describe our algorithm to maximize the flexibility of a given RST. Our algorithm solves the problem optimally subject to a set of constraints.

A. Problem Formulation

- Given a stable RST T,
- Map it to a stable RST T' such that $G(T') - G(T)$, the increase in flexibility of the RST T,is maximized,
- Subject to:
  - Topology and stability remain unchanged,
  - Each edge is routed with minimum rectilinear length,
  - No edge is degraded in flexibility [1].

Theorem 1: In a stable RST, if the Steiner point p has a flexible edge, the degree of p is 3. Only one flexible edge can be incident to a Steiner point.

Proof: More than one flexible edge cannot be incident to the Steiner point due to the stability of the RST.The examples of stable and non-stable Steiner points in a tree are shown in Figure 4. ∎

No Steiner cross-point can have a flexible edge. Steiner point with edge degree 3 is either a T-point (not incident to a flexible edge) or including a flexible edge. When a Steiner point is incident to a flexible edge, we refer to that point as a *flexible* point (see Figure 4).

Definition: The Steiner point p in a stable RST is *movable* if we can move p to any other location while keeping the RST stable and the
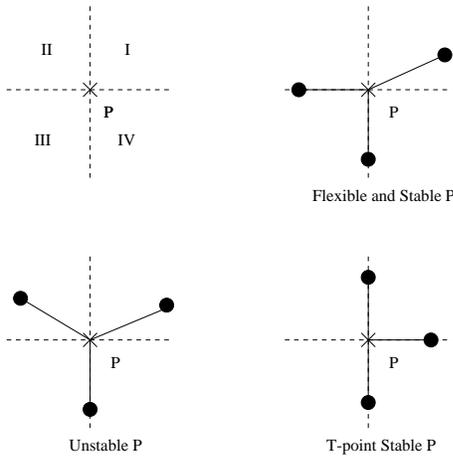
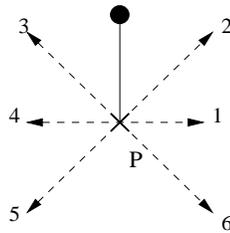Fig. 4.   Examples of Stable and Unstable Point p in a RST

Fig. 5.   Six Different Directions for Moving Steiner Point p.

topology unchanged. Similarly, an edge e is movable if e can be moved and the resulting RST remains stable with no change in topology.

Theorem 2: There is no single movable node in a stable RST. At least two adjacent nodes (two Steiner nodes) have to be moved.

Proof: In order to make segment s more flexible, p can move in six different directions (see Figure 5). Changing the connectivity between Steiner nodes and terminals is not allowed. To discover the requirements, the moving of node p along all possible directions is investigated. Figures 5, and 7 are referred during the proof. By moving p towards any of directions $2, 3, 5, 6$, the RST gets unstable unless the topology of the RST changes. In Figure 6, point p is moved towards direction 6. At most one edge would be located on the right side of p and the rest will be on the left side of p or along the vertical borders. Only one of the edges on the left side of p can be flexible and only one edge can be horizontal. In all different configurations of the edges the RST would be unstable.

If node p is moved towards direction 1, no edge in direction 4 should exist. Otherwise, by moving point p, RST gets unstable. Therefore, point p cannot be a T-point with the collinear incident edges perpendicular to segment s. By moving point p to point $p + \triangle l$ in direction 1,

---

[1] This assumption slightly simplifies the problem. The problem can be extended without it.
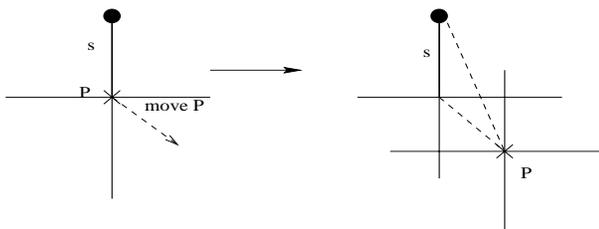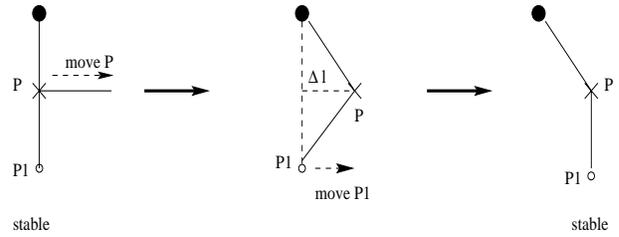

Fig. 6.   Unstable Move

Fig. 7.   Moving Steiner Point p Along Direction 1.

the tree is unstable unless point p_1 is movable in direction 1 as well. By moving p_1 in parallel with p in direction 1, tree would remain stable while the topology also remains unchanged (see Figure 7). This shows that two incident nodes (i.e. an edge) have to be moved. In other word, moving one node results in moving one of the nodes incident to that. Basically point p_1 needs to be a T-point or flexible point where in both cases the two collinear incident edges are along direction 1.

Moving towards direction 4 is similar to direction 1. ■

Note that an edge can also be moved only from one end point. When an edge is moved from both ends, the incident edges to both end nodes of the edge are moved from one end. We do not consider such edges as movable edge. We will see later such edges can be flexible candidates or parallel edges of movable sets.

Theorem 3: An edge s in a stable RST is movable along direction y if both vertices incident to the segment s

- *are Steiner points with edge degree 3.*
- *have incident edges towards the same direction, i.e. direction y. That is the direction of move for edge s.*
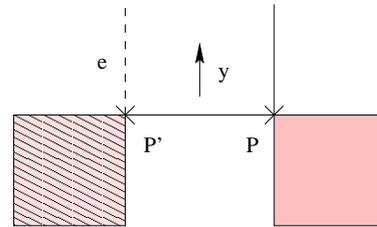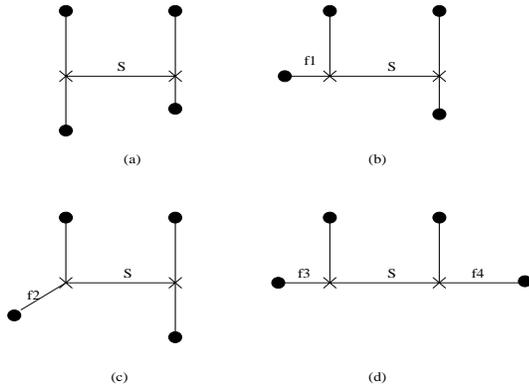

Fig. 8.   Movable Segment s.

Proof: Consider Figure 8. According to the proof of Theorem 2, an edge is movable in either horizontal or vertical direction while both ends are moved. When node p is moved towards direction y, there has to exist an edge incident to the node towards direction y. If there is no edge e incident to p' towards direction y, the bounding boxes of the edges incident to p' will overlap after edge pp' is moved toward direction y. That violates stability of RST. Therefore, there must be an edge incident to p' in direction y. Since RST is stable and edge degree of p' is three, the other edge incident to p' is in darkened region shown in Figure 8. Therefore, Steiner point p' has an incident edge towards direction y and the edge pp' is movable. ■

According to Theorem 3, the conditions under which an edge can be moved is similar to slide transformation in rectilinear Steiner tree. The difference is that in movable edge topology cannot change. Therefore by moving the movable edge in a given rectilinear RST, it is mapped to another RST with same topology. We can refer to the movable segment as *sliding* segment as well.

Corollary 1: On a movable edge slide transformation can be applied while topology does not change.

A subtree consisting of movable segment s and a set of edges adjacent to s , E_s , can be defined as a *Movable Set* (s, E_s).

$$(s, E_s) = \{P_s, M_s, F_s\},$$

Fig. 9. Movable Set With Movable Edge s.
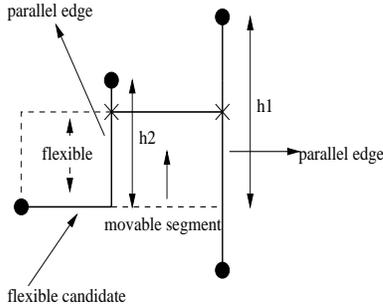
Fig. 10. Flexible Segment Generated by Algorithm *Generate_flexible_Tree*.

- $P_s = \{e_1 \in E_s, e_2 \in E_s; e_1 \neq e_2, e_1 \parallel e_2\}$,
- $M_s : \{s\}$,
- $F_s = \{e \in E_s - P_s \cup M_s \mid e$ is flexible candidate$\}$,
- $R_s = E_s - P_s \cup M_s \cup F_s$,

where E_s is the set of all adjacent edges to s including segment s. E_s consists of three different edge subset. P_s is the set of the two parallel edges according to Theorem 3. M_s is the movable segment, i.e. s. F_s is the set of flexible candidates among the adjacent edges to s. By moving edge s the flexibility of edges in F_s increases. R_s is the set of edges excluding the ones belonging to P_s, M_s, and F_s. For an example of a movable set, see Figure 10.

The movable set is a subtree with 4 leaves and two internal nodes. It is similar to definition of full component in Steiner tree. Full component is defined as a subtree with leaves of terminals [2]. However, in a movable set, the leaves need not to be the terminals.

Corollary 2: If the leaves of subtree of movable component are the terminals, the movable set is a full rectilinear Steiner component with 4 nodes.

B. Algorithm

Figure 9 shows 4 different movable sets (s, E_s). The movable sets have different subsets of edges in E_s. The edges adjacent to s are different in terms of flexibility. The movable set in Figure 9 (a) has no flexible candidate i.e. $F_s = \emptyset$. The other movable sets have at least one flexible candidate. The flexible subsets(F_s) are $\{f_1\}$ in Figure 9 (b), $\{f_2\}$ in Figure 9 (c) and $\{f_3, f_4\}$ in Figure 9 (d). Edge f_2 is already a flexible edge. However by moving segment s the flexibility of f_2 can increase. Edges f_1, f_3, and f_4 become flexible after segment s is moved. In Figure 9 (a) moving segment s produces no additional flexibility. In the other cases, the flexibility of the RST increases by moving segment s. This implies that at least one flexible candidate has to exist in a movable set in order to increase the flexibility of the RST by moving the movable edge.

Theorem 4: In a stable RST, by moving a movable edge, flexible edge(s) can be generated or flexibility of an edge(s) can increase if at least one of the two vertices incident to the movable edge is a flexible node with an incident edge towards the moving direction.

Proof: If both endpoints of the moving edge in a movable set are T-points, no flexible edge can be generated by moving the movable edge. ∎

In order to increase the flexibility of a RST, we first need to search for movable sets which have the potential to generate flexible edges. Movable sets in a RST can be found in $O(E)$, where E is the the edge set of the Steiner tree. According to Theorem 3, each edge is checked whether it is movable with incident flexible candidate.

When a movable set is found, the movable segment can move along the parallel edges. The maximum movement is the minimum length of the two parallel edges incident to the movable segment. In other words, this is the maximum range for sliding transformation on movable segment without changing the topology of RST. In Figure 10, the maximum length of sliding is $\ell_{max} = min(h_1, h_2)$ where h_1 and h_2 are the length of the parallel edges. Since topology should not change, the segment can move as far as one grid before it reaches the end node of any of the two parallel edges. According to Theorem 3, by moving a movable edge, the flexibility of the RST improves without changing the wirelength and topology. However, by maximizing the flexibility of a movable set, we may restrict the flexibility of another movable set. We discuss this situation, called *overlap*, next.

C. Overlap in Movable Sets

Once every individual movable set is found in a given RST, the question is: Are the movable segments independent and by moving all of them, can maximum flexibility be obtained? In order to answer the question, we need to see how movable segments can *overlap*.

When there is no overlap, we easily compute the flexibility function for each moving segment. Since the flexibility function is monotonically increasing in terms of x-coordinate and y-coordinate of the edges, the maximum of G occurs if each moving edge is moved to its maximum limit. Unfortunately, the flexibility cannot be computed independently for two overlapping segments.

Two movable segments have overlap with each other in either of the two following cases:

1) The movement of movable edge s is limited by the movement of the other movable edge (Figure 11).
2) Moving the movable edge s increases flexibility of an edge e while the movement of some other movable segment t leads to a decrease in the flexibility of e (Figure 12).

Two movable sets *intersect* if their corresponding edge sets share a common edge. There are six possible intersections between any two movable sets. Only three of them cause overlap between the segments. Let (s_1, E_{s_1}) and (s_2, E_{s_2}) be two movable segments and $V(s_1)$ and $V(s_2)$ be the pair of incident nodes for segment s_1 and s_2 respectively. The six different intersections between the two sets are as follow:

1) $\exists e \in E_{s_1} \bigcap E_{s_2}, e \in P_{s_1}, e \in P_{s_2}$.

Segment s_1 and segment s_2 have a same parallel edge.

   a) $V(s_1) \bigcap V(s_2) \neq \emptyset$.

   The moving direction of both must be the same. Otherwise the degree of v cannot be 3. This is overlap type 1. This case is shown in Figure 11 (a). In order to have the same topology and maximize the flexibility, the amount of movement for each segment would be:
   $$\ell(s_1) = \ell(s_2) = min(\ell(s_1), \ell(s_2)).$$

   b) $V(s_1) \bigcap V(s_2) = \emptyset$.

   $V(s)$ is the set of nodes incident to s. If the moving direction of both is the same, no overlap would occur (Figure 11
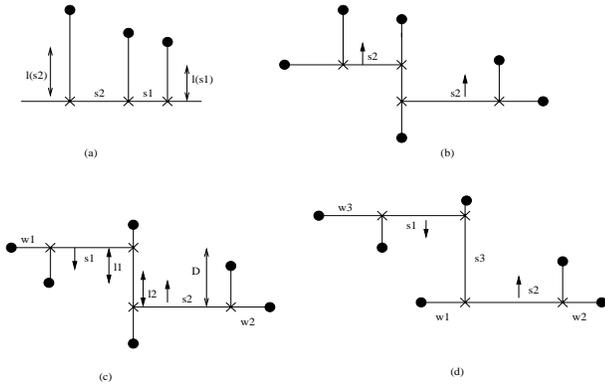
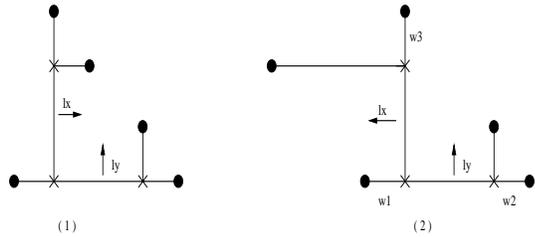Fig. 11.   Intersection Between Two Movable Sets With Shared Parallel Edge.

Fig. 12.   Intersection When a Parallel Edge Being Another Moving Edge.

Fig. 13.   Two Movable Segment Sharing a Flexible Candidate.
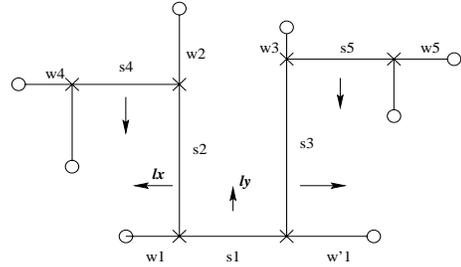
Fig. 14.   Maximum Overlap of a Movable Set with Other Movable Sets.

(b)). When they move in opposite directions, overlap can occur. The two cases in which overlap occur are shown in Figure 11 (c) and (d).

In Figure 11 (d), the segment s_3 overlaps with s_1 as well. This case is complicated will be discussed later. This is a combination of overlap types 1 and 2.

In the case shown in Figure 11 (c), overlap can occur if the condition below holds.

$$\ell_1 + \ell_2 \leq D$$

This is overlap type 2.

2) $s_1 \in F_{s_1} \lor s_2 \in F_{s_1}$

The moving edge s_1 is the flexible edge of segment s_2. This is same as intersection $1(a)$.

3) $s_1 \in P_{s_2} \lor s_2 \in P_{s_1}$

The movable edge of a set is a parallel edge of another set. The two cases are shown in Figure 12. The first case does not overlap, but the second one is similar to the overlap shown in Figure 11 (d).

4) $s_1 = s_2 \land E_{s_1} \neq E_{s_2}$

Such a case will not happen since the RST is stable.

5) $e \in P_{s_1} \land e \in F_{s_2}$

The parallel edge of s_1 is a flexible edge in set s_2. The configuration is same as the one shown in Figure 12 (2).

6) $e \in F_{s_1} \land e \in F_{s_2}$

This case is shown in Figure 13. Figure 13(a) shows when two sets have the same flexible edge in the same moving direction. In this case there is another movable set between those, which has overlap case 1 with both of them. This is a chain of case 1 with overlap type 1. There is no overlap in the case shown in Figure 13(b).

According to the study on all possible intersection between the two movable sets, there are two types of overlap that can occur:

- **Overlap Type 1**: It occurs when a parallel edge of two movable sets is the same and inequalities mentioned in intersection 1 hold.
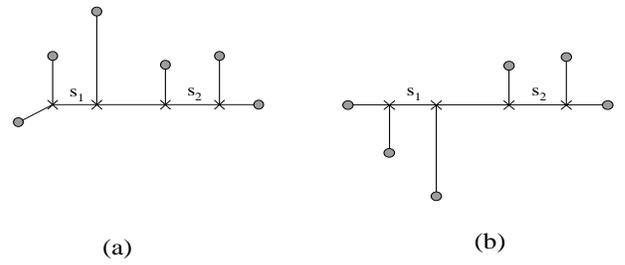- **Overlap Type 2**: It occurs when flexible edge of a movable segment is a parallel edge of the other moving set.

Theorem 5: Each movable segment set can have at most two overlaps of type 1 and two overlaps of type 2.

Proof: Since each movable set has two parallel edges, each can have overlap (type 1) with another movable set. A movable set has at most two flexible edges. Therefore, at most two overlaps of type 2 can occur by flexible edges being the parallel edges of the two other movable sets. Overlaps more than 4 leads to edge degree 4 for the steiner nodes at two ends of movable edges which contradicts with the properties of movable set.   ∎

In Figure 14, movable set (s_1, E_{s_1}) overlaps with segments (s_2, E_{s_2}), (s_3, E_{s_3}), (s_4, E_{s_4}) and (s_5, E_{s_5}). Two of the overlaps are overlap type 1 and the other two overlaps are of type 2.

When movable sets are being found in a RST, overlaps among the movable set can be detected as well. When an edge is being re-visited it means that the edge has already been recognized as a member of another movable set which has already been found. In such a case the algorithm checks whether such an intersection generates overlap. Since each edge can participate at most in 2 movable sets according to Theorem 5, the complexity of the algorithm to find the movable sets and their overlap still remains $O(E)$. The first loop of the pseudo-code shown in Figure 16 finds the movable sets and the overlaps among the sets.

In both types of overlap, we need to measure the flexibility in order to decide on moving the movable segments. The behavior of the overlap will depend on the flexibility function. If the movable sets overlap with each other, the maximum of flexibility function can not be obtained by maximizing the flexibility function for each movable set. Therefore dealing with overlaps depends on the definition of flexibility function. We show how to resolve the overlap if flexibility functions introduced in Section II-D are used.

Consider moving sets shown in Figure 15. In both figures, segment s_1 overlaps with segment s_2. We will study how the decision on moving the overlapping edges is made by using each flexibility function mentioned in Section II-D. Note that in the following discussion variables x_i or y_i is used to define the amount of sliding for the movable segment s_i in movable set (s_i, E_{s_i}). They are not related to coordi-
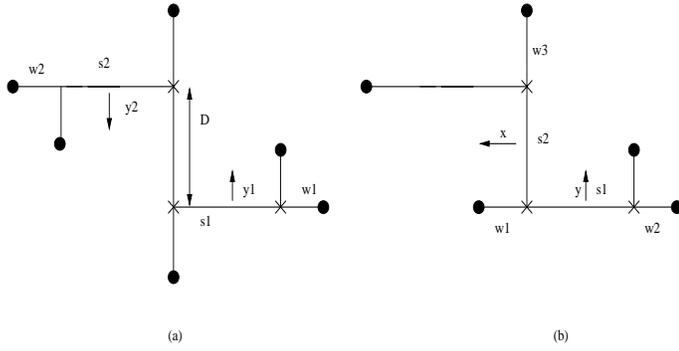
Fig. 15. Example of Overlap Type 1 (a) and Overlap Type 2 (b) for Movable Sets (s_1, E_{s_1}) and (s_2, E_{s_2}).

nates of the points on the plane but variables of the flexibility functions when movable segment is moved. X_i and Y_i correspond to the x-coordinate and y-coordinate of edge e_i before any segment is moved.

$$\triangleright g_1(e) = g_1(X_e, Y_e) = \begin{cases} 0 & \text{if } X_e = 0 \text{ or } Y_e = 0 \\ X_e + Y_e & \text{Otherwise} \end{cases}$$

This function is a discontinuous function at $X_e = 0$ or $Y_e = 0$. If overlap 1 occurs like the one shown in Figure 15(a), the flexibility function of the subtree $T_{1,2}$, a union subtree of overlapping movable set 1 and movable set 2, will be :

$$G(T_{1,2}) = G(y_1, y_2) = \begin{cases} X_2 + y_2 & y_1 = 0 \wedge y_2 \neq 0 \\ X_1 + y_1 & y_2 = 0 \wedge y_1 \neq 0 \\ X_2 + y_2 + X_1 + y_1 & (y_1 \neq 0 \wedge y_2 \neq 0 \\ & \wedge y_1 + y_2 \leq D) \end{cases} \quad (8)$$

Maximum of G occurs when $y_1 \neq 0$ and $y_2 \neq 0$ and $y_1 + y_2 = D$.

If overlap type 2 occurs as shown in Figure 15(b), the flexibility of subtree $T_{1,2}$ is formulated as:

$$G(T_{1,2}) = G(x, y) = \begin{cases} X_1 + X_2 + 2y & \text{if } x = 0 \wedge y \neq 0 \\ X_3 + x & \text{if } y = 0 \wedge x \neq 0 \\ X_1 + X_2 + X_3 + 2y & \text{if } y \neq 0 \wedge x \neq 0 \end{cases} \quad (9)$$

Maximum of $G(T_{1,2})$ occurs when $x \neq 0$ and $y = l_{max}$ ($x \leq w_1$). The value of x is not fixed , but it has to be nonzero. The value of x can be chosen such that the segment passes through less congested area.

When g_1 is used as the flexibility function, in the case of overlap type 1 or 2 in the subtrees of RST, the decision on moving the movable edges can be found by solving a linear equation. If there are more than one overlaps between the movable sets in subtree T, the function $G(T)$ is formulated in terms of variables associated with each movable set.The values of the variables are assigned such that $G(T)$ is maximized. In both types of overlap, $G(T)$ is maximized when the variables are non-zero. Therefore, when resolving the overlap among a set of movable sets, we use the case with non-zero variables in equation $G(T)$. Therefore G(T) can be defined as a equation of set of independent variables which can be maximized in linear time in terms of number of variables in the equation.

$\triangleright g_2(e) = X_e \times Y_e$

This function is a nonlinear continuous monotonically increasing function in terms of X_e and Y_e. If overlap type 1 occurs (Figure 15(a)), y_1 can be represented as $y_1 = d - y_2$. Then the function G will be linear in terms of y_2 and the problem can be linearly solved. The linear solution would lead to acceptance of just one of the two moving sets.

In the case of overlap type 2 as shown in Figure 15(b), the flexibility function of two overlapping moving segments will be:

Fig. 16. Algorithm *Generate_flexible_tree* Pseudo-code.

$$G(T_{1,2}) = G(x, y) = (X_1 + X_2) \cdot y + X_3 \cdot x - x \cdot y \quad (10)$$

The Hessian matrix ($\nabla^2 G$) is indefinite. The function creates a saddle point. The maximum occurs on the limits of x and y. So four points need to be checked to find the greatest value for G.

Note that for the case when a segment has overlaps from both types on one parallel edge as shown in Figure 11.d, the variable associated with movable set s_2 can be defined in terms of the variable associated with moving segment s_1. Therefore the problem will be reduced to finding a maximum of a function of two variables with overlap type 2.

In Figure 16, algorithm *Generate_flexible_tree* is presented. Basically, the first part finds movable sets (*MovableList*) and the overlaps between these sets (*overlapList*). Using appropriate data structures, the first "for" loop has O($|E|$) time. The second "for" loop moves the independent movable sets individually. *solve_overlap_and_move()* resolves the conflict between overlapping sets while maximizing the flexibility of the RST. The complexity of the second part depends on the complexity of the flexibility function.

If there is a chain of configurations as shown in Figure 14 and g_2 is used as a flexibility function, the expression of flexibility function will include all the quadratic terms of overlapping sets. Finding the maximum of G is exponential in terms of number of movable sets in the overlapping chain. We expect these chains to be short in most applications.

Theorem 6: The algorithm Generate_flexible_tree solves the problem of maximizing flexibility in a given RST T under the defined flexibility function g and constraints as formulated in Section III-A. If flexibility function g is a linear function, the algorithm runs in polynomial time. If g is quadratic function the algorithm complexity can be pseudo-polynomial in terms of number of movable set in overlapping chain.

Proof: After all the movable set are discovered ($O(E)$), those moving edges which do not overlap with any other movable sets (independent sets) are moved as far as possible. For those which overlap, function *solve_overlap_and_move()* finds the maximum value of flexibility function for the overlapping segments and determines the limit of move for each of the two segments. According to aforementioned details about resolving the overlap, the maximum value of $G(T_1)$ provides the most flexible configuration for subtree T_1. Therefore the algorithm is able to find the most flexible RST. The time complexity of the algorithm depends on the flexibility function, g, If g is linear (like

g_1), the overlaps are solved in polynomial time as described above. In the case where flexibility function is $g = X \cdot Y$ and chain of k moving edges overlap (type 2), the maximum flexibility is found by checking all the maximum limit of overlapping movable edges. Therefore the complexity can grow up pseudo-polynomially in terms of k where k is the number of movable sets in overlapping chain. ∎

It is important to mention that the RST resulted from algorithm *Generate_flexible_tree* has the same wirelength as the original RST. There is no degradation in the length of the given RST while flexibility is maximized. We have shown that movable sets are the only feasible moves for Steiner nodes so that the resulting RST is still stable with the same topology. In addition, since each edge is routed in minimum rectilinear length, total wirelength will remain the same. In our algorithm, the rectilinear length of flexible edges are increased as much as the rectilinear length of parallel edges is decreased. Therefore, the total wirelength remains the same.

IV. EXPERIMENTS

In this section, we perform experiments which attempt to show the relationship between flexibility and routability. First, we present the characteristics of the benchmarks. Then, we present experiments to show that flexibility is related to routability.

A. Benchmarks

To perform our experiments, we used MCNC standard-cell benchmark circuits [17] and benchmarks from the ISPD98 suite [12]. The characteristics of the circuits are shown in Table I. The circuits were placed into using the Dragon global and detailed placement engine [20] which is comparable in quality to the commercial version of Timberwolf [21]. The ISPD98 benchmarks are slightly modified [20] so that they could be placed.

The MCNC benchmarks are relatively small compared to today's designs and are questionable to use with the fixed-die placement engines [9]. Yet, they are the only publicly available placement benchmarks and are widely used in literature. The ISPD98 benchmarks are larger and comparable in size to current ASIC designs.

Data File	Num Cells	Num Nets	Num Pins	Global Bins
avqlarge	25278	33298	90665	256×64
avqsmall	21584	30038	84081	30×80
avqsmall.2	21584	30038	84081	80×80
biomed	6417	7052	22253	20×40
biomed.2	6417	7052	22253	40×40
ibm01.1	12036	13056	45815	64×64
ibm01.2	12036	13056	45815	256×64
ibm05.1	28146	29647	127509	64×64
ibm05.2	28146	29647	127509	256×64
ibm10.1	68685	75940	298311	64×64
ibm10.2	68685	75940	298311	256×64
primary1	833	1156	3303	8×16
primary2	3014	3671	12014	16×16

TABLE I
BENCHMARK CIRCUIT INFORMATION

B. Experimental Results

Our experiments focus on determining the effect of flexibility using global routing. We performed our experiments using Labyrinth [1], a global router based on maze routing. Labyrinth sequentially maze routes every net. Then, a rip-up and reroute phase is applied to deal with the net ordering problem.

Our experiments focused on 4-terminal nets. We choose the 4-terminal nets which have only one movable edge. We refer to this set of nets as *considered nets*. Then, we produced two Steiner trees for each net, an inflexible tree we call *inflex* and a flexible tree named *flex*. The flexibility of the *inflex* is guaranteed to be less than the flexibility of the *flex*. The experimental flow is as follows: first all the nets other than considered nets are routed using maze routing. After congestion is generated on the gridded graph, the considered nets are routed. In one setup of experiments, they are routed with *inflex* tree pattern. In another set of experiment, they are routed with *flex* tree pattern.

Before we discuss the results, we summarized the data categories of presented in Table II.

- **Number of Nets considered**: the number of nets which have 4 terminals and have one movable edge
- **Total Route Length**: the length of the routed Steiner tree (net) summed over all the considered nets. Remember that the route length for the flexible and inflexible tree is equivalent.
- **Total Overflow**: The overflow for one net is $\sum_{e \in Route}[demand_e - capapity_e]$. The total overflow is the summed overflow of the considered nets.
- **Total Demand**: Demand of a net is $\sum_{e \in Route}$ [number of nets routed through e]. Total demand is the summed demand of the considered nets.
- **Overflow Improvement**: $\frac{total\ overflow - flex\ overflow}{inflex\ overflow}\%$. A positive percentage indicates that the overflow of the flexible tree is better than the overflow of the inflexible tree.
- **Demand Improvement**: $\frac{inflex\ demand - flex\ demand}{inflex\ demand}\%$.

In our experiment, we compared the overflow and routing demand of a flexible and inflexible tree. The edges of the Steiner tree (both the flexible and inflexible) were routed in an Z-shaped pattern. We compared two properties of the routing, the overflow and the demand; both are related to the congestion of the circuit (both are defined in Section II-A). The overflow of the route is the demand minus the capacity summed over all the route edges. Intuitively, this determines the amount of routing resources which are needed, but can not be supplied. The demand simply is amount of other routings which are competing for the same routing resources (edges).

Referring to Table II, we can see that the flexible tree produces a routing which has, on average, 20.17% less overflow than the inflexible tree (we want to minimize overflow). In fact, the flexible tree was better in every benchmark except one (biomed). Additionally, the flexible tree encouraged a routing which passed through less demanded regions. The average demand was 4.49% better for the flexible tree as compared to the inflexible tree.

In another set of experiment, we used a L-shaped pattern-route for the Steiner edges. The results had a similar trend as the z-shaped pattern results. In this case, the flexible tree has an overflow which is 18.95% better than the overflow of the inflexible tree. The demand for the routing of the flexible tree is 4.34% less (better) than the demand for the inflexible tree.

In both of the experiments, the flexible tree produced a routing which has less overflow. Furthermore, the flexible tree routing tend to pass though edges with less demand.

V. CONCLUSION AND FUTURE WORK

This paper studied the problem of routability in global routing from a different perspective. Flexibility – a new geometrical property of a RST – was defined and discussed. We argued that flexibility has a high correlation to the routability of the Steiner tree. An algorithm was

Circuit	Number of Nets	Length of Route	Total Overflow		Total Demand		Overflow Improvement	Demand Improvement
			Flex	In-flex	Flex	In-flex		
avql	18	151	5	19	392	462	73.69%	15.16%
avqs	4	26	24	26	241	255	7.69%	5.49%
avqs.2	14	127	181	235	684	739	22.98%	7.44%
biomed	5	170	785	810	2202	2265	3.09%	2.78%
biomed.2	8	318	122	157	3055	3104	22.29%	1.58%
ibm01.1	24	259	377	466	2823	2966	19.10%	4.82%
ibm01.2	66	1519	1596	1990	15116	15858	19.80%	4.68%
ibm05.1	4	94	336	364	3750	3881	7.69%	3.37%
ibmo5.2	16	485	807	853	12646	12867	5.40%	1.72%
ibm10.1	99	2437	16646	18890	88183	90882	11.88%	2.97%
ibm10.2	221	10707	14533	19021	267668	282095	23.60%	5.11%
Primary1	4	45	31	42	227	237	26.19%	4.22%
Primary2	21	652	493	612	5587	5881	19.44%	5.00%
Total	504	16990	35936	43485	402574	421492	-	-
Average	-	-	-	-	-	-	20.17%	4.49%

TABLE II

OVERFLOW AND CONGESTION RESULTS WITH Z-SHAPED PATTERN ROUTES FOR FLEXIBLE EDGES.

proposed to generate an optimally flexible RST given a stable RST. Our initial experimental results supports our idea. Our results showed that a more flexible Steiner tree produces a route which has smaller overflow and has less demand. Since the routability of a circuit is related to the overflow of its route edges, a flexible RST produces a global routing solution with lesser congestion.

We believe that the proposed algorithm can be used in early stages to assign the location of Steiner points while considering routability. By using this method, the global routing can deal with routing congestion more easily. As future work we plan to study the flexibility functions in more detail and integrate the flexibility algorithm into our existing global router.

VI. ACKNOWLEDGMENT

Authors wish to thank Dr. Dirk Stroobandt for his valuable feedback regarding various aspect of this work. Authors are also grateful to reviewers for their detailed comments technically and conceptually.

REFERENCES

[1] R. Kastner, and M. Sarrafzadeh. "Labyrinth: a Global Router and Development Tool ". URL: *http://www.cs.ucla.edu/ kastner/labyrinth/*. 1999.

[2] F. K. Hwang, D. S. Richards, and P. Winter. "The Steiner Tree Problem". *Annals of Discrete Mathematics 53, North-Holland*, pp. 203-282, 1992.

[3] C. Albrecht.ACM/SIGDA International Symposium on Physical Design "Provably good global routing by a new approximation algorithm for multicommodity flow". In *Proc. ACM/SIGDA International Symposium on Physical Design*, April 2000.

[4] R. C. Carden IV, J. M. Li, and C. k. Cheng. "A global router with a theoretical bound on the optimal solution". *IEEE Transaction on Computer-Aided Design*, 15(2), pp.208, 1996.

[5] C. Chiang, C. K. Wong, and M. Sarrafzadeh. "A Weighted Steiner Trees-Based Global Router with Simultaneous Length and Density Minimization". In *IEEE Transactions on Computer-Aided Design*, 13(12), pp. 1461-1469, December 1994.

[6] X. Hong, T. Xue, E. S. Kuh, C. K. Cheng, J. Huang. " Performance-driven Steiner tree algorithm for global routing ". In *Proc. ACM/IEEE Design Automation Conference*, pp.177-181, 1993.

[7] E. Bozorgzadeh, R. Kastner, and M. Sarrafzadeh. "Creating and Exploiting Flexibility in Steiner Trees". In *Proc. ACM/IEEE Design Automation Conference*, June 2001.

[8] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. "Pattern Routing: Use and Theory for Increasing Predictability and Avoiding Coupling". In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 777-791, vol. 21, No. 7, July 2002.

[9] A. Caldwell, A. Kahng and I. Markov. "Can Recursive Bisection Alone Produce Routable Placements?". In *Proc. ACM/IEEE Design Automation Conference*, June 2000.

[10] J. Hu, S. Sapatnekar. "A Timing-constrained Algorithm for Simultaneous Global Routing of Multiple Nets". In *ACM/IEEE International Conference on Computer Aided Design*, November 2000.

[11] A. Jagannathan, S.-W. Hur and J. Lillis. "A Fast Algorithm for Context-Aware Buffer Insertion". In *Proc. ACM/IEEE Design Automation Conference*, June 2000.

[12] C. Alpert. "The ISPD98 Circuit Benchmark Suite". In *Proc. International Symposium on Physical Design*, April 1998.

[13] E.F. Moore. *"The Shortest Path through a Maze"*. Annals of the Harvard Computation Laboratory, Vol. 30, Pt. II, 1959.

[14] M. Garey and D. Johnson. "The Rectilinear Steiner Tree Problem is NP-Complete". *SIAM Journal on Applied Mathematics*, March 1977.

[15] J. Cong and X. Yuan. "Routing Tree Construction Under Fixed Buffer Locations". In *Proc. ACM/IEEE Design Automation Conference*, June 2000.

[16] J. Ho, G. Vijayan and C.K. Wong. "A New Approach to the Rectilinear Steiner Tree Problem". In *Proc. ACM/IEEE Design Automation Conference*, June 1989.

[17] K. Kozminski. "Benchmarks for Layout Synthesis - Evolution and Current Status". In *Proc. ACM/IEEE Design Automation Conference*, June 1991.

[18] M. Lai and D.F. Wong. "Maze Routing with Buffer Insertion and Wire sizing". In *Proc. ACM/IEEE Design Automation Conference*, June 2000.

[19] M. Sarrafzadeh and C.K. Wong. *An Introduction to VLSI Physical Design*. McGraw-Hill, New York, NY, 1996.

[20] M. Wang, X. Yang and M. Sarrafzadeh. "DRAGON: Fast Standard-Cell Placement for Large Circuits". In *Proc. IEEE International Conference on Computer Aided Design*, November 2000.

[21] W.J. Sun and C. Sechen. "Efficient and Effective Placement for Very Large Circuits". In *Proc. International Conference on Computer Aided Design*, November 1993.