Creating and Exploiting Flexibility in Steiner Trees *

Elaheh Bozorgzadeh Ryan Kastner Majid Sarrafzadeh

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095-1596
{elib,kastner,majid}@cs.ucla.edu

ABSTRACT

This paper presents the concept of flexibility – a geometric property associated with Steiner trees. Flexibility is related to the routability of the Steiner tree. We present an optimal algorithm which takes a Steiner tree and outputs a more flexible Steiner tree. Our experiments show that a net with a flexible Steiner tree increases its routability. Experiments with a global router show that congestion is improved by approximately 20%.

1. INTRODUCTION

The main purpose of global routing is to find an approximate path (route) for each net. 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. A global routing solution with minimum delay and chip size accomplishes nothing if the detailed router can not find a solution. We refer to this concept as the *routability* of the global routing solution. The routability of the circuit depends on the congestion of the routing. Congestion is attributed to a number of factors e.g. number of vias [1, 7], number of nets routed through a local area, etc. Depending on the constraints the congested areas produce, the detailed router may not be able to find a feasible solution.

Typically, critical nets are a small subset of the overall number of nets; as a rough estimate, 10% of the nets are critical in the MCNC benchmark suite [6]. The delay of non-critical nets is not crucial to the performance on the circuit. Therefore, these nets should be routed in manner which increases the likelihood that the detailed router can find a final, valid solution.

In this work, we study Steiner trees in terms of routability. We introduce the notion of *flexibility* – a geometric property associated with Steiner trees. [2] introduces a timing-driven global routing while considering congestion by exploiting flexibility in routing trees. We attempt to 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.

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 2). 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.

The paper proceeds as follows: Section 2 gives some definitions used in this work. Section 3 presents an algorithm which takes an existing Steiner tree and increases its flexibility. Section 4 gives experimental results which study the impact of flexibility during global routing. We conclude and give some possibilities for future work in Section 5.

2. PRELIMINARIES

We assume in global placement cells are placed into global bins (rectangular regions on a chip) by top-down partitioning the chip. Cells are located at the center of global bins.The global bins and edges can be transformed into a possible irregular grid graph [9].

Congestion in layout means that there are too many nets routed in a local area. *Capacity* (or *supply*) of an edge *e*, c_e is maximum number of nets that can be routed over global bin edge. The routing *demand* of an edge, d_e, is defined as the number of nets passing through the bin edge. An edge is overflown if and only if the $d_e > c_e$. The total overflow reflects the shortage of routing resources for a particular set of capacities. A routing with a minimized total overflow is one of the objectives of our global router [9]. Steiner tree construction is an essential part of global routing. Given a net embedded in a grid graph, a *Rectilinear Steiner Tree* (RST) is a tree interconnecting the terminals of the net and some arbitrary points called *Steiner* points to reduce the cost of routing tree. Construction of steiner tree with minimum cost is a hard problem [7]. Since we are focusing on routability, we define the cost of the Steiner tree as a linear function of the overflow and route length:

$$cost_{tree} = \alpha \times overflow_{tree} + length_{route}$$

In order to use this definition of cost, we must determine a method for assigning the route edges (routing) to the Steiner edges of the tree. Specifically, we need to assign a routing to the edges which are neither vertical nor horizontal. We choose to *pattern route* such edges. Pattern routing is the notion of using predefined patterns to route a two terminal net. We can view a Steiner edge as a two-terminal net. Usually these are simple patterns such as a L-shaped – 1-bend – or a Z-shaped pattern – 2-bends, route restricted within bounding box. Pattern routing is shown to be useful for wireplanning without affecting the overal quality[9].

Ho *et al.* introduced the notion of *stability* in an RST [5]. A RST is stable if there is no pair of edges such that their bounding boxes

*This work was partially supported by NSF under Grant #CCR-0090203

38th DAC, June 18-22, 2001, Las Vegas, Nevada, USA.

intersect or overlap except at a common endpoint (if any) of the two edges. A stable tree is locally optimal when the edges are routed as a Z or L-shaped pattern.

The *topology* of a RST is the vertices and edges of the Steiner tree. The edges are the set of connections between any two vertices. As long as the edges between the vertices remain consistent, the topology is considered unchanged.

A *flexible edge* is an edge of a Steiner tree which has a non-zero bounding box area. Equivalently, it is any edge which is neither vertical nor horizontal.

Theorem 2.1. *An unflexible Steiner edge has only one minimum length routing. A flexible edge has more than one minimum length route[3].*

Assume that there is a Steiner tree which contains an edge that passes through a congested region. A flexible edge has the opportunity to choose another minimum length route in order to avoid the congested region. Looking at Figure 1, we show two Steiner trees for the same net. One tree has some flexible edges while the other has only unflexible 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 unflexible tree has no choice but to be routed through the congested area.

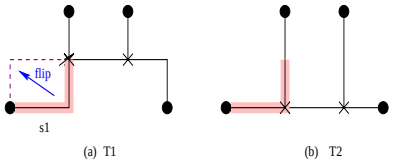

Figure 1: (a)RST with flexible edges. (b) RST with no flexible edges.

We need to define a function which evaluates the flexibility of a given Steiner edge. The function should reflect the routability of the edge. Additionally, a function f should have the following properties:1- $f = 0$ when $l = 0$ or $w = 0$. 2- f is a monotonically increasing function of l and w. The definition of a function expressing flexibility is not clear. Two possible flexibility functions are shown in Equations 1 and 2.The flexibility of a Steiner tree is found by summing the flexibility over all the edges.

$$f_1 = \begin{cases} 0 & \text{if l = 0, w = 0} \\ l + w & \text{Otherwise} \end{cases} \quad (1)$$

$$f_2 = l \times w \quad (2)$$

3. ALGORITHM

In this section, we introduce an algorithm which increases the flexibility of a Steiner tree. Due to space constraints, the corollaries and lemmas stated without proof; please see the technical report [3] for further details.

Problem Formulation:

Given a stable RST, maximize the flexibility of the RST subject to: 1– Topology (inherently wirelength) remains unchanged. 2– No initially flexible edge is degraded in flexibility[1].

Since the terminals of a net are fixed, the flexible edges can only be generated by moving Steiner nodes. The Steiner point p in a stable RST is *movable* if we can move p to any one other location while keeping the RST stable and the topology unchanged. Similarly, an edge e is movable if e can be moved and the resulting RST remains stable with no change in topology.

[1]The problem can easily be extended without this assumption by considering them in overlap cases.

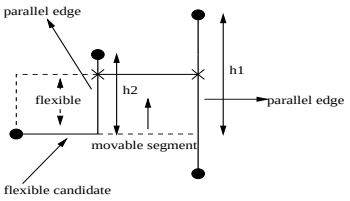

Figure 2: Movable Segment Set.

Theorem 3.1. *An edge in a stable RST is movable if both vertices incident to the edge are Steiner points with edge degree 3 and have an incident edge in same direction.*

We define a set of movable segment s and adjacent edges to s, E_s, as a *Movable Set* (s, E_s). E_s is the set of all adjacent edges to s including segment s. E_s consists of two parallel edges and set of flexible and unflexible candidates. Flexible candidate is an edge that has the potential to become flexible by moving the movable segement (Figure 2).

Figure 3 shows 3 different movable sets (s, E_s). The movable set in Figure 3 (a) has no flexible candidate. The other movable sets have at least one flexible candidate. 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. When a movable set is found, the movable segment can move along the parallel edges. The maximum movement is $\ell_{max} = length(min(h_1, h_2)) - 1$ (See Figure 2). 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.

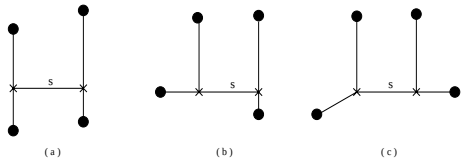

Figure 3: Movable Set with movable edge s.

Lemma 3.1. *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 does not have an edge in the opposite direction of moving direction.*

According to the lemmas and definition of flexibility, we need to search for movable sets which have the potential to generate flexible edges. Finding such edges is shown in the first part of the pseudocode of algorithm *Generate_flexible_tree* in Figure 4. First movable sets of a RST are found. Function *Is_Movable(e)* checks if the edge e is movable according to Theorem 3.1. *Flexible_exist(e)* checks if there is any flexible candidate in movable edge set. *Create_movable_set(e)* generates the movable set associated with edge e.

Lemma 3.2. *The complexity of finding movable edges is $O(E)$, where E is the number of edges in RST.*

By maximizing the flexibility of each movable set, we may restrict the flexibility of another movable set. When there is no such a

ALGORITHM 1. Generate_flexible_tree

input: E_{RST}: *Edge set of an RST of a multi-terminal net.*
output: RST_{new}.

for *each edge* $e \in E_{RST}$
  if *Is_movable(*e*) and Flexible_exist(*e*)*
    Create_movable_set(e*, MovableSetList)*
    Check_overlap(movable_set(e*),overlapList)*
for *each movable set* $M = (s, E_s) \in$ *MovableList*
  if *No_overlap(*E_s*)*
    Move_segment(s*)*
Solve_overlap_and_move(overlapList)

Figure 4: Algorithm *Generate_flexible_tree* Pseudocode.

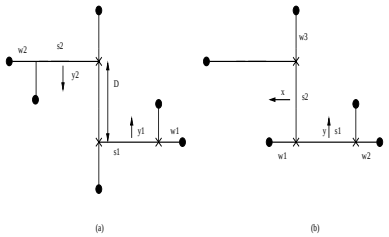

Figure 5: Example of Overlap Type 1 and Overlap Type 2 for Movable Set s_1.

restriction, we easily compute the flexibility function for each moving segment. Since the flexibility function is monotonically increasing in terms of length and width of the edges, the maximum of f occurs if each moving edge is moved to its maximum limit. Unfortunately, the flexibility cannot be computed independently for all movable segements if there are *overlap* between the segments. Two movable segments have overlap with each other if the movable edge s is limited by the movement of the other movable edge or 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 (See Figure 5).

Lemma 3.3. *There are two types of overlap that can occur[3]:*

- **Overlap type 1***: It occurs when one parallel edge of two movable sets is the same, thier moving direction is opposite and their range of move on the shared parallel edge has conflict (Figure 5(a)).*
- **Overlap type 2***: It occurs when flexible edge of a movable segment is a parallel edge of the other moving set (Figure 5(b)).*.

Lemma 3.4. *Each movable segment set can have at most two overlaps of type 1 and two overlaps of type 2.*

In Figure 6, 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.

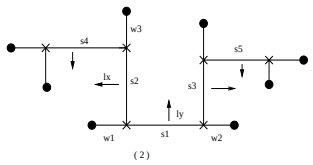

Figure 6: The case in which a movable segment has maximum overlap with other movable sets.

Function *Check_overlap* in Figure 4 finds the overlaps among the movable set. Each edge can participate at most in 2 movable sets according to Lemma 3.4. The complexity of the first loop still remains $O(E)$. In both types of overlap, we need to have a measure for flexibility to decide on moving the movable segments. The behavior of the overlap will depend on the flexibility function. In [3] we have shown how to resolve the overlap when flexibility functions introduced in Section 2 are used. Consider moving sets shown in Figure 5. In both figures, Segment s_1 overlaps with segment s_2. If f_1 is a flexibility function both types of overlap can be resolved in linear time. If function f_2 is used as a metric to evaluate flexibility, the overlap type 1 can be similarly resolved. However for overlap type 2 function f_2 generates quadratic expressions of l and w. The maximum of f_2 occurs on the limits of movable segments s_1 and s_2. When a segment has overlaps from both types on one parallel edge as shown in Figure 6, 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 4, 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 6 and f_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 f is exponential in terms of number of movable sets in the overlapping chain. We expect these chains to be short in most applications.

Lemma 3.5. *The algorithm* Generate_flexible_tree *generates the most flexible ST from a given RST under the defined flexibility function and constraints mentioned in Section 3. If f is a linear function, the algorithm runs in polynomial time. If f is quadratic function the algorithm complexity can be exponential in terms of number of movable set in overlapping chain.*

4. EXPERIMENTS

In this section, we perform experiments which attempt to show the relationship between flexibility and routability. To perform our experiments, we used MCNC standard-cell benchmark circuits [6] and benchmarks from the ISPD98 suite [4]. The circuits were placed into using the Dragon global and detailed placement engine [8]. We performed our experiments using global router based on maze routing [3].

Our experiments focused on 4-terminal nets. We choose the 4-terminal nets which had only one movable edge. We refer to this set of nets as *considered nets*. Initially all other nets routed by our global router. Then, we produced two Steiner trees for each net, an unflexible tree we call *unflex* and a flexible tree named *flex*. The flexibility of the *unflex* is guaranteed to be less than the flexibility of the *flex*. The final results of these two sets are compared as shown in Table 1.

Before we discuss the results, we summarized the data categories of presented in Table 1. "Number of Nets" is the number of 4-terminal flexible nets. "Length of Routes" is the length of the routed Steiner tree (net) summed over all the considered nets. Remember that the route length for the flexible and unflexible tree is equivalent. The overflow for one net is $\sum_{e \in Route}[demand_e - supply_e]$. "Total Overflow" is the summed overflow of the considered nets. "Total

Demand" is the summed demand of the considered nets. "Overflow Improvement" is $\frac{total\ overflow - flex\ overflow}{unflex\ overflow}\%$. Finally, "Demand Improvement" is $\frac{unflex\ demand - flex\ demand}{unflex\ demand}\%$.

In our experiment, we compared the overflow and routing demand of a flexible and unflexible tree. The edges of the Steiner tree (both the flexible and unflexible) 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. Intuitively, overflow determines the amount of routing resources which are needed, but can not be supplied. The demand is simply amount of other routings which are competing for the same routing resources.

Circuit	Number of Nets	Length of Routes	Overflow Improve %	Demand Improve %
avql	18	151	73.69	15.16
avqs	4	26	7.69	5.49
avqs.2	14	127	22.98	7.44
biomed	5	170	3.09	2.78
biomed.2	8	318	22.29	1.58
ibm01.1	24	259	19.10	4.82
ibm01.2	66	1519	19.80	4.68
ibm05.1	4	94	7.69	3.37
ibmo5.2	16	485	5.40	1.72
ibm10.1	99	2437	11.88	2.97
ibm10.2	221	10707	23.60	5.11
Primary1	4	45	26.19	4.22
Primary2	21	652	19.44	5.00
Total	504	16990	-	-
Average	-	-	20.17	4.49

Table 1: Overflow and Congestion Results when Z-shape patterns are used for flexible edges.

Referring to Table 1, we can see that the flexible tree produces a routing which has, on average, 20.17% less overflow than the unflexible 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 unflexible tree. We conducted similar experiments using L-shaped pattern-route for the steiner edges. The results have a similar trend as the Z-shaped pattern results. 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.

5. CONCLUSION AND FUTURE WORK

This paper studied the problem of routability in global routing from a different perspective. Flexibility – a 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 proposed to generate an optimally flexible RST given a stable RST. Our initial experimental results supports our idea. We believe that the proposed algorithm can be used in early stages to assign the location of Steiner points while considering routability. As future work we plan to study the flexibility functions in more detail and integrate the flexibility algorithm into our existing global router.

6. ACKNOWLEDGEMENT

We wish to thank Dr. Dirk Stroobandt for his valuable feedback regarding various aspect of this work.

7. REFERENCES

[1] A. Kahng, S. Mantik and D. Stroobandt. "Requirements for Models of Achievable Routing". In *Proc. International Symposium on Physical Design*, April 2000.

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

[3] E. Bozorgzadeh, R. Kastner, and M. Sarrafzadeh. "Flexibility in Steiner Trees". Technical report, Computer Science Department, UCLA, October 2000. URL: *http://www.cs.ucla.edu/~elib/pub.html*.

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

[5] 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.

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

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

[8] 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.

[9] R. Kastner, E. Borzorgzadeh and M. Sarrafzadeh. "Predictable Routing". In *Proc. IEEE International Conference on Computer Aided Design*, November 2000.