Constructing Lower and Upper Bounded Delay Routing Trees Using Linear Programming

Jaewon Oh, Iksoo Pyo and Massoud Pedram

CENG 96-05

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213) 740-4458

February 28, 1996

Constructing Lower and Upper Bounded Delay Routing Trees Using Linear Programming

Jaewon Oh, Iksoo Pyo and Massoud Pedram
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089

February 28, 1996

Abstract

This paper presents a new framework for solving Lower and Upper Bounded delay routing Tree (LUBT) problem using linear programming. LUBT is a Steiner tree rooted at the source such that the delays from the source to sinks lie between the given lower and upper bounds. It is shown that our proposed method produces minimum cost LUBT under a given topology for linear delay model. Unlike some recent works which control only the amount of skew, which is merely the difference between the maximum and the minimum source-sink delay, we construct routing trees with distinct lower and upper bounds on the source-sink delays. This extension exploits all the flexibility that is present in low power and high performance clock routing tree design.

Contents

1 Introduction 1

2 Terminology and Problem Definition 2

3 Topology and the Bounds 4

4 Edge-Based Formulation(EBF) 5

4.1 Steiner Constraints 6

4.2 Delay constraints 7

4.3 Summary of the Formulation 7

4.4 Optimality of Our Method 7

4.5 Example 8

4.6 Reduction of the constraints 9

4.7 The EBF Method is NOT for Euclidean Metric 9

5 Placement of Steiner points 10

6 Tolerable Skew Clock Routing 13

7 Extensions of EBF to other problems 13

8 Experimental Results 15

9 Conclusion 17

10 Appendix 21

10.1 Feasible Regions Revisited 22

10.2 Intersection of TRRs 22

10.3 Proof of Theorem 4.1 by contradiction 24

1 Introduction

Control of signal delays while minimizing the routing cost has been the major issue in routing in the recent past. Routing affects various aspects of chip design such as chip area, performance and power dissipation. In the global routing problem, the routing cost is minimized while the maximum delay from the source to any sink is kept within a given bound. Global routing algorithms for bounded path length minimum spanning trees and Steiner trees are presented in [1]. In the clock routing, the routing cost is minimized while the skew of the routing tree, which is the difference between the minimum and maximum delay from the source to any sink, is minimized. A number of zero skew clock routing techniques are presented in [2], [3], [4] [7] and [5].

In practice, exact zero skew is not an actual design requirement. We can allow some tolerable skew with which the system can function correctly. Bounded skew clock routing methods are presented in [8] and [9] to reduce the routing cost over zero skew routing. Their method is based on the observation that with some skew bounds, the feasible locations for Steiner points are *octilinear convex polygons* [1]. The topology is generated by successively joining the two nearest feasible regions and placement of Steiner points in the feasible region is done heuristically. These researchers however only considered the skew bound and did not control the maximum source-sink delay. Excessively long wires may require more buffers and cause slower rising and fall time. More buffers and slower switching results in more power consumption, which is a primary concern in many applications. A power optimal clock routing with bounded skew and bounded maximum source-sink delay under the Elmore delay model was presented in [10]. However, delays are controlled by buffer sizing, rather than by controlling the wire lengths. Their clock tree is an equal source-sink path length Steiner tree (zero skew tree under linear delay) regardless of skew bounds. Their routing cost may be unnecessarily high when non-zero skew is required.

Our method also allows the user to specify different delay bounds for each individual sink, which can lead to a further reduction of the routing cost. Consider a pipelined design with L pipeline stages. If the combinational delay through each pipeline stage happens to be different (which is often the case), then the lower and upper bound delays on the clock source to input pins of the Flip Flops in stage i can be different from those to input pins of the Flip Flops in stage j. This difference (which may be substantial) can be exploited to reduce the power consumption of the clock tree. Our proposed formulation thus allows for distinct lower and upper bound delays for each Flip Flop/Sink of the clock tree. Also, in case of global routing, if there is a path violating short path delay constraints, a usual practice is to insert a delay buffer in the short path. Instead, we can adjust wire length until the delay is larger than some lower bound which meets the short path delay constraints. Since routing delays dominate gate delays these days,

[1] In zero skew routing, feasible locations for Steiner points are line segments.

wire-length control is more effective in introducing delays than buffers. Also it will take less area and consumes less power than buffer insertion method. These motivated us to develop develop a method for controlling the path lengths such that any delays lie between given upper and lower bounds.

Mathematical programming in Manhattan metric has not received much attention so far. The main difficulty arises when the Manhattan distance is expressed in terms of absolute valued functions. The Manhattan distance between two points (x_1, y_1), (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$ and a mathematical programming problem with these absolute valued functions is not easy to solve. These functions are not differentiable and the signs of absolute terms in the formulation, whether they appear in the objective function or in the constraints, must be maintained during the search for a solution. When we move from a feasible solution to a better solution, we can only move as far as none of the signs change. Once we reach a new solution, we update the signs of absolute terms and continue the search with the new signs. This slows down the optimization speed significantly. For this reason, many researchers have replaced Manhattan distance with the less accurate Euclidean or Quadratic distance or other approximations.

Our method, Edge-Based Formulation (EBF) overcomes this problem. The variables to the mathematical programming are not the positions of the Steiner points. Instead, the variables are the edge lengths of the tree, eliminating any absolute terms in the formulation. The proposed formulation leads to a simple linear programming problem under the linear delay model which can be solved optimally in polynomial time. Our formulation results in a nonlinear programming problem under the Elmore delay model which can be solved heuristically using any general purpose nonlinear programming solver. Once the edge lengths are determined, the position of Steiner points are determined using geometric considerations.

The remainder of this paper is organized as follows. Section 2 gives our terminology and defines the LUBT problem. Section 3 describes the relationship between the topology and the bounds. Section 4 formulates the EBF problem. Section 5 describes our method for embedding the tree with given topology and edge lengths in the Manhattan plane. Section 6 describes tolerable skew clock routing using LUBT. Section 7 describes some of extensions of our method. Section 8 shows our experimental results and Section 9 concludes the paper. Finally the Appendix proves the correctness of our proposed algorithm.

2 Terminology and Problem Definition

Let $T(S, E)$ be a given rooted tree topology. Let $S = \{s_0, s_1, s_2, \ldots, s_n\}$ be the vertices of T. Among these vertices, s_0 is the root (source) of T whose location may or may not be given. $\{s_1, s_2, \ldots, s_m\}$ are the sinks whose locations are given, and $\{s_{m+1}, \ldots, s_n\}$

are the steiner points whose locations are to be determined. *points* refer to the source, sinks and Steiner points. Since T is a tree, there is a unique path between any two points. We say s_i is the *parent* of s_j if in the path from s_0 to s_j, s_i comes immediately before s_j. Inversely, s_j is a child of s_i.

Let $E = \{e_1, e_2, \ldots, e_n\}$ be the set of edges in the tree. We identify each point s_i, except s_0, of the rooted topology T with edge e_i. So e_i connects s_i to its parent in T. When there is no confusion, we use e_i to refer to the ith edge or to refer to the edge length of the ith edge. Also we use s_i to denote the ith point or the location of the ith point.

Let $dist(s_i, s_j)$ be the Manhattan distance between s_i and s_j. As mentioned in section 1, we determine the edge lengths of the tree first and then find the locations of Steiner points. Suppose s_p is the parent of s_i and we know the edge length of e_i. It is clear that s_i and s_p must be placed such that:

$$e_i \geq dist(s_i, s_p)$$

If $e_i = dist(s_i, s_p)$, we say e_i is *tight*. If $e_i > dist(s_i, s_p)$, we say e_i is *elongated*. When $e_i = 0$, e_i is said to be *degenerate*. In this case, s_i and s_p coincide on the same location. We allow wire elongation in the routing tree. So it is possible that $e_i > dist(s_i, s_p)$ in the final tree embedding.

The cost of a tree T is the sum of the edge lengths, i.e. $cost(T) = \sum_{i=1}^{n} e_i$.

Let $path(s_i, s_j)$ be the set of edges in the path from s_i to s_j in T. We define the *delay of a sink* s_i as the delay from the source to that sink and denote it as $delay(s_i)$. Since we are using linear delay model, $delay(s_i)$ is defined by:

$$delay(s_i) = \sum_{e_k \in path(s_0, s_i)} e_k \tag{1}$$

Under linear delay model, *diameter* is the distance between the farthest two sinks. If the source location is not given, the *radius* is defined as the half of the *diameter*. Otherwise, the *radius* is the distance from the source to the farthest sink.

$skew(s_i, s_j)$ is the difference between the delays of sinks s_i and s_j. $skew(T)$ is the skew of the tree T and is the maximum of $skew(s_i, s_j)$ for all the sinks s_i, s_j. We now define *Lower/Upper Bounded routing Tree* (LUBT) problem.

Definition 2.1 Lower/Upper Bounded routing Tree (LUBT) Problem: *Given a rooted tree topology T(S,E) and two sets of bounds* $L = \{l_1, l_2, \ldots, l_m\} \subset \mathcal{R}$, $U = \{u_1, u_2, \ldots, u_m\} \subset \mathcal{R}$, *find a tree embedding in the Manhattan plane, i.e., find the locations of Steiner points and the values of* e_i*'s, such that the tree cost is minimal and the delay from the source* s_0 *to any sink* s_i *satisfies the following inequalities:*

$$l_i \leq delay(s_i) \leq u_i \qquad i = 1, \ldots, m \tag{2}$$

where l_i's are lower bounds and u_i's are upper bounds satisfying the following:

$$0 \leq l_i \leq u_i \quad and \quad u_i \geq dist(s_0, s_i) \qquad or \tag{3}$$

$$0 \leq l_i \leq u_i \quad and \quad u_i \geq radius \tag{4}$$

Equation 3 holds when the source location is given and Equation 4 holds when the source location is not given.

So LUBT is a tree whose source to any sink delay is no less than the given lower bound and no more than the given upper bound.

3 Topology and the Bounds

When we say topology, we mean the connectivity between the points. The final embedding of the tree may look very different since some points may overlap on the same location or some edges may degenerate. In any case, the topology of the tree is preserved regardless of the tree embeddings.

It can be easily shown that for a given topology T, the solution to a LUBT problem may not exist depending on the bounds. Figure 1-(a) is a simple example where it has no solution. However, when the topology is (b) or (c), there is a solution.

However, if every sink is a leaf node in the topology, then there is a solution for any lower and upper bounds.

Lemma 3.1 *Given a tree topology where every sink is a leaf node of the tree, it is always possible to find a LUBT for any lower and upper bounds given in Equation (3),(4) under linear delay model.*

Proof When all the sinks are leaves of T, in the path from the source to a sink, there are only Steiner points if there are any. One can obtain a Shortest Path Tree (SPT) by placing all the Steiner points to the source and making $e_{m+1} = e_{m+2} = \ldots = e_n = 0$. SPT already satisfies the upper bound. The lower bound can be met by elongating $e_1, \ldots, e_m$ until they are equal to the lower bound (These elongation do not violate the upper bound constraints since the upper bounds are higher than the lower bounds). The tree so formed is a valid LUBT. □

Although our method applies to any topology, we will mainly consider topologies in which every sink is a leaf node since they guarantee the existence of solutions.

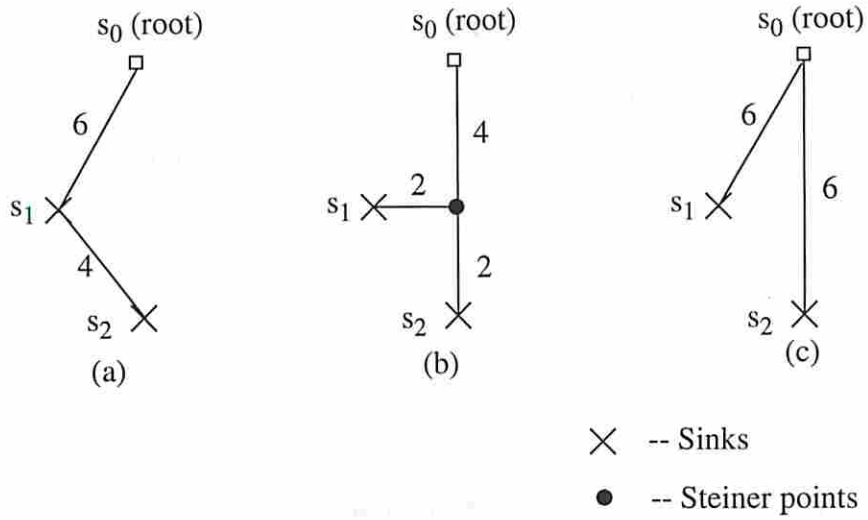

Figure 1: Three topology variations for the same source and sinks locations. Assume the lower bounds are zero and upper bounds are 6 for all the sinks. (a) has no Steiner point and the tree shown is the only possible tree. The path length from the source to s_2 cannot be less than 6. So no LUBT with those bounds can be made. However, (b) and (c) can have LUBTs with such bounds.

In Manhattan space, every Steiner point has degree 3 or 4. When a Steiner point S has degree 4, we split S into S_1 and S_2. We attach two edges of S to S_1 and the other two edges to S_2. Then we join S_1 and S_2 with a new edge e and fix edge length of e to zero (Figure 2). If we do this for all the Steiner points with degree 4, then every Steiner point in T can be made to have degree 3. This conversion does not affect the solution to our LUBT problem. Its purpose is to make every Steiner point has exactly one parent and two children for the convenience of the following sections. In particular, if the root location is not given, it has two children and no parent. If the root location is given, the root is of degree one and has only one child. In the remainder of this paper, we assume that every Steiner point in T has degree 3.

4 Edge-Based Formulation(EBF)

We present an EBF formulation for the LUBT problem. The edge lengths are determined such that they satisfy both the *Steiner constraints* and the *delay constraints* as described next.

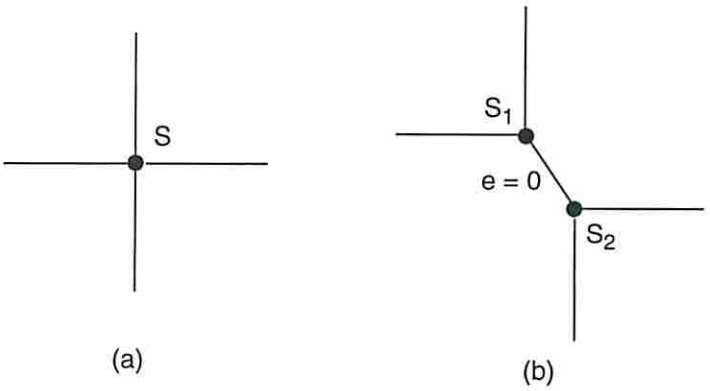

Figure 2: Splitting of Steiner point S of degree 4 in (a) into two Steiner points S_1 and S_2 of degree 3 in (b). The edge length of e is set to zero.

4.1 Steiner Constraints

When we determine the edge lengths, it is important that there exist valid locations for Steiner points that achieve those edge lengths. The following is a necessary condition for edge lengths.

$$\sum_{e_k \in \ path(s_i,s_j)} e_k \geq dist(s_i, s_j) \qquad \text{for every pair of sink } s_i, s_j \tag{5}$$

Otherwise, the two sinks s_i, s_j will get separated, breaking the tree into two components. We will prove that the above equation is a sufficient condition also. The following theorem is the key feature of this paper.

Theorem 4.1 *Let $e_1^*, \ldots, e_n^*$ be a solution to the following set of linear inequalities.*

$$\sum_{e_k \in \ path(s_i,s_j)} e_k \geq dist(s_i, s_j) \qquad \textit{for every pair of sink } s_i, s_j \tag{6}$$

Then there exist placements of steiner points $s_{m+1}, \ldots, s_n$ such that

$$e_k^* \geq dist(s_k, s_p) \quad k = 1, \ldots, n \tag{7}$$

where s_p is the parent of s_k.

Proof The proof is provided in the Appendix.

4.2 Delay constraints

The delay constraints dictate that the delays from the source to any sink is bounded. Under linear delay model, we have:

$$l_i \leq \sum_{e_k \in \ path(s_0, s_i)} e_k \leq u_i \qquad \text{for all sinks } s_i \tag{8}$$

4.3 Summary of the Formulation

Our objective is to minimize the total sum of edge lengths. Together with the Steiner constraints and the delay constraints, we have the following mathematical formulation.

$$\begin{aligned} \text{Min} \quad & \sum_{k=1}^{n} e_k \\ \text{Subject to} \quad & \sum_{e_k \in \ path(s_i, s_j)} e_k \geq dist(s_i, s_j) \qquad \text{for every pair of sink } s_i, s_j \\ & l_i \leq delay(s_i) \leq u_i \qquad \text{for all sinks } s_i \end{aligned} \tag{9}$$

Depending on the value of l_i and u_i, the LUBT problem reduces to many different problems.

- $[l_i = 0, u_i = \infty]$ This is an unbounded delay tree problem. So LUBT is an optimal Steiner tree under given topology.
- $[l_i = 0, u_i < \infty]$ This is an upper bounded delay tree problem. So LUBT is a general global routing problem.
- $[l_i > 0, u_i < \infty]$ This is a lower/upper bounded delay tree problem. So LUBT is equivalent to a bounded skew clock routing tree problem with a specific upper bound.
- $[l_i = u_i = c < \infty$ for some constant c] This is equivalent to a zero skew clock routing problem. It was shown in [7] that under linear delay, an optimal zero skew tree under a given topology has $l_i = u_i = radius$.

4.4 Optimality of Our Method

Our method constructs minimum cost LUBT as described next.

Theorem 4.2 *Our method gives minimum cost LUBT for a given topology.*

Proof Let T be a LUBT solved by EBF. Suppose T' is an another LUBT whose cost is lower than T. Then T' should satisfy the Steiner constraints and the delay constraints since these constraints are necessary conditions for LUBT. Since cost(T) is obtained by a mathematical programming which should produce optimal cost for a given set of constraints, cost(T') cannot be lower than cost(T). □

4.5 Example

Consider the topology of Figure 3. We want to find a LUBT for lower bound of 4 and upper bound of 6 for all the sinks. Assuming the source position is not given, we have the following formulation.

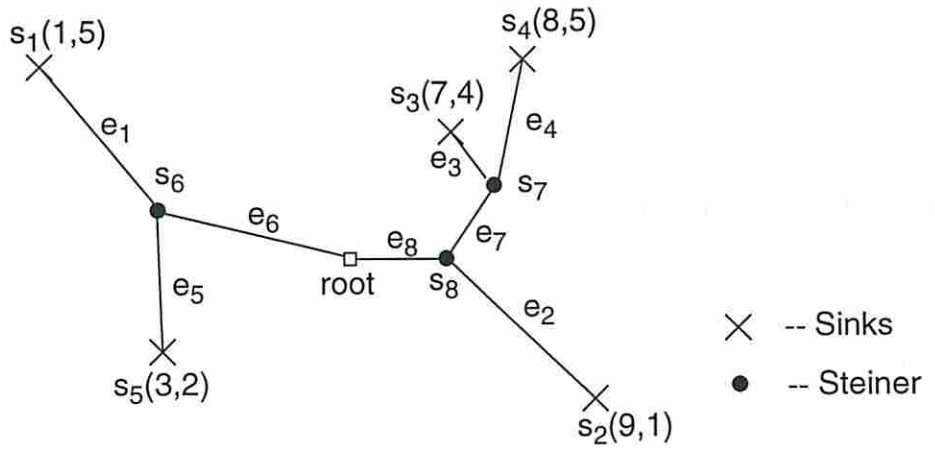

Figure 3: A 5 point example

$$
\begin{array}{ll}
\text{Min} & e_1+e_2+e_3+e_4+e_5+e_6+e_7+e_8 \\
\text{Subject to} & \\
& \left.\begin{array}{l}
e_1+e_6+e_8+e_2 \geq 12 \\
e_1+e_6+e_8+e_7+e_3 \geq 7 \\
e_1+e_6+e_8+e_7+e_4 \geq 7 \\
e_1+e_5 \geq 5 \\
e_2+e_7+e_3 \geq 5 \\
e_2+e_7+e_4 \geq 5 \\
e_2+e_8+e_6+e_5 \geq 7 \\
e_3+e_4 \geq 2 \\
e_3+e_7+e_8+e_6+e_5 \geq 6 \\
e_4+e_7+e_8+e_6+e_5 \geq 8
\end{array}\right\} \text{Steiner Constraints} \\
& \left.\begin{array}{l}
4 \leq e_1+e_6 \leq 6 \\
4 \leq e_2+e_8 \leq 6 \\
4 \leq e_3+e_7+e_8 \leq 6 \\
4 \leq e_4+e_7+e_8 \leq 6 \\
4 \leq e_5+e_6 \leq 6
\end{array}\right\} \text{Linear delay Constraints}
\end{array}
$$

4.6 Reduction of the constraints

There are $\binom{m}{2}$ Steiner constraints in EBF where m is the number of sinks. Together with the $2m$ delay constraints(m constraints for lower bounds and m constraints for upper bounds) , there are $\binom{m}{2} + 2m$ constraints in total. However, many Steiner constraints can be deleted using geometric considerations. Also the delay constraints help delete some of the Steiner constraints. For zero skew clock routing, it can be shown that all the constraints are reduced to only n linear inequalities where n is the number of edges. Besides, with zero skew, inequalities can be replaced with equalities. The constraints are reduced to n linear equations, so no optimization is necessary. Simply solving the n linear equations gives the optimal solution. The reduction of constraints speeds up the execution of our algorithm.

4.7 The EBF Method is NOT for Euclidean Metric

Our method does not work for Euclidean metric. A simple counter example is shown in Figure 4. The sinks are located at the vertices of an equilateral triangle with each side

is of length one. We have the following inequalities.

$$e_1 + e_2 \geq 1$$
$$e_2 + e_3 \geq 1$$
$$e_1 + e_3 \geq 1$$

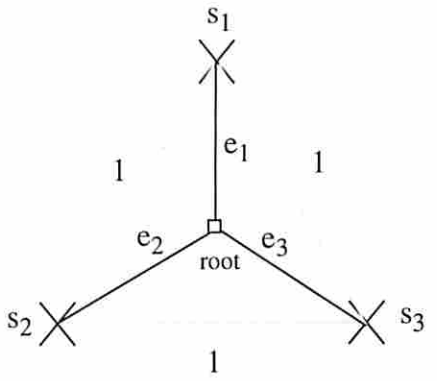

Figure 4: Example where EBF does not work in Euclidean space. Three sinks are located on the vertices of a unit length equilateral triangle.

An obvious solution is $e_1 = e_2 = e_3 = 1/2$, but there is no feasible location for the root which would satisfy these edge lengths.

5 Placement of Steiner points

Once the edge lengths are determined, the actual positions of Steiner points (and the root if its position is not given) should be determined. Our method for placement of Steiner points is similar to *Deferred Merge Embedding* (DME) [7] algorithm exploited by most zero skew and bounded skew clock routing algorithms. In DME algorithm, the feasible regions for Steiner points and the edge lengths are found in bottom up fashion, and then Steiner points are placed in the feasible regions in top down fashion. Our method is different in that edge lengths are predetermined and the feasible regions are rectangular regions instead of simple line segments as in zero skew algorithms or octilinear regions as in bounded skew algorithms.

We revise the DME algorithm along with our own definitions. Let *Rectangular Region*, or *RR*, be a set of the points on the boundary and the interior of a rectangle in a Manhattan space. When *RR* is rotated by +45 degrees from its center, we call it *Tilted Rectangular Region*, or *TRR* (Figure 5-(a)). The set of points within a fixed distance from a *TRR* is also a *TRR* (Figure 5-(b)). We denote $TRR(TRR_a, \mathrm{r})$ as a *TRR* whose points are within distance r from TRR_a. Formally,

$$TRR(TRR_a, r) = \{t \in \mathcal{R}^2 | s \in TRR_a, dist(s,t) \leq r\}$$

The intersection of two or more *TRR*s is also a *TRR* (Figure 5-(c)).

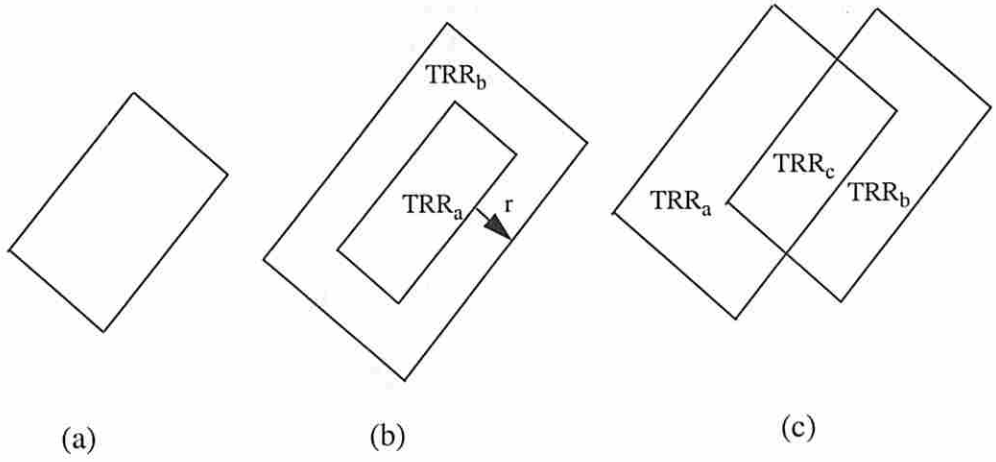

Figure 5: (a) A TRR, (b) $\text{TRR}_b = \text{TRR}(\text{TRR}_a, \text{r})$, (c) $\text{TRR}_c = \text{TRR}_a \cap \text{TRR}_b$

The boundary of a *TRR* has four sides. If the length of every side of a *TRR* is equal, then it is called a *Square TRR*. A Square *TRR* is analogous to the circle in Euclidean space, and has a *center* and the *radius* defined as the distance from the center to the *TRR*'s boundary. The *width* of *TRR* is defined as the length of smaller sides. When the width of a *TRR* is zero, it looks like a line segment. Nevertheless, it is still a *TRR*. Also, a singleton set of one point, such as $\{s_k\}$ is also a *TRR*.

Bottom Up Feasible Region Build-Up

Suppose s_i and s_j are two sinks and their parent s_k is a Steiner point. The intersection of $TRR(\{s_i\}, e_i)$ and $TRR(\{s_j\}, e_j)$ defines the feasible region of s_k. That is, s_k cannot be placed outside this feasible region. We refer to the new *TRR* as the *Feasible Region* of s_k, or simply FR_k. The *TRR* for s_k is constructed by $TRR(FR_k, e_k)$. This *TRR* is again used for construction of the *FR* of the parent of e_k. So we can denote $TRR_k = TRR(FR_k, e_k)$. Starting from the leaves of the tree, this process of constructing *FR*s and *TRR*s goes up the tree until we meet the root (Figure 6). When FR_k is a simple line segment, both e_i and e_j are tight. When FR_k has non zero width, at least one of the edges should be elongated.

Top Down Placements

Once the feasible regions of all points are found, the actual placements of steiner points are found in a top-down fashion. If the position of the source(root) is not given, we

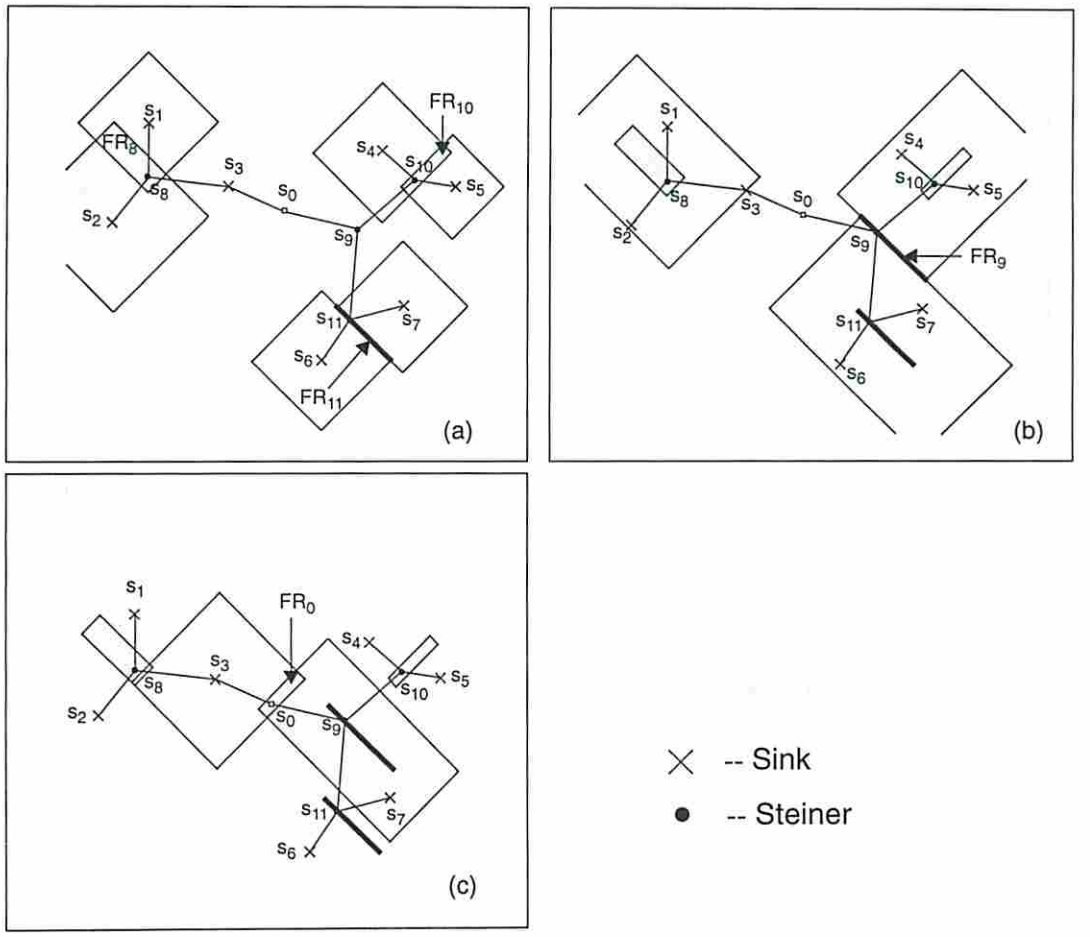

Figure 6: Bottom up feasible regions (shaded area/bold lines) and TRR (solid lines) creations in the order (a) through (c). The locations of Steiner points are not necessarily their actual positions.

choose any position in FR_0. Note that FRs for sinks are simply their given locations. Let s_p be a Steiner point whose location is determined and s_l, s_r be the two children of s_p. Make a Square TRR which is centered at s_p and has a radius of e_l, i.e., make a $TRR(\{s_p\}, e_l)$. The intersection of FR_l and $TRR(\{s_p\}, e_l)$ gives the possible placements of s_l. This intersection is not empty. Then we place s_l anywhere within the intersection. Do the same for s_r. Starting from the root, this procedure is repeated in top down fashion until we meet leaves of the tree (Figure 7).

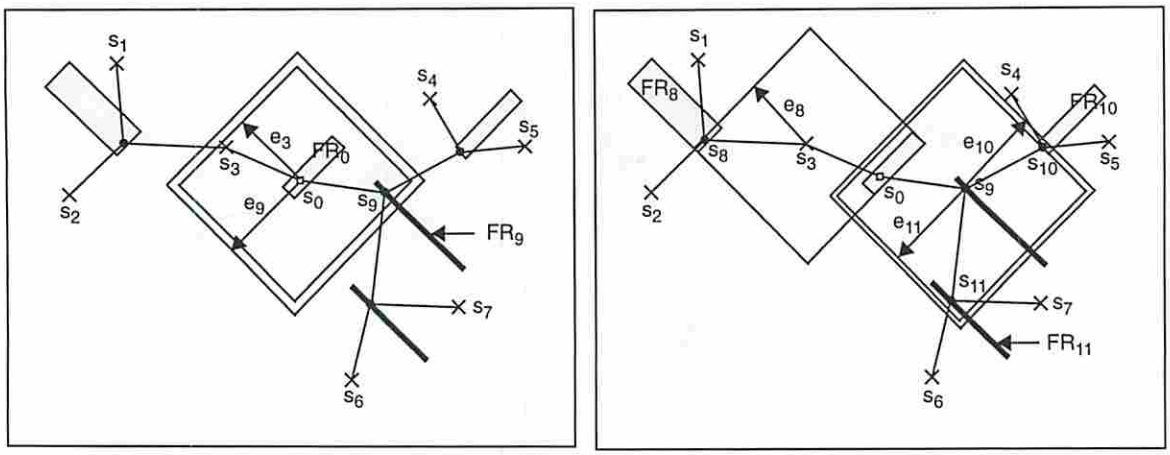

Figure 7: Placement of Steiner points in their respective feasible regions in the order (a) and (b). In (a), s_0 is placed somewhere within FR_0 and the location of s_9 is determined (s_3 is a sink, so its location is fixed). In (b), the locations of s_8, s_{10}, s_{11} are determined. For example, the placement of s_{10} is within the intersection of $TRR(\{s_9\}, e_{10})$ and FR_{10}

6 Tolerable Skew Clock Routing

An important application of LUBT is the tolerable skew clock routing. A common design requirement in clock routing is to have a common delay upper bound u for all the sinks and to bound the $skew(s_i, s_j)$ by some constant d. That is,

$$\begin{aligned} &delay(s_i) \leq u \qquad \text{for all sinks } s_i \\ &|delay(s_i) - delay(s_j)| \leq d \qquad \text{for all sinks } s_j, s_j \end{aligned} \tag{10}$$

By letting $u_i = u$ and $l_i = l$ for $l - u = d$ in the delay constraints of EBF, we have

$$l \leq delay(s_i) \leq u \qquad \text{for all sinks } s_j \tag{11}$$

$delay(s_i)$ in Equation (11) satisfy the design requirement of Equation (10). Thus LUBT is a bounded skew clock routing with an upper bound on all sink delays.

7 Extensions of EBF to other problems

In this section, we consider extensions of the EBF method to some other problems. In all cases, the Steiner constraints remain the same. We can only make modifications to the delay constraints or the objective function to obtain other problems.

The Elmore delay

The Elmore delay is defined as follows [4]. Let T_k be the subtree of the routing tree rooted at s_k. We use C_k to denote the effective tree capacitance at s_k, namely the sum of sink and edge capacitances of T_k. Let the unit resistance and unit capacitance of routing wire be r_w and c_w respectively. Then the delay at a sink s_j is defined by:

$$delay(s_j) = \sum_{e_k \in\ path(s_0,s_j)} r_w e_k \ (\frac{c_w e_k}{2} + C_k) \tag{12}$$

The delay equation is quadratic with respect to e_k's (Note that C_k is also a function of edge lengths). With this delay function, our delay constraints becomes:

$$l_i \leq \sum_{e_k \in\ path(s_0,s_j)} r_w e_k \ (\frac{c_w e_k}{2} + C_k) \leq u_i \quad \text{for all sinks } s_j$$

Since the Elmore delay function is quadratic and the sum of the quadratic terms is positive (or *posynomial*), the delay function is strictly convex (positive definite [12]). The feasible set defined by a convex function with both lower and upper bounds is however not a convex set. So the EBF with Elmore delay constraints is not a convex programming problem. However if we don't have lower bounds ($l_i = 0$), then our formulation is a convex programming.

Different weights on edges

In the EBF method, the objective is the cost of the tree where the weight of each edge is equally weighted to one. However, some edges may be given higher weights to account for wire-ability concerns, blockage, type of metals used, crosstalk or switching activities. In that case we can give different weights $w_1, w_2, \ldots, w_n$ to edges. The objective function for the EBF thus becomes:

$$\text{Min} \sum_{k=1}^{n} w_k e_k$$

Our resulting problem can still be solved efficiently.

8 Experimental Results

The EBF is a Linear Programming problem which can be solved efficiently using a number of commercially available LP solvers. Especially we have chosen LOQO as our solver [11]. LOQO uses the interior point method[2] which is known to be faster than Simplex method for large problems. We have implemented our algorithm in C for SPARC and HPPA workstations.

The topology generator is adopted from [9]. Their topology generator is based on nearest neighbor merge [5] technique and the topology changes dynamically during the construction phase based on the skew. The topologies are full binary trees in which every sink is a leaf node. So by Lemma 3.1, a solution exists for any lower and upper bounds. We tested our method on benchmark data prim1, prim2 [2] and r1, r3 [4]. Our results are compared to those reported in [9] in Table 1. Note that the bounds are normalized to the radius. Algorithm of [9] produces optimal solutions for infinite skew bounds and suboptimal solutions for small skew bounds. Since [9] accepts only the skew bounds and does not allow the user to specify lower/upper bounds, we first ran their algorithm with a skew bound and extracted the topology and the actual shortest/longest sink delays from the solution. Then we ran our algorithm with those shortest/longest sink delays as our lower/upper bounds of LUBT for the same topology.

Our algorithm has a flexibility to control the upper bounds for the same skew. For example, algorithm of [9] returns a tree cost of 206140.0 for prim2 with skew bound of 0.5. The [shortest, longest] sink delays are [0.741, 1.241]. This algorithm however cannot produce other solutions that have the same skew bound but prescribed lower and upper bound delays. For example, [0.5, 1], [0.6, 1.1] or [0.9, 1.4] cannot be generated by the algorithm of [9]. Our algorithm can do this and the results are presented in Table 2 for prim1 and prim2 with skew bounds 0.3 and 0.5. Note that for the same skew, the longest delay can be reduced with little increase in the tree cost.

In addition, [9] cannot produce solutions with zero lower bound except the case when the skew bound is infinite. Trees with zero lower bounds and some finite upper bounds are useful for global routing. Table 3 shows results for some other interesting bound combinations useful for global routings and bounded skew - bounded longest delay routings. Note that as the skew bound is tightened, the tree cost increases.

A trade off curve of various [lower, upper] bounds versus the tree cost of prim2 benchmark is shown in Figure reftradeoff.

[2] Also known as Kamakar's method.

	[9]				LUBT
bench	skew bound	shortest delay	longest delay	tree cost	tree cost
prim1	0.000	1.000	1.000	132565.0	132539.75
	0.010	0.995	1.005	130060.2	129872.23
	0.050	0.968	1.018	122779.0	122020.00
	0.100	0.910	1.020	113805.0	112887.03
	0.500	0.648	1.148	93650.0	93647.38
	1.000	0.439	1.439	84915.0	84915.00
	2.000	0.029	2.029	79860.0	79840.03
	∞	0.000	∞	79810.0	79810.00
prim2	0.000	1.000	1.000	315630.0	315628.20
	0.010	0.990	1.000	305332.0	303963.30
	0.050	0.945	1.000	268497.0	267062.90
	0.100	0.954	1.054	251540.0	249448.30
	0.500	0.741	1.241	206140.0	205783.60
	1.000	0.382	1.382	182490.0	182457.20
	2.000	0.114	2.114	179040.0	179040.00
	∞	0.000	∞	173200.0	173200.00
r1	0.000	1.000	1.000	1312498.0	1311913.38
	0.020	0.996	1.023	1356429.8	1343863.00
	0.060	1.000	1.059	1867170.0	1839234.60
	0.100	0.947	1.032	1797884.9	1750177.80
	0.500	0.714	1.214	932256.5	931271.69
	1.000	0.444	1.444	848555.5	847653.00
	2.000	0.053	1.884	780289.0	780289.13
	∞	0.000	∞	780100.0	780100.25
r3	0.000	1.000	1.000	3331097.5	3330921.00
	0.010	0.996	1.006	3227565.5	3212405.00
	0.050	0.984	1.034	2960921.5	2930180.00
	0.100	0.918	1.018	2732820.5	2709491.30
	0.500	0.741	1.241	2261973.0	2254820.50
	1.000	0.566	1.566	2137096.0	2135432.00
	2.000	0.336	2.336	2028338.0	2027412.00
	∞	0.000	∞	1929421.0	1929421.00

All bounds are normalized to the radius.

Table 1: Routing costs for [9] and for the LUBT method

bench	LUBT skew bound	lower bound	upper bound	tree cost
prim1	0.3	0.70	1.00	103219.5
		0.80	1.10	102122.9
		*0.89	*1.19	103051.8
		0.95	1.25	103671.0
	0.5	0.50	1.00	98120.7
		0.60	1.10	93152.0
		*0.65	*1.15	93647.4
		0.75	1.25	94700.0
prim2	0.3	0.70	1.00	247834.4
		0.80	1.10	237720.3
		*0.85	*1.15	225650.0
		0.95	1.25	230756.0
	0.5	0.50	1.00	212068.8
		0.60	1.10	211034.6
		*0.74	*1.24	205783.6
		0.85	1.35	207344.5

All bounds are normalized to the radius.
*: bounds produced by [9].

Table 2: Routing cost of LUBT for the same skew but different upper bounds

9 Conclusion

We proposed a new method of solving lower/upper bounded delay routing tree (LUBT) problems. The method is based on linear programming in which the variables are the edge lengths of the tree. The LUBT problem is a generalization of global routing and clock routing. Our method produces optimal LUBT for a given topology under the linear delay model. Due to optimality of our method, we can immediately know the existence of a solution for a given topology and bounds since, in case there is no solution, there will be no initial feasible solution to EBF.

Implementation of the EBF method under the Elmore delay model is currently being investigated. Under the Elmore delay, the optimality of the LUBT cost is assured only when the lower bound is zero. When the lower bound is not zero, a sequential quadratic optimization is needed to solve the EBF.

The EBF method is a general framework for optimization problem in Manhattan space. We are also considering an application of the EBF to the placement problems and other related problems.

Our method requires input tree topology. The topology generator we adopted [9] uses the amount of skew to guide the topology generation, rather than the explicit lower/upper bounds. So future work will include better topology generation which is guided by both the lower and the upper bounds, and at the same time, results in lower tree cost.

bench	LUBT lower bound	upper bound	tree cost
prim1	0.99	1.00	129818.3
	0.98	1.00	127570.2
	0.95	1.00	121833.6
	0.90	1.00	113728.9
	0.50	1.00	98120.7
	0.00	1.00	97234.1
	0.00	1.50	85240.0
	0.00	2.00	79840.0
prim2	0.99	1.00	304058.7
	0.98	1.00	291532.9
	0.95	1.00	269495.2
	0.90	1.00	248388.0
	0.50	1.00	212068.8
	0.00	1.00	213276.0
	0.00	1.50	179340.0
	0.00	2.00	173300.0
r1	0.99	1.00	1284095.9
	0.98	1.00	1262461.9
	0.95	1.00	1218575.8
	0.90	1.00	1215419.9
	0.50	1.00	963928.4
	0.00	1.00	1099360.8
	0.00	1.50	789926.9
	0.00	2.00	780288.8
r3	0.99	1.00	3211281.5
	0.98	1.00	3125272.0
	0.95	1.00	2924382.0
	0.90	1.00	2707221.8
	0.50	1.00	2374080.0
	0.00	1.00	2197381.0
	0.00	1.50	2113469.5
	0.00	2.00	2025446.0

All bounds are normalized to the radius.

Table 3: Routing cost of LUBT for various other bounds

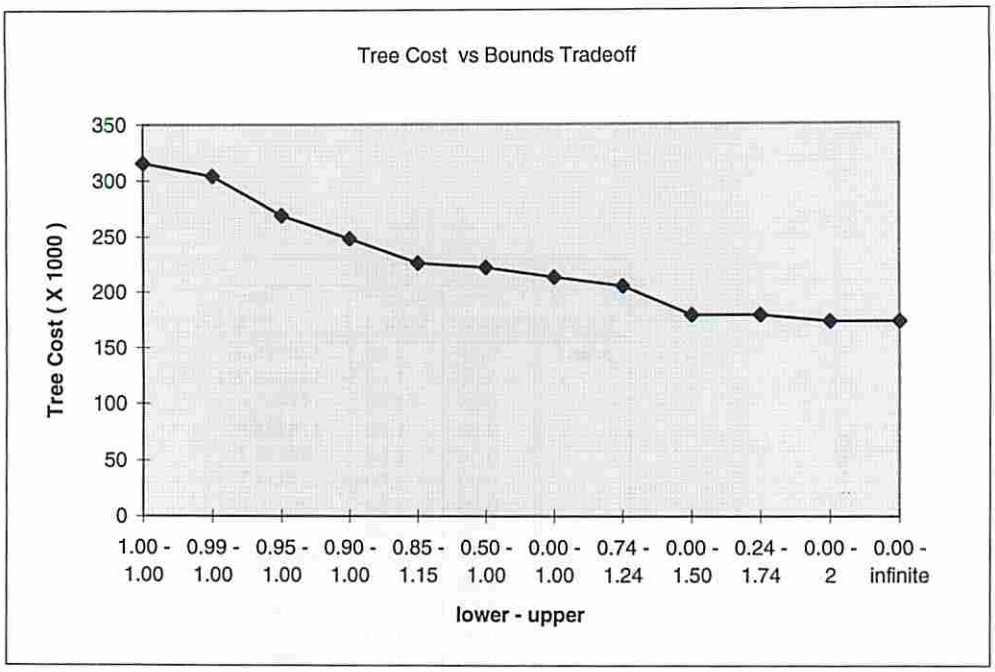

Figure 8: Trade off curve between the tree cost versus bounds for prim2

References

[1] Jingsheng Cong, Andrew B. Kahng, Gabriel Robins, Majid Sarrafzadeh, C. K. Wong, "Provably Good Performance-Driven Global Routing," *IEEE Transactions on Computer Aided Design* , Vol. 11, NO. 6, pp. 739-752, June, 1992.

[2] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh, "Clock routing for high-performance ICs," *27th Design Automation Conference* , pp. 573-579, 1990.

[3] A. Kahng, J. Cong, and G. Robins, "High-performance clock routing based on recursive geometric matching," *28th Design Automation Conference* , pp. 322-327, 1991.

[4] R-S Tsay, "Exact zero skew," *International Conference on Computer-Aided Design* , pp. 336-339, 1991.

[5] M. Edahiro, "A clustering-Based Optimization Algorithm in Zero-Skew Routing," *30th Design Automation Conference* , pp. 612-616, 1993.

[6] M. Borah, R. Owens and M. Irwin, "An edge-based heuristic for Steiner Routing," *IEEE Transactions on Computer Aided Design*, Vol. 13, No. 12, pp. 1563-1568, December 1994.

[7] Kenneth D. Boese and Adrew B. Kahng, "Zero-Skew Clock Routing Trees With Minimum Wirelength," *Proc. IEEE International Conference on ASIC*, pp. 1.1.1-1.1.5, 1992.

[8] Jason Cong and Cheng-Koh Koh, "Minimum-Cost Bounded-Skew Clock Routing," *International Symposium on Circuits and Systems*, pp. 215-218, 1995.

[9] Dennis J.-H. Huang, Andrew B. Kahng and Chung-Wen Albert Tsao, "On the Bounded-Skew Clock and Steiner Routing Problems," *Proc. ACM/IEEE Design Automation Conference*, pp. 508-513, 1995.

[10] Joe G Xi, Wayne W.M. Dai, "Buffer Insertion and Sizing Under Process Variations for Low Power Clock Distribution," *32nd Design Automation Conference*, pp. 491-496, 1995

[11] Robert J. Vanderbei, "LOQO User's Manual," *Program and the manual are available for free for academic users at ftp://elib.zib-berlin.de/pub/opt-net/software/loqo.*

[12] David G Luenberger, "Linear and Nonlinear Programming," *Addison-Wesley*, 1989.

10 Appendix

Proof of Theorem 4.1

To prove Theorem 4.1, some additional definitions and lemmas are needed.

Let T_k denote the subtree rooted at s_k. We define $pathlenght(s_x, s_y)$ as:

$$pathlength(s_x, s_y) = \sum_{e_k \in\ path(s_x, s_y)} e_k$$

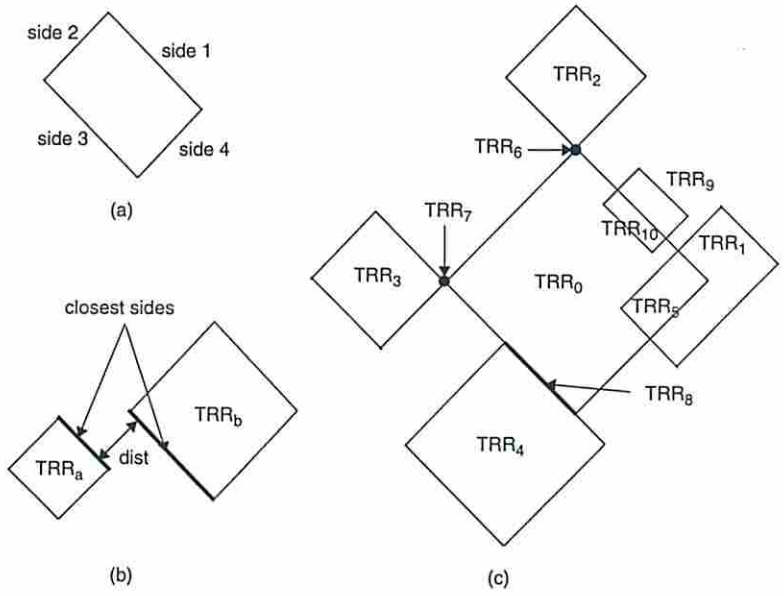

Figure 9: (a) Labeling of sides in TRR, (b) Various intersections of TRRs, (c) Definition of closest sides

A TRR has four sides. We will label them side1 through side4 starting from the side in the first quadrant and proceeding counterclockwise as shown in Figure 9 -(a). Even when a TRR is a line segment or a single point, we assume that it has four sides.

Two sides are said to be *overlapped* when they have the same slope and intersect. When a TRR_c is created from the intersection of two TRRs, we can uniquely determine which side of TRR_c overlaps with which TRR. In the Figure 9 -(c), sides2,3 of TRR_5 overlap with TRR_1 and sides1,4 of TRR_5 overlap with TRR_0. Also side1 of TRR_{10} overlaps with TRR_0 and sides2,3,4 of TRR_{10} overlap with TRR_9. Even when the intersection is a line segment or a point, the overlap relation is uniquely determined. For example, sides3,4 of TRR_8 overlap with TRR_0 and sides1,2 of TRR_8 overlap with TRR_4. Likewise sides1,2 of TRR_6 overlap with TRR_0 and sides3,4 of TRR_6 overlap with TRR_2. Also sides2,3 of TRR_7 overlap with TRR_0 and sides1,4 of TRR_7 overlap with TRR_3.

When two *TRR*s are separated, we can define the distance between the two *TRR*s as the minimum Manhattan distance between the two points belonging to their respective *TRR*s. Formally,

$$dist(TRR_i, TRR_j) = \text{MIN}\{dist(s_x, s_y) | s_x \in TRR_i, s_y \in TRR_j\}$$

Clearly, s_x and s_y are located at the boundaries of the *TRR*s when their distance is minimum. The two sides that have minimum distance from one another are called *closest sides* (see Figure 9 -(b). The closest sides can be found be extending one *TRR* until its boundary abuts the boundary of the other *TRR*. In the Figure 9 -(b), the closest side in TRR_a from TRR_b is side1. Likewise, the closest side in TRR_b from TRR_a is side3. Depending on the relative position of the two *TRR*s, there can be one or two closest side(s) of a *TRR* from the other *TRR*.

These overlap relationships and the definition of closest sides will be used in later subsections.

10.1 Feasible Regions Revisited

We consider feasible regions from a different point of view. As discussed in section 5, the feasible region of a Steiner point is found by intersecting the *TRR*'s of its two children. The feasible region is indeed the only possible locations for the Steiner point. So any location s_x in the Manhattan plane that satisfies

$$dist(s_i, s_x) \leq pathlength(s_i, s_k) \text{ for all sinks } s_i \in T_k$$

should be a feasible location for the Steiner point s_k. In other words, the feasible region of s_k is the intersection of square *TRR*s that are centered at sinks $s_i \in T_k$ and have radius equal to $pathlength(s_i, s_k)$. Formally,

$$FR_k = \bigcap_{s_i \in T_k} TRR(\{s_i\}, pathlength(s_i, s_k)) \tag{13}$$

where s_i's are sinks. For example, in Figure 6, FR_9 is the intersection of $TRR(\{s_4\}, e_4 + e_{10})$, $TRR(\{s_5\}, e_5 + e_{10})$, $TRR(\{s_6\}, e_6 + e_{11})$ and $TRR(\{s_7\}, e_7 + e_{11})$.

10.2 Intersection of TRRs

Intersection of *TRR*s has an important property as will be described in the following Lemma.

Lemma 10.1 *Suppose there are n TRRs. If pairwise intersection of every pair of TRRs are non- empty, then these TRRs have a common non-empty intersection.* [3].

Proof The case $n = 2$ is trivial. Suppose the Lemma holds for $n = k$. Let TRR_c be the common intersection of k TRRs. When $n = k + 1$, we introduce a new TRR_{k+1}. This must intersect with every TRR_i, $i = 1, \ldots, k$. Suppose however TRR_{k+1} does not intersect with TRR_c. Then we can find a TRR which does not intersect TRR_{k+1} as follows. Consider Figure 10 in which TRR_c and TRR_{k+1} are non-intersecting. In the figure, the closest side of TRR_c from TRR_{k+1} is side3. Then there is a TRR which includes TRR_c and its boundary overlaps with side3 (TRR_s in the figure). Clearly, TRR_s does not intersect with TRR_{k+1}. □

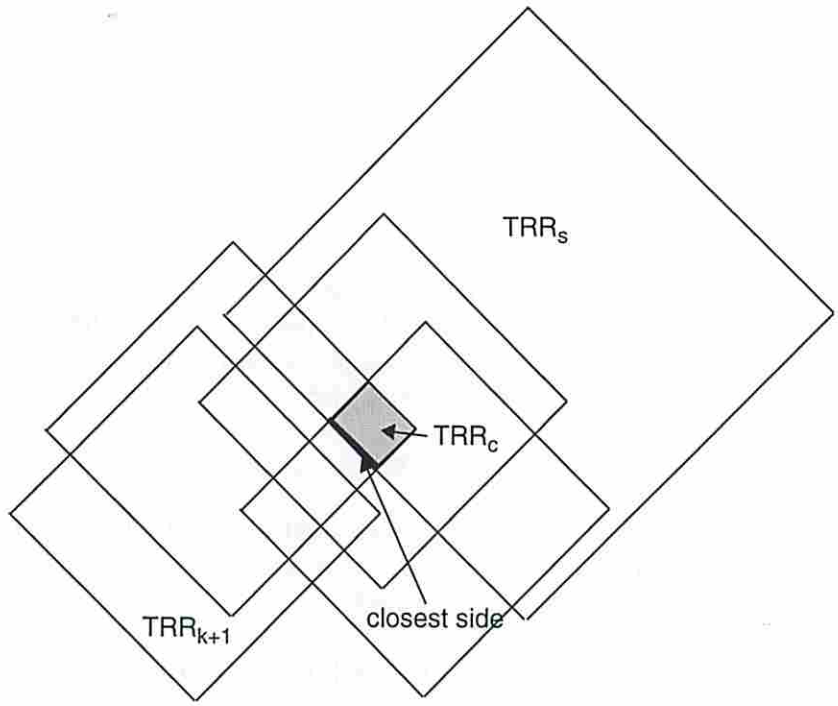

Figure 10: A counter example that TRR_{k+1} intersect with every other TRRs except the common intersection TRR_c.

[3]This lemma is not true in Euclidean space. One can easily draw three circles that have non-empty pairwise intersections but no common intersection. That is why the EBF method does not work in the Euclidean space.

10.3 Proof of Theorem 4.1 by contradiction

Suppose we cannot find the placements of Steiner points that satisfy Equation 7 for a set of edge lengths that satisfy Equation 6. Then while constructing feasible regions in a bottom up fashion, for some Steiner point s_k, its feasible region becomes empty. Note that the feasible region is the intersection of square TRRs defined in Equation 13. By Lemma 10.1, there should be a pair of square TRRs that do not intersect. Let the two square TRRs, $TRR(\{s_l\}, pathlength(s_l, s_k))$, $TRR(\{s_r\}, pathlength(s_r, s_k))$ be two such TRRs. Since they are separated, the sum of their radii is smaller than the Manhattan distance between the centers (i.e. the locations of sink s_l and s_r) of the two TRRs. That is,

$$pathlength(s_l, s_k) + pathlength(s_k, s_r) < dist(s_l, s_r)$$

or

$$pathlength(s_l, s_r) < dist(s_l, s_r)$$

contradicting the Steiner constraint given in Equation 6. This concludes the proof. □