Delay Bounded Minimum Steiner Tree Algorithms for Performance-Driven Routing

Qing Zhu Wayne W.M. Dai

UCSC-CRL-93-46
Oct. 10, 1993

Board of Studies in Computer Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064

ABSTRACT

The timing-driven routing of a critical net can be implemented as a delay-bounded minimum Steiner tree. This routing tree has the path delays always limited no larger than a delay bound D, while the total wire length is minimized. The delay bound is usually specified at the system/logic design stage. The routing maintains this delay bound to guarantee the correct timing of the finally implemented system in physical layout. We propose a new algorithm of constructing delay bounded minimum Steiner tree. This algorithm is designed based on the observation that this tree can be obtained based on the trade-off between two other special types of Steiner tree: (1) minimum Steiner tree and (2) minimum path delay Steiner tree, by taking into account of a delay bound D. We also proposed a new algorithm of constructing the minimum delay Steiner tree. During the tree growing, the Elmore delay is incrementally updated in constant time. These novel Steiner tree algorithms have been applied on a clock routing scheme for multi-chip modules based on area pad interconnections.

Keywords: Steiner tree, performance driven layout, delay-bounded Steiner tree, path delay, critical net

Contents

List of Figures

List of Tables

1 Introduction

While IC density is doubling roughly every 18 months, the feature size is steadily decreased. However, chip scale is steadily increased to incorporate more transistors and wirings. So, thinner and longer wire results in higher resistance of the interconnect. Also, some critical nets such as clock net are distributed to more loadings. The signal propagation delay through the interconnect becomes to dominate the timing delay of the whole digital system, especially for the off-chip communication. Indeed it has been reported that interconnection delay contributes up to 70% of the clock cycle in the design of dense, high-performance circuits. According to the ideal CMOS scaling rules, decrease device dimensions by a factor of $1/\Delta$ and increase the wiring area by a factor of α, then increase a global net delay by $\Delta^2 \cdot \alpha^2$ but decrease a gate delay by Δ [10]. Thus, interconnect delay has had an increasing impact on circuit speed. So, performance-driven layout has become critical to the design of high-performance digital systems.

Early work on performance driven layout focused on performance driven placement, with the usual objective being the close placement of cells in timing-critical paths as in [5, 11]. For the timing-driven routing issue, given a signal net, current existing methods appeared in the literature, majorly try to minimize the maximum signal delay from the source pin to any sink pin. Dunlop et al. [6] determined net priorities based on static timing analysis, and process higher priority nets earlier. A hierarchical approach to timing driven routing is outlined in [9]. Prastjutrakul and Kubitz first time constructed a routing tree based on A^* search to minimize the maximum path delay from source to sinks. Cong et al. [3] proposed a algorithm to construct radius-bounded Steiner trees with total wire length within a constant factor of optimal. This tradeoff of cost-radius was similarly formulated by [1]. Recently, Boese et al. [2] proposed methods to generate a class of Elmore delay routing tree constructions, which iteratively add tree edges to minimize Elmore delay from source to sinks.

In this paper, we propose a method of constructing the *delay bounded minimum Steiner tree*. The motivation behind this is that this kind of tree can be immediately applied on clock routing [17, 16]. Also, in leading-edge IC system design, system/logic design stage specifies the timing requirement of the physical nets. The timing requirement is usually set by the bound of the path delay through a net. Previous performance driven routing methods minimize the maximum path delay but results in the penalty of wiring area increase. Actually, in real design, people care about if the delay bound of the critical net is satisfied or not, then further hope to minimize the wiring area. We define the performance-driven routing of a critical net as a delay bounded minimum Steiner tree problem. In this type of tree, supposed a delay bound D is specified, then all path delays from the source to sinks are limited no larger than D. While the timing bound is meet, we still minimize the total wiring length of this tree.

While the *minimum path delay Steiner tree* achieves the minimization of maximum path delay and the *minimum Steiner tree* achieves the minimization of total wire length, the *delay bounded minimum Steiner tree* should be a trade-off between the above two extreme Steiner trees in terms of the largest path delay and total edge length, taking into account of a delay bound D. Based on this observation, we presented a new algorithm of constructing the delay bounded minimum Steiner tree.

This paper is organized as follows. In Section 2, we define the delay bounded minimum Steiner tree problem and the routing model we use. Then a new optimization algorithm

of delay bounded minimum Steiner tree is presented in Section 3. We also proposed an algorithm in Section 4 to obtain the minimum Elmore delay Steiner tree. During the tree growing, the Elmore delay is incremently updated in constant time. The time complexities of our Steiner tree algorithms are analyzed in Section 5. Experimental results in Section 6 compare the variations, in terms of total wire length and the longest path delay, among three special Steiner trees: (1) minimum Steiner tree, (2) minimum path delay Steiner tree, and (3) delay bounded minimum Steiner tree. Conclusions are summarized in Section 7.

2 Overview and Routing Model

Delay-Bounded Minimum Steiner Tree Problem: Given a net consisting of a source (o), a set of sinks, and a delay bound D, construct a Steiner tree T, in which the delay from o to every sink in T is less than D and the sum of the edge lengths of the tree is minimized.

The delay bound D for a minimum Steiner tree is specified by the timing requirement of this net. In [17, 16], a chip is partitioned into isochronous bins according to clock pin distribution. A local clock tree inside an isochronous bin is a delay-bounded minimum Steiner tree. The delay bound for a local clock tree is taken the deduction of the system tolerable skew to the skew on the MCM substrate.

The path delay evaluation depends on the delay model used. Some previous works have been done in constructing zeroth-order delay (path length) bounded Steiner tree [8] [3]. Boese et al. [2] proposed methods to generate a class of Elmore delay routing tree constructions, which iteratively add tree edges to minimize Elmore delay from source to sinks. Prasitjutrakul et al. [13] constructs the minimum delay routing tree based on the RC delay model in [15].

We propose a method of constructing the *Elmore delay bounded minimum Steiner tree* based on the trade-off of two special trees: (1) *minimum path delay Steiner tree* and (2) *minimum Steiner tree*. Since the routing in the IC chip is usually constructed in Manhattan wirings, we use *Hanan grid* [7] to construct rectilinear Steiner trees. Hanan grid is derived by extending a horizontal line and a vertical line through each sink and the source, as shown in Figure 2.1. A *node* in Hanan grid is the intersection of a vertical line and a horizontal line; a *edge* intervenes two nodes. Some of the nodes are the locations of source and sinks.

A routing tree T in a Hanan grid can be modeled as an RC tree. Every edge in the routing tree is modeled as an RC line as shown in Figure 2.2(a). For an edge e_i, we have edge resistance $r_i = r_s l_i$, and edge capacitance $c_i = c_s l_i$, where r_s is unit length resistance and c_s the unit length capacitance, and l_i the edge length. Each sink has a loading capacitance as shown in Figure 2.2(b). The signal input at source o is sent out by a driver with an output resistance r_d and an output capacitance c_d as shown in Figure 2.2(c). The Elmore delay $t(s_j)$ from source o to a sink s_j can be calculated as follows [14, 2].

$$t(s_j) = r_d(c_d + C_0) + \sum_{e_i \in path(o,s_j)} r_i(c_i/2 + C_i) \tag{2.1}$$

where e_i is the edge from node n_i to its parent. C_0 is the total capacitance of sinks and edges of the routing tree T. C_i is the total capacitance of sinks and edges in the subtree of T rooted at n_i.

Notations of the Steiner trees used in this paper are listed as follows.

are deleted from T_b (also deleted from G_m). So, there is no min-delay edge or mixed edge outside T_b. It follows that p_c is a min-length path which connects $T_b{}^1$ and $T_b{}^2$. □

If a min-length cut path p_c is added to T_b between $T_b{}^1$ and $T_b{}^2$, T_b becomes connected again. p_c is called a *feasible* min-length cut path of T_b, if this path satisfies the following two conditions: (1) this path is shorter than the path length of min-delay simple path which is just deleted from T_b; (2) After adding p_c, T_b still has path delays bounded by D. Among those feasible min-length cut paths, the algorithm selects the one with the shortest length resulting in the least total edge length. We thus obtain the reroute rule for the delay-bounded Steiner tree.

Reroute Rule: *at each stage, always add in T_b the feasible min-length cut path with the minimal path length.*

Our algorithm improve T_b by iteratively applying the above Rip-Up and Reroute rules. Note that at some stages, we may find no feasible min-length cut path exists after a min-delay simple path is deleted. In this case, we restore this min-delay simple path back to T_b and continue to process other min-delay simple paths. If a feasible min-length cut path is successfully added to T_b, edges along this path are marked as mixed edges. The iteration continues until no longest min-delay simple path exists.

Shown in Figure 3.1(a), T_d has total edge length $l_d = 420\mu m$ and the largest path delay $t_d = 0.84ps$; T_c has $l_c = 309\mu m$ and $t_c = 1.09ps$. Figure 3.3 illustrates this algorithm for this example to construct T_b where delay bound is $D = 0.9ps$. T_b starts from T_d. Shown in Figure 3.3(a), a longest simple path s_1s_5 has been deleted from T_b and G_m, but a min-length path s_1s_2 is added to T_b. In Figure 3.3(b), the longest min-delay simple path v_1s_5 is replaced by a min-length path v_1v_2. In Figure 3.3(c), min-delay simple path v_9v_{11} is replaced by $v_{10}v_{11}$. In Figure 3.3(d), after min-delay path v_4s_4 is replaced by a min-length path v_3s_4, path ov_4v_2 becomes the longest min-delay path. But min-length path v_6s_4 is not a *feasible* min-length path, since the largest path delay of T_b will exceed D if path v_6s_4 is added to replace path ov_4v_2. So, path ov_4v_2 is still remained in T_b. The final T_b is shown in Figure 3.3(e), which has the total wire length $l_b = 309\mu m$ and the largest path delay $t_b = 0.87ps$. t_b is less than the delay bound $D = 0.9ps$, while l_b is decreased 27% of l_d which achieves the same total wire length of T_c for this case. The delay bounded Steiner tree depends on the specified delay bound D. If we set a tighter delay bound $D = 0.85ps$ for this example, the final T_b is shown in Figure 3.3(f), resulting in $l_b = 340\mu m$ and $d_b = 0.85ps$.

The delay bounded Steiner tree algorithm is summaried in the following pseudo-code.

Elmore delay bounded Steiner tree algorithm

Input: a signal net N with a source o and a set of sinks, a delay bound D;
Output: a delay bounded Steiner tree T_b

```
Procedure DelayBoundedSteinerTree(N, D, T_b) {
    T_d = Minimum Elmore delay Steiner tree over N on Hanan grid;
    T_c = Minimum edge length Steiner tree over N on Hanan grid;
    G_m = T_d + T_c;
    T_b = T_d;
    p_d = the longest min-delay simple path in T_b;
    while (p_d ≠ NULL) {
        Delete p_d from T_b by marking edges as non-tree edges;
```

p_c = the shortest feasible min-length cut path connecting two disjoint subtrees of T_b;
if ($p_c \neq NULL$)
Add p_c to T_b by marking edges as mixed edges;
else
Add p_d to T_b by marking edges as mixed edges;
p_d = the longest min-delay simple path in T_b;
}
}

The final T_b has tree edges with only one type of mixed edge. The next theorem shows the correctness of this algorithm.

Theorem 1: *The total edge length of T_b is monotonically decreased as the iterative tree optimization proceeds, while the largest path delay of T_b is always bounded by the delay bound D if D is specified not less than the largest path delay of T_d.*

Proof: We use the mathematical induction according to the series of iterative update of T_b with configurations ${T_b}^0, {T_b}^1, \ldots, {T_b}^n$.

The basis, at the first stage ${T_b}^0 = T_d$. Since D is not less than the largest path delay of T_d, ${T_b}^0$ has its largest path delay $t_b \leq D$.

The inductive step, T_b^i is updated to T_b^{i+1} after rip-up and reroute. Suppose that T_b^i has total edge length l_b^i and the largest path delay $d_b^i \leq D$. Based on Rip-Up rule, a longest min-delay simple path p_d is deleted from T_b^i. Based on Reroute rule, there are two cases: (1) a feasible min-length cut path p_c is found and p_c is added to T_b^{i+1}; (2) no such a p_c is found and thus p_d is added back to T_b^{i+1}. In Case 1, based on definition of a feasible min-length cut path, p_c has less path length than p_d and T_b^{i+1} still has bounded path delays. So, T_b^{i+1} has less total edge length than T_b^i since other edges keep the same as T_b^i, and also the largest path delay is still bounded. In Case 2, $T_b^{i+1} = T_b^i$ with no update of T_b. □

4 Minimum Steiner Tree Algorithms

The delay bounded Steiner tree is derived based on the mixture of minimum edge length Steiner tree T_c and minimum Elmore delay Steiner tree T_d. Obtaining T_c is an NP complete problem. We employ an efficient method in [12] of constructing an approximation of T_c. This method is a single partial tree growth algorithm each time a sink is connected. The partial tree starts at the source o and repeatedly calling Dijistra's shortest path algorithm to grow the partial tree to connect an unconnected sink. T_c is obtained until all sinks are connected. Shown in Figure 3.1(b), T_c is obtained by using this method.

We apply the single partial tree growth method on the construction of the approximation of minimum Elmore delay Steiner tree T_d. Instead of total edge length minimization for T_c, we approach to Elmore-delay minimization for T_d. The Elmore delay calculation, shown in (2.1), is *naturally* incorporated into T_d growing process.

To connect a sink, the algorithm expands wave frontier of a partial tree in a breadth first search until an unconnected sink is encounted. To keep track of the progress, the algorithm colors each vertex *white*, *gray*, or *black*. All nodes in Hanan grid start out white and may later become gray and then black. A node becomes gray when it is *discovered* the first time it is encountered during the search. A gray node becomes darken when it is expanded to neighbors. Gray nodes represent the wave frontier between discovered and undiscovered

nodes. We have a priority queue Q to store all gray nodes at current stage. The priority of a gray node is the Elmore delay estimate from the source to this node along the search tree. Everytime we extract a gray node u from Q which has the least Elmore delay estimate among all current gray nodes. u is then expanded to neighbors. Whenever u is expanded to neighbor v, u is the parent of v in the breadth-first search. Let $d[u]$ be the the *path delay estimation* of u from source o in the search tree, and $r[u]$ the sum of driver resistance r_d and all resistances of ancestor edges of u in the tree. Suppose node u is expanded to v, based on Elmore delay formula in (2.1), we can update $d[v]$ and $r[v]$ incrementally as

$$d[v] = \begin{cases} d[u] + (r[u] + \frac{r_1}{2})c_1 & \text{if } v \text{ is not a sink} \\ d[u] + (r[u] + \frac{r_1}{2})c_1 + (r[u] + r_1)c_t & \text{if } v \text{ is a sink with loading capacitance } c_t \end{cases} \tag{4.1}$$

and

$$r[v] = r[u] + r_1 \tag{4.2}$$

where r_1 and c_1 are resistance and capacitance of edge $\overline{uv}$. (4.1) can be calculated in constant time resulting in no order increase of computational time during expansion. If v previously has been a gray node, we assign it the smaller one of new $d[v]$ based on (4.1) and previous $d[v]$.

The search continues until an unconnected sink is encounted. At that time, starting from the sink, re-traversing the path from the sink to the partial tree by using the recursive relation of parent node. Edges in this path are marked as min-delay edges and this sink is thus connected. After a sink is connected and if there are still some sinks are unconnected, we *reset* all nodes in the partial tree as gray nodes, and other nodes on Hanan grid as white nodes. Starting from the new partial tree, the algorithm expands the wave frontier again until next unconnected sink is encountered. This partial tree expansion and growing process continues until all sinks are connected.

The next Theorem 2 shows the optimality of the choice of the next sink to be connected to the existing tree. We first prove the next lemma that shows the monotone increase of path delay estimation during the search expansion.

Lemma 1: *When a gray node u is extract from the queue Q, none of at present gray and white nodes will become have less path delay estimate than $d[u]$ in the following expansion.*

Proof: At the time of u extracted from the queue Q, u should have the least path delay estimate $d[u]$ among gray nodes in Q. In other word, any gray node v has $d[v] \geq d[u]$. If w is gray or white node at the time u is extracted, We can show that $d[w] \geq d[u]$ forever after u is extracted. In the continuing search expansion, suppose node w is encountered and its $d[w]$ is updated. w should be expanded from some node v which is a gray node when u is extracted from Q, since only gray nodes are expanded to neighbors. So, v is an ancestor node of w in the path from source o to w (see Figure 4.1) resulting in $d[w] \geq d[v]$. But we know that $d[v] \geq d[u]$, such that $d[w] \geq d[u]$. □

Theorem 2: *The algorithm always connects the next new sink with the minimal path delay from source to unconnected sinks.*

Proof: When a sink is first time extracted from Q, it has the least path delay among all unconnected sinks. According to Lemma 1, other unconnected sinks can never have less path delays than the first extracted unconnected sink even if the search expansion continues. So,

$O(k^3 \lg k)$ to connect k sinks. So, it is showed that the minimum Elmore delay Steiner tree algorithm has the same order of the time complexity of the minimum length Steiner tree algorithm in [12].

Suppose there are E_c edges of T_c and E_d edges of T_d. T_b is the mixture of T_c and T_d, such that it has edge number $E_b \leq E_c + E_d$. The delay bounded Steiner tree algorithm takes, in the worst case, $O(E_d E_c E_b) = O(E_d E_c max(E_c, E_d)$ time complexity. The analysis is as follows. There are $O(E_d)$ min-delay simple paths. When a min-delay simple path is rip-uped, it takes $O(\lg E_d)$ to select the one with the longest path length, if we store these min-delay simple paths is a binary heap. There are $O(E_c)$ min-length cut paths which may be rerouted, each time it takes $O(\lg E_c)$ to select the one with the shortest path length. When a min-length cut path is selected, it needs to be checked if it is a feasible cut path by calculating Elmore delays to sinks in T_b. The Elmore delay calculation takes $O(E_b)$. So, the total time complexity of this algorithm is $O(E_d(\lg E_d + E_c(\lg E_c + E_b))) = O(E_d E_c E_b) = O(E_d E_c max(E_d, E_c))$.

6 Experiment Results

The delay bounded Steiner tree algorithm for local clock routing has been implemented in ANSI C. This algorithm has been tested both on random sink distribution and large benchmarks. We take the electrical parameters of chip in [2], and electrical parameters of a advanced thin-film MCM substrate in [18]. These parameters are listed in Table 6.1. We size the driver at clock area pad resulting in two output resistances $R_d = 100\Omega$ and $R_d = 50\Omega$ to test the effect of driver sizing on the Steiner tree construction.

	$R_b(m\Omega/\mu m)$	$C_b(fF/\mu m)$	$L_b(pH/\mu m)$	$R_d(\Omega)$	$C_t(pF)$
Chip	30	0.352	0	100, 50	0.0153
MCM	8	0.06	0.38	25	0.2

Table 6.1: Electrical parameters. R_b, C_b and L_b are resistance, capacitance and inductance of unit length wire. R_d is the driver output resistance, and C_t is the loading capacitance of a terminal. The clock driver in chip (die) is sized with two output resistances.

Table 6.2 and Table 6.3 show the comparison of three kinds of Steiner trees T_d, T_c and T_b for the examples of random distribution of 8, 16, 32 and 64 sinks. In these two tables, t_d, t_b, t_c are the largest path delays of T_d, T_b and T_c respectively, while l_c, l_b and l_d are the total edge lengths of T_c, T_b and T_d respectively . D is the delay bound for T_b, which is selected between t_d and t_c. Table 6.2 is obtained based on $R_d = 100\Omega$, and Table 6.3 based on $R_d = 50\Omega$. The largest path delays are obviously shortened in Table 6.3 because of the smaller R_d compared to Table 6.2.

On the average, the minimum length Steiner tree T_c is 30% less total edge length than minimum path delay Steiner tree T_d, while T_d is 78% less largest path delay than T_c. If T_c and T_d are optimum, T_b should have less total edge length than T_d and more total edge length than T_c. But, since T_c is obtained based on a approximation algorithm [12], it is showed in Table 6.2 and Table 6.3 that l_b is less than l_c in most examples. For all these examples, $t_b \leq D$ is satisfied.

	Largest Path Delay (ns)			Total Edge Length (mm)		
	t_d	t_b	t_c	l_c	l_b	l_d
8 sinks	4.8	4.8 (D = 5.5ns)	7.0	61.8	63.9 (D = 5.5ns)	79.8
16 sinks	5.05	5.06 (D = 5.06ns)	5.07	70.4	58.6 (D = 5.06ns)	83.7
32 sinks	6.8	6.86 (D = 6.9ns)	7.0	91.0	71.1 (D = 6.9ns)	131.0
64 sinks	8.2	10.97 (D = 13.4ns)	23.7	122.5	114.1 (D = 13.4ns)	163.2

Table 6.2: Comparison of Steiner tree results on random distribution of 8, 16, 32 and 64 sinks. The driver output resistance $R_d = 100\Omega$. t_d, t_b and t_c are the largest path delays of T_d, T_b and T_c respectively . l_c, l_b and l_d are the total wire lengths of T_c, T_b and T_d respectively . Sinks are randomly distributed in a chip size of 20 x 20 mm.

	Largest Path Delay (ns)			Total Edge Length (mm)		
	t_d	t_b	t_c	l_c	l_b	l_d
8 sinks	3.5	3.6 (D = 4.3ns)	5.9	61.8	58.3 (D = 4.3ns)	75.0
16 sinks	3.6	3.66 (D = 3.7ns)	3.9	70.4	59.5 (D = 3.7ns)	82.6
32 sinks	4.0	4.23 (D = 4.4ns)	5.2	91.0	89.4 (D = 4.4ns)	127.5
64 sinks	5.2	5.4 (D = 10.7ns)	21.8	122.5	117.7 (D = 10.7ns)	171.6

Table 6.3: Comparison of Steiner tree results when the driver output resistance $R_d = 50(\Omega)$.

7 Conclusion

We formulate a performance driven critical net routing as a *delay bounded Steiner tree* problem. The delay bound is usually specified at the system/logic design stage. The layout maintains this delay bound to guarantee the correct timing of the finally implemented system in physical layout.

We propose a new algorithm of constructing delay bounded Steiner tree. This algorithm is based on the observation that this tree can be obtained based on the trade-off between two extreme Steiner trees: (1) minimum Steiner tree and (2) minimum path delay Steiner tree, taking into account of a delay bound D. The resultant tree has the path delay always bounded by D.

This Steiner tree algorithm has been applied on the clock routing for multi-chip modules based on area pad interconnection [17, 16].

This algorithm can also be extended to arbitrary-angle grids instead of Hanan (rectilinear) grids. We expect this Steiner tree algorithm can be applied on critical nets routing for high-performance layout.

8 Acknowledgement

This work was supported partially by Intel Corporation and partially by the National Science Foundation Presidential Young Investigator Award under Grant MIP-9009945. We want to thank Prof. Andrew Kahng and Kenneth Boese of UCLA for providing the source codes of their Steiner tree algorithms in [2].

References

[1] C.J. Alpert, T.C. Hu, J.H. Huang, and A.B. Kahng. A direct combination of the prim and dijkstra constructions for improved performance-driven global routing. *Technical Report, CSD-TR-920051, Computer Science Department, UCLA*, 1992.

[2] K. D. Boese, A. B. Kahng, and G. Robins. High-performance routing trees with identified critical sinks. In *Proc. of 30th Design Automation Conf.*, pages 182–187, 1993.

[3] J. Cong, A.B. Kahng, G. Robins, M. Sarrafzadeh, and C.K. Wong. Provably good performance-driven global routing. *IEEE Trans. on Computer-Aided Design*, CAD-11(6):739–752, 1992.

[4] Jason Cong, Kwok-Shing Leung, and Dian Zhou. Performance-driven interconnect design based on distributed rc delay model. In *Technical Report, University of California, Los Angeles*, pages 1–36, 1992.

[5] W.E. Donath, R.J. Norman, B.K. Agrawal, S.E. Bello, S.Y. Han, J.M. Kurtzberg, P. Lowy, and R.I. McMillan. Timing driven placement using complete path delays. In *Proceedings of ACM/IEEEE Design Automation Conference*, pages 84–89, 1990.

[6] A.E. Dunlop, V.D. Agrawal, D. Deutsch, M.F. Jukl, P. Kozak, and M. Wiesel. Chip layout optimization using critical path weighting. In *Proceedings of ACM/IEEEE Design Automation Conference*, pages 133–136, 1984.

[7] M. Hanan. On steiner's problem with rectilinear distance. *SIAM Journal of Applied Mathematics*, pages 255–265, March 1966.

[8] J. Ho, D. T. Lee, C. H. Chang, and C. K. Wong. Bounded-diameter spanning tree and related problems. In *Proc. ACM Symp. on Computational Geometry*, pages 276–282, 1989.

[9] Xianlong Hong, Tianxiong Xue, Ernest S. Kuh, Chung-Kuan Cheng, and Jin Huang. Performance-driven steiner tree algorithms for global routing. In *Proceedings of ACM/IEEEE Design Automation Conference*, 1993.

[10] M.A.B. Jackson, E.S. Kuh, and M. Marek-Sadowska. Timing-driven routing for building block layout. In *Proceedings of IEEE Intl. Symposium on Circuits and Systems*, pages 518–519, 1987.

[11] K.C.Saraswat and F. Mohammadi. Effects of scaling of interconnections on the time delay of vlsi circuits. *IEEE Journal of Solid State Circuits*, SC-17:275–280, 1982.

[12] I. Lin and D.H.C. Du. Performance-driven constructive placement. In *Proceedings of ACM/IEEEE Design Automation Conference*, pages 103–106, 1990.

[13] T. Ohtsuki. *Layout Design and Verification, Advances in CAD for VLSI, Vol. 4*. North-Holland, 1986.

[14] Somchai Prasitjutrakul and William J. Kubitz. A timing-driven global router for custom chip design. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 48–51, 1990.

[15] J. Rubinstein, P. Penfield, and M. A. Horowitz. Signal delay in rc tree networks. *IEEE Trans. on Computer-Aided Design*, CAD-2(3):202–211, 1983.

[16] T. Sakurai. Approximation of wiring delay in mosfet lsi. *IEEE Journal of Solid State Circuits*, SC-18:418–426, 1983.

[17] Qing Zhu and Wayne W.M. Dai. Hierarchical clock distribution using area pad interconnection in multi-chip modules. *submited to 31th Design Automation Conf.*, 1993.

[18] Qing Zhu and Wayne W.M. Dai. Hierarchical clock routing for multi-chip modules based on area pad interconnection. *Technical Report, UCSC-CRL-93, University of California, Santa Cruz.*, 1993.

[19] Qing Zhu, Wayne W.M. Dai, and Joe G. Xi. Optimal sizing of high speed clock networks based on distributed and lossy transmission line models. *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 628–633, Nov. 1993.