STEINER TREE OPTIMIZATION FOR BUFFERS, BLOCKAGES, AND BAYS

Charles J. Alpert*, Gopal Gandham†, Jiang Hu‡, Jose L. Neves†
Stephen T. Quay† and Sachin S. Sapatnekar‡

*IBM Austin Research Lab, †IBM Microelectronics, ‡Department of ECE, University of Minnesota

ABSTRACT

Buffer insertion is essential for achieving timing closure. This work studies buffer insertion under two types of constraints: (i) avoiding blockages, and (ii) inserting buffers into pre-determined buffer bay regions. We propose a general Steiner tree routing problem to drive this application and present a maze-routing based heuristic. We show that this approach leads to useful solutions on industry designs.

1. INTRODUCTION

Buffer insertion has become a critical step in modern VLSI design methodologies (see [2] for a survey). Several works have studied the delay driven-buffer insertion problem. Van Ginneken's algorithm [11] has become a classic in the field. His dynamic programming algorithm finds the optimal buffer placement under the Elmore delay model. Several extensions to this work have been proposed (e.g., [1] [6] [7] [9]). Most works assume that a Steiner tree is given and that buffers must be placed along the Steiner wires. The works of [7] [9] also perform routing of the tree during buffer insertion but do not consider blockages.

When attempting to insert buffers into a floorplanned design, buffers may not be placed on top of pre-existing macros; these regions are called blockages. If the Steiner tree completely spans blockages, then any buffer insertion algorithm which uses the routing topology will fail to find a solution. Figure 1(a) shows an example 2-pin net whose route runs over a large blockage, thereby making buffer insertion infeasible. If one re-routes the tree as in Figure 1(b), then buffers can be inserted, albeit for an additional wire length cost. Here, the Steiner tree serves as a guide for buffer insertion, but does not necessarily represent the final route. Figure 1(c) shows how the global router may re-route the newly created nets while considering delay, noise, congestion, etc. This helps ensure that regions of the chip without blockages do not become unnecessarily congested with interconnect.

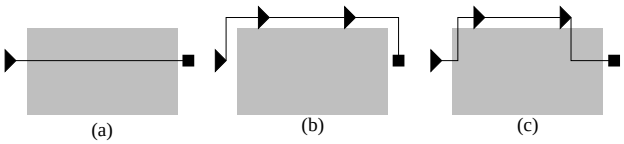

Figure 1: Rerouting example.

Figure 2 illustrates an example where the best Steiner tree construction avoids some, but not all, of the blockages. In (a), the existing route is completely blocked for buffering, while in (b), the re-routed tree avoids all blockage, allowing buffers to be inserted. However, the most efficient solution is shown in (c) which avoids only some of the blockage. The problem of buffer insertion in the presence of blockage constraints has been addressed in [5] and [12]. The approach of [12] also allows routing over some blockages while avoiding others. However, the algorithm is only applicable to two-pin nets.

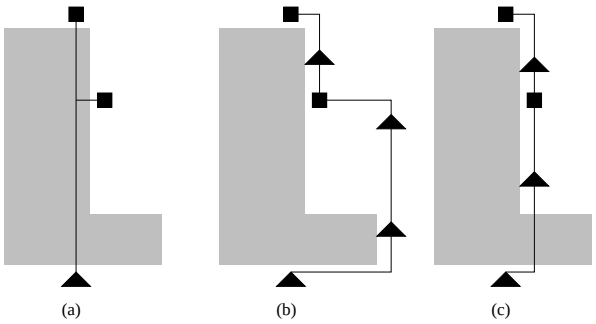

Figure 2: Avoiding some blockage yields the best solution.

In some design methodologies, it may be suitable to pre-allocate space for buffers during floorplanning, rather than trying to squeeze buffers between large blocks during physical design, which can cause both logical and wiring congestion. We call these pre-allocated regions buffer bays. In this scenario, the entire layout area is viewed as blockage except for the buffer bays. Figure 3(a) shows an example of a two-pin net that does not cross any buffer bays and is thus totally blocked from buffer insertion. By re-routing the tree through a buffer bay (b), buffers can be suitably inserted (c).

Figure 3: A blocked tree (a) can be re-routed through a buffer bay (b) which enables buffer insertion (c).

We propose a general Steiner tree problem formulation for buffer insertion with either blockage or buffer bay constraints and present a new Steiner tree optimization for this problem. The algorithm iteratively rips up a sub-path of an existing Steiner tree and uses maze routing to re-connect the two remaining sub-trees. In contrast to [5] [9] [12], which simultaneously insert buffers during routing, we first construct the Steiner tree, then inserts buffers. The simultaneous approach is arguably superior considering that one cannot design the best tree until buffer locations are known. However, the simultaneous operations of tree construction and buffer insertion necessitate that the buffering component be somewhat simplistic. Indeed, no efficient and effective simultaneous heuristic for multi-sink nets has ever been proposed in the literature.

2. PROBLEM FORMULATION

Given a unique source so and a set of sinks SI, a rectilinear Steiner tree (RST) $T(V,E)$ is a spanning tree in the rectilinear plane that connects every node in $V = \{so\} \cup SI \cup W$, where W is a set of additional nodes. W typically includes two types of nodes: (i) internal Steiner nodes of degree three or four, denoted by the set IN, and (ii) corner nodes of degree two that connect a horizontal and vertical edge, denoted by the set CO. We add a third node type to W: a boundary node (belonging to the set BY) has degree two, an incident edge lying over blockage, and an incident lying in a blockage-free region. For example, the RST in Figure 4 shows a Steiner tree with source $so = s$ and sinks $SI = \{d, i, k\}$. All other nodes are in W with $b \in IN$, $g, j \in CO$, and $a, c, e, f, h \in BY$.

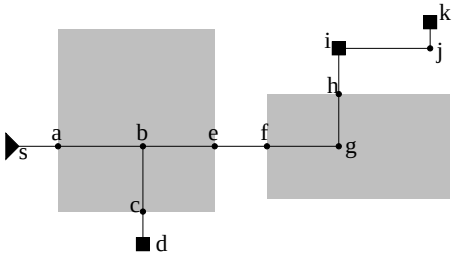

Figure 4: A Steiner tree illustrating different node types.

Definition: A 2-path of a tree $T(V,E)$ is a path $p(u,v) = \{(u,v_1),(v_1,v_2),\ldots,(v_m,v)\} \in T$ such that $\{v_1,\ldots,v_m\} \subseteq BY \cup CO$ and $u, v \in \{so\} \cup SI \cup IN$.
Every tree T can be uniquely decomposed into a set of 2-paths, e.g., the tree in Figure 4 can be decomposed into four 2-paths: $p(s,b), p(b,d), p(b,i)$ and $p(i,k)$.

A rectangle r has a unique bounding box $(x_1,y_1),(x_2,y_2)$, where $x_1 \le x_2$ and $y_1 \le y_2$. Given a set of rectangles B (i.e., the blockage map), an edge $e \in E$ is inside B (denoted by $e \in B$) if there exists a rectangle $r \in B$ such that both endpoints of e lie inside the bounding box of r. Let l_e denote the length of edge e.
Problem Formulation: Given a parameter α, a source so, a set of sinks SI, and a set of rectangular blockages B, construct a Steiner tree $T(V,E)$ with $\{so\} \cup SI \subseteq V$ that minimizes

$$cost(T(V,E)) = \sum_{e \in E} l_e + \alpha \sum_{e \in B} l_e \qquad (1)$$

The parameter α represents the degree of the penalty for routing over blockage. This problem is NP-Complete by reduction to the Rectilinear Minimal Steiner tree problem (when $\alpha = 0$). Observe $cost(T(V,E))$ can be expressed as the sum of the costs of all 2-paths in T, where the cost of a 2-path is given by:

$$cost(p(u,v)) = \sum_{e \in p(u,v)} l_e + \alpha \sum_{e \in B \cap p(u,v)} l_e \qquad (2)$$

For example, if $\alpha = 1$, then edges that intersect blockage have twice the cost of the other edges. Recall the re-route in Figure 1. If the wire length more than doubles when changing the route from (a) to (b), then (a) is the lower cost solution. The appropriate value for α depends on the technology, though empirically a value of one seems to work well. An advantage of this cost function is that it can be used to handle both buffer bays and blockages. If B represents a set of buffer bays, then routing over rectangles in B should actually reduce the cost function. Using α between -1 and 0 achieves this effect, e.g., if $\alpha = -\frac{1}{2}$, then the cost of routing outside a buffer bay is twice that of routing inside a buffer bay.

3. THE EXTENDED HANAN GRAPH (EHG)

Our Steiner tree heuristic is based on maze routing, which has a fundamental concept of a grid graph, $G(V_G, E_G)$. A grid graph can be viewed as a tessellation of rectangular tiles with V_G being the set of tile centers and E_G being edges that connect tile centers. A grid graph can be uniquely induced by the sets $X = \{x_1,\ldots,x_n\}$ and $Y = \{y_1,\ldots,y_m\}$ of sorted non-duplicate coordinates. The induced grid graph $G(V_G,E_G)$ from X and Y has vertices $V_G = \{(x,y) \mid x \in X, y \in Y\}$, and edges $E_G = \{((x_i,y),(x_{i+1},y)) \mid 1 \le i < |X|, y \in Y\} \cup \{((x,y_i),(x,y_{i+1})) \mid 1 \le i < |Y|, x \in X\}$. We do not use a uniform grid graph since sink and blockage distributions are non-uniform. Our grid graph allows high density channels in difficult routing areas and low density channels elsewhere. Assume that some low-cost RC tree T has already been computed over $\{so\} \cup SI$. Our grid graph is a superset of the Hanan grid [3] for T. The algorithm is shown below.

Procedure Grid_graph(T, B)
Input: Steiner tree $T(V,E)$, set of rectangles B
Output: Grid graph $G(V_G,E_G)$
1. Set $X = \emptyset, Y = \emptyset$.
2. For each $v \in V$ with coordinates (x,y), $X \leftarrow x \cup X$, $Y \leftarrow y \cup Y$.
3. For each $r \in B$ with bounding box $(x_1,y_1),(x_2,y_2)$
If r is a blockage
$X \leftarrow (x_1 - M) \cup (x_2 + M) \cup X$,
$Y \leftarrow (y_1 - M) \cup (y_2 + M) \cup Y$.
If r is a buffer bay
$X \leftarrow (x_1 + M) \cup (x_2 - M) \cup X$,
$Y \leftarrow (y_1 + M) \cup (y_2 - M) \cup Y$.
4. Sort the coordinates in X and Y
5. Generate induced grid graph $G(V_G,E_G)$ from X,Y.
6. $\forall e \in E_G$, compute value of $blocked(e)$.

Step 1 initializes sets X and Y to be empty, and Step 2 adds the coordinates of each tree node into X and Y. Step 3 adds the coordinates of the blockages (where M is the minimum separation between a buffer and a blockage), and Steps 4-5 construct the grid graph induced by X and Y. Finally, Step 6 sets the attribute $blocked(e)$ for each edge e in G. If e overlaps with a blockage in B or does not overlap with a buffer bay in B, then the attribute is set to $true$; otherwise, it is set to $false$.

4. ALGORITHM DESCRIPTION

Our algorithm first decomposes the existing Steiner tree into disjoint 2-paths and computes each 2-path cost. It then iteratively chooses a poorly routed 2-path, removes it, and re-routes it. The 2-path with the highest cost is not necessarily the most poorly routed path, as the highest cost path could simply be a very long path. We identify poorly routed 2-paths with the highest value of $cost(p(u,v))/l_{p(u,v)}$. Such a 2-path has the highest ratio of wire length routed over blockage to total wire length which implies room for improvement.

A complete description of the algorithm is shown below. Step 1 computes the underlying grid graph for T and B. Step 2 finds the set of all 2-paths, and Steps 3 and 4 iterate through these 2-paths, each time picking the one with the highest overlap cost. The selected 2-path is removed in step 5, which induces two subtrees T_s and T_t. Step 6 performs the maze routing which returns a minimum cost 2-path between T_s and T_t, and Step 7 re-connects the tree using this 2-path. We now explain the maze routing performed in Step 6.

Steiner_Tree Algorithm (T, B)
Input: $T(V,E)$, a Steiner routing tree B, rectangles representing blockages or bays
Output: Re-routed Steiner tree T
1. $G(V_G, E_G) =$ Grid_graph(T, B).
2. Compute the set P of disjoint 2-paths in T. Compute cost of each 2-path in P from Equation (2).
3. While $P \neq \emptyset$
4. Choose $p(u,v) \in P$ with max $cost(p(u,v))/l_{p(u,v)}$.
5. Remove $p(u,v)$ from T and P to get two sub-trees. Label sub-tree containing so as T_s; the other is T_t.
6. Find 2-path $p(q,w) =$ Maze_Routing(G, T_s, T_t).
7. Add the edges in the 2-path $p(q,w)$ to T.

The path re-connecting two subtrees is found via maze routing [8]. Each grid edge is assigned a cost, e.g., edge length for unblocked and infinity for blocked edges. The source node is initially assigned zero cost, and then wave expansion proceeds out from the source, labeling all intermediate nodes until the target node is reached. The grid node labels reflect the routing cost from the source. For a linear cost function, maze routing guarantees the least cost path for connecting two points. The primary variation of our algorithm is we wish to find the lowest cost path between subtrees as opposed to unique points. This is accomplished by labeling all nodes in the source tree with zero cost and stopping when any node in the target tree is reached. The procedure is shown below.

Maze_Routing(G, T_s, T_t) Algorithm
Input: Underlying grid graph $G(V_G, E_G)$. Disjoint RSTs T_s and T_t embedded in G.
Output: 2-path $p(q,w)$ with $q \in T_s, w \in T_t$
1. $\forall v \in V_G$, set $label(v) = \infty$, $visited(v) = false$, $parent(v) = \emptyset$
2. For each node $v \in T_s$ Set $label(v) = 0$ and set $Q = Q \cup \{v\}$.
3. While $Q \neq \emptyset$
4. Let $v \in Q$ with minimum $label(v)$. Delete v from Q. Set $visited(v) = true$.
5. For each u, such that $(u,v) \in E_G$, $u \neq parent(v)$ $newLbl = label(v) + l_{(u,v)}$. If $blocked(u,v)$ then $newLbl = newLbl + \alpha l_{(u,v)}$
6. If $newLbl < label(u)$ then $label(u) = newLbl$, $parent(u) = v$.
7. If $visited(u) = false$ and $u \notin T_t$, insert u into Q.
8. Find node $w \in T_t$ such that $label(w)$ is minimum.
9. Find path $p(q,w)$ from w to $q \in T_s$ by tracing back $parent$. Return $p(q,w)$.

Step 1 initializes three arrays, $label$, $visited$, and $parent$ for each node in the grid graph. The $label(v)$ value is the cost of the best path from a node in T_s to v, the $visited(v)$ value indicates whether v has been explored, and $parent(v)$ is stores the best path to v. Step 2 initializes the labels of all nodes in T_s to zero and puts them into a priority queue Q. Steps 3-7 search the grid graph by iteratively deleting the node v with smallest label from Q and exploring that node. Each neighbor node u of v is explored in Steps 5-6, and the label for u is updated according to length of edge (u,v) and whether edge (u,v) is blocked. If the new label, corresponding to a path to u through v, is less than the previous label, the label is updated and v becomes the parent for u. Steps 8-9 find the node with the smallest label in the target tree, and uncover the path back to the source tree by following the $parent$ structure.

The time complexity of Maze_Routing is $O((|V|+|B|)^2 \log(|V|+|B|))$ since the size of the grid graph is $O((|V|+|B|)^2)$. The number of times this procedure is called is bounded by $O(|V|)$, which means the complexity for the entire algorithm is $O(|V|(|V|+|B|)^2 \log(|V|+|B|))$.

5. IMPROVING EFFICIENCY

The high time complexity makes speedups a necessity. We have implemented two techniques, a sparsified grid graph construction and branch-and-bound maze routing, that together improve runtimes by more than a factor of ten.

When $|B|$ is large, EHG may be dense. Several edges can be extremely close together, and a routing tree algorithm could choose any of these edges and result in essentially the same tree. Our first sparsification technique searches for pairs of redundant tracks that are closer than a given step size, such as 0.1 mm, and removes one of them. Our second technique severs some tracks which span the entire grid graph. If a grid edge is caused by a blockage in the lower left, it likely does not need a corresponding track in the lower right part of the design. So the track is cut short to avoid spanning the entire grid.

Since maze routing cannot distinguish between good and bad global directions, branch-and-bound techniques are required for efficient implementation. Recall Steps 3-7 of Steiner_Tree Algorithm which iteratively delete and then reconstruct 2-paths. The 2-path $p(u,v)$ removed in Step 5 has $cost(p(u,v))$ which is also an upper bound for the cost of the new 2-path.. Let $upCost$ denote this value. After Step 4 of the maze routing procedure, one can compare $label(v)$ to $upCost$ to determine if node v is worth expanding. If $label(v) > upCost$ then the cost of the path from T_s to v is already higher than the cost of the original 2-path, which makes it wasteful to expand v. Whenever a node $v \in T_t$ is reached, the value for $upCost$ can be replaced by $label(v)$ if this value is less than $upCost$.

The bound can be made even tighter by using a lower bound on the cost of the remaining routing to be done from v to T_t. Let $dist(v, T_t)$ be the Manhattan distance from v to the bounding box of T_t (which can be computed in constant time).[1] Now the test becomes whether $label(v) + dist(v, T_t) > upCost$ holds. If so, node v is not worth further exploration and and Step 7 of the maze routing procedure is skipped.

6. EXPERIMENTS

We performed experiments on two designs. Test1 is a small hand crafted test case [10] and Test2 is a large macro block. The comparisons that follow are made between two algorithms, SMT, a Steiner minimal tree algorithm that is used for net analysis within an industrial physical design tool suite, and BBB, our proposed algorithm.

Note that works which perform simultaneous buffer insertion and routing tree construction such as [7] [9] are inappropriate since they do not consider blockages. The work of [12] attacks the right problem space, but cannot be applied to multi-pin nets. Run times are reported in seconds for an IBM RS6000/595 processor with 512MB of RAM.

We first measure the additional wire length caused by BBB compared to SMT. Since BBB is aware of blockage constraints while SMT is not, BBB should naturally increase total wire length, while decreasing wire length overlapping blockages. Tables 1 and 2 present these results for Test1 and Test2, respectively.

For Test1, SMT and BBB were on 23 nets with 7 random blockages inserted and on 30 other nets with 7 random

[1] If $\alpha < 0$, then $(1+\alpha)dist(v, T_t)$ is the lower bound.

mode	#net	Avg. wire length			Avg. in-bloc. length		
		SMT	BBB	%imp	SMT	BBB	%imp
blck	23	21.3	21.7	-1.8	15.9	5.8	63.5
bays	30	22.7	22.2	-2.2	22.2	10.4	52.7

Table 1: Routing costs of SMT versus BBB for Test1.

buffer bays inserted. Table 1 presents the averaged results. The average wire length increases by only 1.8% for blockages and 2.2% for buffer bays which indicates that BBB is virtually as effective as SMT obtaining low wire lengths. However, the total wire length in blocked regions was reduced by 63.5% for blockages and 52.7% for buffer bays by BBB. The average CPU time to run BBB on a net was less than 0.2 seconds for both blockages and bays.

net (#pin)	Wire length			Blocked wire length			cpu
	SMT	BBB	impr.	SMT	BBB	impr.	
1 (2)	10.7	12.2	-13.9%	9.3	2.0	78.6%	0.5
2 (2)	9.0	9.0	0.0%	5.2	0.4	92.9%	0.8
3 (9)	14.6	15.1	-3.8%	12.7	4.9	61.4%	1.3
4 (9)	14.6	15.2	-4.6%	12.8	7.1	44.4%	1.3
5 (9)	18.4	18.7	-1.7%	18.2	14.0	23.2%	2.2
6 (11)	17.1	17.6	-2.8%	17.1	2.6	84.9%	2.7
7 (17)	24.1	24.1	-0.1%	22.4	21.9	2.3%	5.8
8 (19)	19.7	20.7	-5.0%	19.7	16.6	16.0%	5.2
9 (19)	20.2	20.8	-3.2%	20.2	17.7	12.3%	5.6
10 (25)	22.2	22.3	-0.3%	22.0	20.9	4.9%	4.7
11 (25)	22.6	22.7	-0.4%	22.4	21.3	4.9%	4.8
12 (25)	23.6	24.1	-2.1%	23.5	14.6	37.8%	5.9
13 (29)	23.3	23.9	-2.8%	15.7	10.9	30.3%	5.4
14 (29)	24.9	25.1	-0.6%	18.4	14.2	22.6%	4.9
15 (29)	30.5	31.4	-3.0%	23.3	11.2	51.8%	9.8
16 (29)	29.0	30.4	-5.0%	19.9	8.6	56.7%	14.0
Ave	20.3	20.8	-2.5%	17.7	11.8	33.3%	4.7

Table 2: Routing costs of SMT versus BBB for Test2.

For Test2, we examined 16 timing critical nets that had differentiating characteristics (number of pins, pin locations, wire length topology, etc.) and ran both algorithms with the 54 blockages that were present in the design. Table 2 presents the results. Observe that BBB results in an average of 2.5% higher than SMT while reducing blocked wire length by 33.3%.

To assess the utility of BBB versus SMT trees, buffer insertion must be performed after routing. The next set of experiments were performed on a net by net basis with the following methodology: (i) compute the SMT tree for the net, (ii) compute the delays to each sink, then compute the slack to the most critical sink based on the RATs supplied by the static timing analyzer, (iii) run BBB re-routing, (iv) perform buffer insertion, and (v) re-compute the slack to the most critical sink. Let $\Delta slack$ denote the difference between this slack and the slack computed in Step 2. Skipping (iii) yields buffer insertion with the SMT algorithm while including (iii) yields results for the BBB algorithm.

Average results for Test1 are presented in Table 3 with $\Delta slack$ values presented in ps. Observe that BBB utilizes more buffers than SMT (2.9 versus 2.2 for blockage and 2.3 versus 1.9 for bays) since BBB offers more potential locations for buffers. BBB trees also reduced slack by an additional 337 (768) ps over SMT trees for blockage (bay) mode.

Table 4 presents the same experiments for the 16 Test2 nets. Overall, SMT trees resulted in an average slack improvement of 519.4 ps as compared to 694.6 ps for BBB. The runtimes reported are for the combination of BBB plus the buffer insertion step. By comparing these runtimes to those reported for BBB alone in Table 2, we see that the runtimes of BBB do not dominate the buffer insertion runtimes.

mode	#net	SMT + BI (Ave.)		BBB + BI (Ave.)		
		$\Delta slack$	#buf	$\Delta slack$	#buf	CPU
blockage	23	2064	2.2	2401	2.9	4.0
bays	30	2494	1.9	3262	2.3	4.3

Table 3: Average slack improvements for Test1.

net (#pin)	SMT $\Delta slack$	BBB $\Delta slack$	#buf	CPU(s)
1 (2)	1032	1118	2	1.2
2 (2)	1034	1036	1	1.2
3 (9)	109	239	2	2.1
4 (9)	109	236	2	2.2
5 (9)	190	452	1	2.9
6 (11)	7	71	1	4.0
7 (17)	850	1181	2	7.8
8 (19)	578	1089	2	7.3
9 (19)	605	880	2	7.8
10 (25)	277	299	2	7.6
11 (25)	295	323	2	7.4
12 (25)	205	228	2	9.0
13 (29)	223	308	5	24.4
14 (29)	371	375	4	25.2
15 (29)	1049	1605	7	35.3
16 (29)	1376	1674	5	36.6
Ave.	519.4	694.6	2.6	11.4

Table 4: Slack improvement for Test2.

We proposed a new Steiner tree heuristic for making nets more amenable to buffer insertion in the presence of blockages Experimental results show that our method achieves the objective of avoiding buffer blockages (or seeking buffer bays) and can provide significant improvements in terms of delay when used in conjunction with an industrial buffer insertion tool.

7. REFERENCES

[1] C. J. Alpert, A. Devgan and S. T. Quay, "Buffer Insertion with Accurate Gate and Interconnect Delay Computation," ACM/IEEE DAC, pp. 479-484, 1999.

[2] J. Cong, L. He, C.-K. Koh, and P. H. Madden, "Performance Optimization of VLSI Interconnect Layout," Integration: the VLSI Journal, vol. 21, pp. 1-94, 1996.

[3] M. Hanan, "On Steiner's Problem with Rectilinear Distance," SIAM J. Appl. Math., 14(2), pp. 255-265, 1966.

[4] D. W. Hightower, "A Solution to Line Routing Problems on the Continuous Plane," in Sixth Design Automation Workshop, pp. 1-24, 1969

[5] J. Hu and S. S. Sapatnekar, "Simultaneous Buffer Insertion and Non-Hanan Optimization for VLSI Interconnect under a Higher Order AWE Model," Intl. Symp. Physical Design, pp. 133-138, 1999.

[6] J. Lillis, C.-K. Cheng and T.-T. Y. Lin, "Optimal Wire Sizing and Buffer Insertion for Low Power and a generalized Delay Model," IEEE J. Solid-State Circuits, 31(3), pp. 437-447, 1996.

[7] J. Lillis, C.-K. Cheng, and T.-T. Y. Lin, "Simultaneous Routing and Buffer Insertion for High Performance Interconnect," Great Lakes Symp. VLSI, pp. 148-153, 1996.

[8] C. Y. Lee, "An algorithm for Path Connection and its Applications," IRE Transactions on Electronic Computers, vol. EC-10, no. 3, pp. 346-365, 1961.

[9] T. Okamoto and J. Cong, "Buffered Steiner Tree Construction with Wire sizing for Interconnect Layout Optimization," IEEE/ACM ICCAD , pp. 44-49, 1996.

[10] W. Thirtle, personal communication, 1997.

[11] L. P. P. P. van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay," IEEE ISCAS, pp. 865-868, 1990.

[12] H. Zhou, D. F. Wong, I-M. Liu and A. Aziz, "Simultaneous Routing and Buffer Insertion with Restrictions on Buffer Locations," ACM/IEEE DAC, pp. 96-99, 1999.