

# Interconnect Layout Optimization by Simultaneous Steiner Tree Construction and Buffer Insertion \*

Takumi Okamoto<sup>1,2</sup>  
okamoto@cs.ucla.edu

Jason Cong<sup>1</sup>  
cong@cs.ucla.edu

<sup>1</sup>Dept. of Computer Science, University of California, Los Angeles, CA 90095

<sup>2</sup>C&C Research Laboratories, NEC Corp., Miyamae, Kawasaki 216, Japan

## Abstract

This paper presents an algorithm for interconnect layout optimization with buffer insertion. Given a *source* and  $n$  *sinks* of a signal net, with given *positions* and a *required arrival time* associated with each sink, the algorithm finds a *buffered Steiner tree* so that the required arrival time (or timing slack) at the source is maximized. In the algorithm, Steiner routing tree construction and buffer insertion are achieved simultaneously by combining A-tree construction and dynamic programming based buffer insertion algorithms, while these two steps were carried out independently in the past. Extensive experimental results indicate that our approach outperforms conventional two-step approaches. Our buffered Steiner trees increase the timing slack at the source by up to 75% compared with those by the conventional approaches.

## 1. Introduction

For timing optimization of VLSI circuits, buffer insertion (or fanout optimization) and interconnect topology optimization take important roles and a number of algorithms were proposed for these problems over the past a few years.

On fanout optimization problem, most of previous work focused on construction of buffered trees in logic synthesis with consideration of user-defined timing and area constraints[1, 2, 3]. The timing measures used during this stage mainly consist of gate delays and a rough approximation for interconnect delay, which is assumed to be piecewise linear with the number of fanouts. When the wiring effect is dominant, traditional synthesis tools that use such a fanout-based model may be optimizing a timing value which is significantly different from the actual post layout value. Another problem with traditional synthesis is in area estimation. Typically, the tools try to optimize only the total gate area, and the interconnect area and the routability of the chip are not taken into account. As a result, although the total gate area of the synthesized netlist is quite small, it may not fit into the target die area after layout.

In recent years, [4, 6, 5, 7, 8] attack the fanout optimization problem after layout information is available. In [4], a fanout optimization algorithm based on alphabetic trees is presented that generates fanout trees free of internal edge crossings thus improving routing area. In [5], buffer insertion based on a minimum spanning tree is proposed. In [6], a polynomial time algorithm using dynamic programming is proposed for delay-optimal buffer insertion problem on a

given tree topology. [7, 8] has integrated wire sizing and power minimization with the algorithm in [6] under a more accurate delay model taking signal slew into account.

On interconnect topology optimization problem, the analysis in [9] and [10] showed that as we reduce the device dimension, resistance ratio, which is defined as the ratio of the driver resistance versus the unit wire resistance, decreases. As a result the *distributed* nature of the interconnect structure must be considered, and conventional algorithm for total wire capacitance minimization does not necessarily lead to the minimum interconnect delay. For interconnect optimization in deep submicron VLSI design, recently a number of interconnect topology optimization algorithms have been proposed, including bounded-radius bounded-cost trees[11], AHHK trees[12], maximum performance trees[13], A-trees[10], low-delay trees[14, 15], and IDW/CFD trees[16].

Although steady progress has been made in buffer insertion and Steiner tree construction for delay minimization, and encouraging experimental results were reported, we believe that these two steps need to be carried out simultaneously in order to construct even higher performance buffered Steiner trees directly. The independent two-step approach often leads to sub-optimal designs due to the following reasons: in the case of buffer insertion followed by Steiner tree construction, there is a problem that wiring delay and routability can not be estimated accurately in buffer insertion as mentioned above; in the case of Steiner tree construction followed by buffer insertion, there is a problem that Steiner tree optimized for delay does not necessarily result in a minimum-delay buffered Steiner tree. Figure 1 shows an example, where sink  $s_1$  is the most critical among all the sinks. In certain cases, which depends on the technology and criticality of sinks, the buffered Steiner tree in Figure 1(a) is desired, while Figure 1(b) shows a minimum-delay Steiner tree followed by buffer insertion.

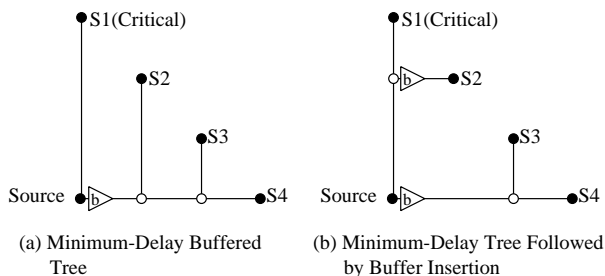


Figure 1. Example of Buffered Steiner Tree

\*This work is partially supported by National Science Foundation Young Investigator Award MIP9357582 and a matching grant from Intel Corporation.

In this paper, we present an algorithm for interconnect layout optimization with buffer insertion. Given a *source* and  $n$  *sinks* of a signal net, with given *positions* and a *required arrival time* associated with each sink, the algorithm finds a buffered Steiner tree so that the required arrival time (or timing slack) at the source is maximized. In the algorithm, Steiner tree construction and buffer insertion are achieved simultaneously by combining A-tree algorithm [10] and dynamic programming based buffer insertion algorithm [6]. Extensive experimental results indicate that our approach outperforms conventional two-step approaches. Our buffered Steiner trees increase the timing slack at the source by up to 75% compared with those by the conventional approaches.

## 2. Delay Models and Problem Formulation

### 2.1. Delay Models

As in most previous works on interconnect layout optimization, we adopt the Elmore delay model [17] for interconnects and standard RC models for buffers. For wire  $e$ , let  $l_e$ ,  $c_e$  and  $r_e$  be its length, capacitance and resistance, respectively. Further, let  $e_v$  denote the wire entering node  $v$  from its parent. We use the following basic models for interconnects delay  $D_{wire}$  and buffer delay  $D_{buff}$ :

$$c_e = c_0 l_e, \quad r_e = r_0 l_e$$

$$D_{wire}(e_v) = r_{e_v} \left( \frac{c_{e_v}}{2} + c(T_v) \right)$$

$$D_{buff}(b, c_l) = d_b + r_b c_l,$$

where  $c_0$  and  $r_0$  are capacitance and resistance for unit length wire, respectively,  $c(T_v)$  is the lumped capacitance of subtree  $T_v$  rooted at  $v$ ,  $d_b$  and  $r_b$  are buffer  $b$ 's intrinsic delay and output resistance, respectively, and  $c_l$  is the load on buffer  $b$ . When wire  $e$  is very long, we can divide  $e$  into a sequence of wires connected by degree-2 nodes to capture the distributed nature of the interconnect delay. Note that we assume wires are of a uniform width. Wiresize optimization can be carried out in a separate step after the buffered tree construction using the algorithm in [18, 19] or during the buffered tree construction as mentioned in Section 6.

### 2.2. Problem Formulation

We use *required arrival time* as our optimization objective. The required arrival time at the root of tree  $T_v$ , denoted  $q(T_v)$ , is defined as follows:

$$q(T_v) = \min_{u \in \text{sinks}(T_v)} (q_u - \text{delay}(v, u)),$$

where  $q_u$  is the required arrival time of sink  $u$ ,  $\text{sinks}(T_v)$  is a set of sinks of tree  $T_v$ , and  $\text{delay}(u, v)$  is delay from  $v$  to  $u$  defined by our delay models. This measure is useful since it is a typical objective when optimizing the performance of combinational networks. If we assume the signal arrives at the root of  $T$  at  $t = 0$ , the timing requirements are met when  $q(T)$  is non-negative. The *buffered Steiner tree problem* for delay minimization is stated as follows:

**Given:** A source  $s_0$  and sinks  $s_1, s_2, \dots, s_n$  of a signal  $S$ , with given positions and a required arrival time associated with  $s_i$  ( $1 \leq i \leq n$ ).

**Find:** Steiner tree  $T_{s_0}$  that spans  $S$  and has buffers inserted.

**Objective:** Maximize  $q(T_{s_0})$ .

Hereafter, it is assumed that only one type of buffer is considered for the buffer insertion, and signal polarity is neglected. Our algorithm, however, is easily extended to general case, where more than one types of buffer can be used and signal polarity must be considered, by using the methods similar to those in [7, 8].

## 3. Related Work

We briefly review the A-tree algorithm in [10] and the buffer insertion algorithm in [6], which are basis of our proposed algorithm.

### 3.1. A-tree Algorithm

In [10], it was shown that a routing tree which minimizes the Elmore delay upper bound in [20] can be achieved by minimizing a weighted combination of the objectives of the minimum Steiner tree, the shortest path tree, and the "quadratic minimum Steiner tree" (a tree that minimizes the summation of source-node path lengths, taken over all possible node locations). Therefore, a minimum-cost *rectilinear arborescence* (*A-tree*) formulated in [21] is of interest since it heuristically addresses all of these terms in the decomposed upper bound at once.

**Definition 1:** A *rectilinear Steiner tree*  $T$  is called an *A-tree* if every path connecting the source  $s_0$  and any node  $p$  on the tree is a shortest path.

In [10], an efficient algorithm based on bottom-up tree construction from the sinks was proposed for minimum-cost *A-tree*, which extends the algorithm in [21]. The algorithm starts with a set of subtrees, each consisting of a sink, and iteratively performs two subtrees "merging" or a subtree "growing" until all subtrees are merged into one tree. Two type of move is used for the bottom-up construction: *safe move* which cannot worsen the sub-optimality of an existing set of subtrees and *heuristic move* that may not lead to an optimal solution. According to their experimental results, A-trees constructed by the algorithm are at most 4% within the optimal, and achieve interconnect delay reduction by as much as 66% when compared to the best-known Steiner routing topology. In our approach here, we use only heuristic move in the A-tree algorithm for simplicity (essentially the algorithm in [21]). Despite using only heuristic moves, [21] has similar performance as the A-tree algorithm in [10].

The algorithm in [21] works as follows: A set called *ROOT* consisting of the roots of current subtrees which will eventually be merged to form the solution is maintained; Initially, *ROOT* contains the roots of  $n$  trivial trees, each consisting of a single sink. The algorithm then iteratively merges a pair of roots such that the "merged" root is as far from the source as possible, and terminates when  $|ROOT| = 1$ .

More formal description of the algorithm is shown in Figure 2, where all sinks are assumed lie in the first quadrant with  $s_0$  at the origin for simplicity. But it is easy to extend the algorithm to the general case.

### 3.2. Buffer Insertion

For given required arrival times at the sinks of a given Steiner tree, the buffer insertion algorithm in [6] chooses the buffering position on the tree such that the required arrival time at the source is as late as possible, where the delay is calculated based on the definition in Section 2. The algorithm assumes that the topology of the routing tree (or Steiner tree) is given, as well as the possible (legal) positions of the buffers.

```

Procedure Heuristic_Atree()
   $ROOT \leftarrow \{s_i \mid 0 \leq i \leq n\}$ ;
  while  $|ROOT| > 1$  do
    Find  $v, w \in ROOT$  such that the sum
       $\min(v_x, w_x) + \min(v_y, w_y)$  is maximum;
     $ROOT \leftarrow ROOT + \{r\} - \{v\} - \{w\}$ , where  $r$  is a node
      with coordinates  $(\min(v_x, w_x), \min(v_y, w_y))$ ;
    Merge  $T_v$  and  $T_w$  to  $T_r$ 
      adding edges from  $r$  to  $v$  and  $w$ , respectively;
  end while;
end procedure

```

Figure 2. A-tree Algorithm Using Heuristic Move

In the algorithm, which is based on dynamic programming technique, a set of  $(q_i, c_i)$  pairs is maintained for possible buffers assignment at each legal positions of the buffers, where  $q_i$  and  $c_i$  are the required arrival time and the capacitance of *dc-connected subtree*<sup>1</sup> rooted at  $i$  corresponding to  $q_i$ , respectively (Figure 3). Note that  $c_i$  is *not* the total capacitance of entire subtree rooted at  $i$ . Each pair is called an *option*. The algorithm consists of two phases as follows. During the first phase the function “bottom\_up( )” in Figure 4 computes the *irredundant set* of all possible options<sup>2</sup>,  $Z_v$ , for each node  $v$  (or legal positions of the buffers) in the tree in bottom up manner (Figure 5(a))<sup>3</sup>. For the options at the root of the entire tree, the actual delay is calculated, using the output resistance  $R_{gate}$  of the gate which produces the signal:  $q_{source} = q_0 - R_{gate}c_0$ , and then the option which gives the maximum  $q_{source}$  is chosen. The second phase traces back the computations of the first phase that led to this option, and determines the computed buffer positions on the way (Figure 5(b)).

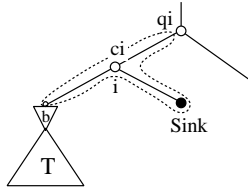


Figure 3. An Option at the Root of Subtree  $T_i$

In the algorithm in [6], candidate points for the buffer insertion are right after the Steiner points in the tree, which makes it possible to unload the critical path as much as possible (Figure 6(a)). In our implementation, we also make each Steiner point itself be a candidate (Figure 6(b)), in addition to the points right after the Steiner points, in order to reduce the number of buffers inserted. Moreover, an edge whose length is longer than certain threshold given by user is divided in order to make it possible to insert buffer in the middle of the wire (Figure 6(b)).

<sup>1</sup> “dc-connected” means “directly connected by wires”.

<sup>2</sup> *Irredundant set* has no two options  $(q, c)$  and  $(q', c')$  such that  $q > q'$  and  $c \leq c'$  [6].

<sup>3</sup> For simplicity, a binary tree is assumed here, but the algorithm is easily applied to general trees by addition of dummy nodes and 0 length wires [8]. Node which has only one child, where  $Z_l$  or  $Z_r$  are NULL in Figure 4, can be also treated by a simple extension.

```

Procedure bottom_up(T)
foreach  $v \in T$  in topological order from sinks to source do
  if  $v$  is a sink then
     $Z_v \leftarrow (q_v - D_{wire}(e_v), c_v + c_{e_v})$ ;
  else
     $Z_l \leftarrow$  a set of options for  $v$ 's left child;
     $Z_r \leftarrow$  a set of options for  $v$ 's right child;
     $Z_v \leftarrow \phi$ ;
    for  $(i \in Z_l, j \in Z_r)$  do
      /* Irredundant Merge of  $Z_l$  and  $Z_r$  */
      if  $q_i \leq q_j$  then
         $Z_v \leftarrow Z_v \cup (q_i, c_i + \min\{c_j \mid q_i \leq q_j\})$ ;
      end if;
    end for;
     $Z_v \leftarrow Z_v \cup (\max_{z \in Z_v} (q_z - D_{buff}(b, c_z)), c_b)$ ;
    for  $z \in Z_v$  do
       $q_z \leftarrow q_z - D_{wire}(e_v)$ ;
       $c_z \leftarrow c_z + c_{e_v}$ ;
    end for;
  end if;
end for;
end Procedure;

```

Figure 4. Algorithm Finding a Set of Options

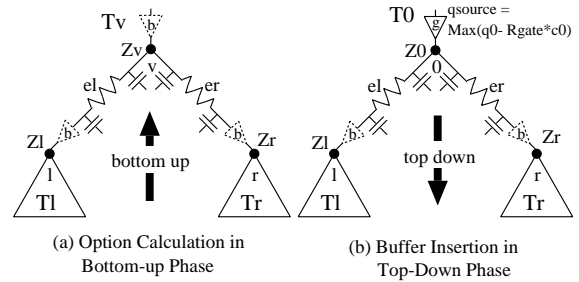


Figure 5. Illustration of buffer insertion

## 4. Simultaneous Steiner Tree Construction and Buffer Insertion

### 4.1. Basic Idea of the Proposed Approach

We develop an algorithm for simultaneous Steiner tree construction and buffer insertion, called *buffered A-tree (BA-tree)* algorithm, by combining the A-tree and buffer insertion algorithms in Section 3. In such combination, the concept of *critical path isolation* (Figure 7(a)) and *balanced load decomposition* (Figure 7(b)) are also applied, which are techniques used for fanout optimization (or buffer insertion) in logic synthesis [1, 2, 3]. In logic synthesis, when one or several sinks are timing-critical, the critical path isolation tech-

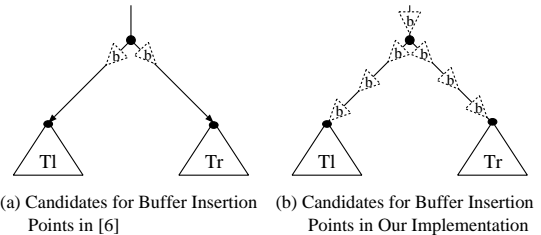


Figure 6. Candidate Points for Buffer Insertion

nique generates a fanout tree so that the root gate drives the critical sinks and a smaller additional load due to buffered non-critical paths. On the other hand, if required times at sinks are within a small range, balanced load decomposition is applied in order to decrease the load at output of root gate. These transformations are applied recursively in a bottom-up process from the sinks in the same manner as the A-tree and buffer insertion algorithms. Therefore, it is natural for us to apply these techniques in combination with the A-tree and buffer insertion algorithms.

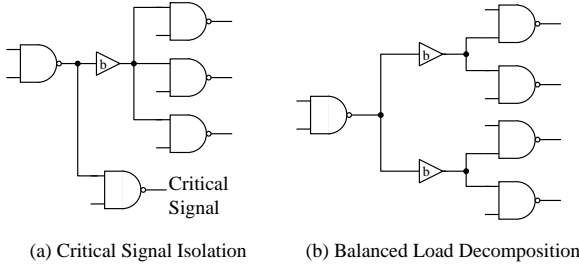


Figure 7. Fanout Optimization in Logic Synthesis

In our approach, the concepts of critical path isolation and balanced load decomposition are used when choosing two subtrees ( $T_v$  and  $T_w$ ) to be merged in the A-tree algorithm. Every pair of subtree roots  $v$  and  $w$  are evaluated by computing the required time at the root of subtree  $T_r$ , which results from merging of  $T_v$  and  $T_w$ . Then, the best pair for merging is chosen so that critical path isolation and balanced load decomposition are achieved (See Section 4.2). The required times at the root of  $T_r$  is calculated based on the options at  $v$  and  $w$ ,  $dist(r, v)$  and  $dist(r, w)$ <sup>4</sup> for the interconnect delay, and the effect of buffer insertion at  $r$ . For the evaluation, we keep a set of options at each of subtree's roots by using `bottom_up()` during the construction of A-tree.

Basically the following two steps are iterated in BA-tree algorithm.

1. Select  $v$  and  $w$  with taking critical path isolation and balanced load decomposition into account.
2. Merge  $T_v$  and  $T_w$  to  $T_r$ , and compute a set of options at  $r$  by `bottom_up( $T_r$ )`.

#### 4.2. Selection of Roots to be Merged in BA-tree

In our algorithm, the computation of options and tree construction are performed simultaneously. Suppose that subtrees  $T_v$  and  $T_w$  are merged into  $T_r$  as shown in Figure 8. Let  $Z_v$  and  $Z_w$  be the sets of options at  $v$  and  $w$ , respectively computed in the previous steps. Based on  $Z_v$ ,  $Z_w$ ,  $dist(r, v)$ ,  $dist(r, w)$ , and buffer  $b$ 's characteristics, a set of options  $Z_r$  at  $r$  are temporarily computed for evaluation of the best merge. Since the parent nodes of the current subtree's roots,  $v$  and  $w$ , are not determined yet at this stage,  $l_{e_v}$  and  $l_{e_w}$  were assumed to be 0 in the computation of  $Z_v$  and  $Z_w$ , respectively. In the temporary computation of  $Z_r$ , we update  $Z_v$  and  $Z_w$  using  $l_{e_v} = dist(r, v)$  and  $l_{e_w} = dist(r, w)$  when computing the arrival time at  $r$  with the assumption that  $l_{e_r} = dist(s_0, r)$ . Note that  $dist(s_0, r)$  is an upper bound of  $l_{e_r}$ .

We introduce the following definitions before describing how to select two subtrees to be merged in BA-tree construction.

<sup>4</sup> $dist(v, w)$  denotes Manhattan distance between  $v$  and  $w$

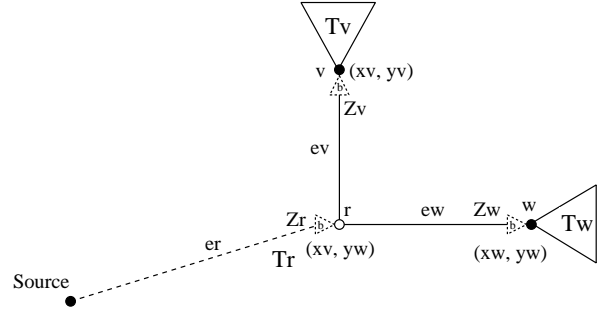


Figure 8. Evaluation of a Merged Subtree

**Definition 2:** The maximum possible required time at the root  $r$  of subtree  $T_r$  generated by merging of  $T_v$  and  $T_w$ , denoted  $R_{vw}$ , is defined as follows:

$$R_{vw} = \max_{z \in Z_r} q_z,$$

where  $r$  is the merging point of  $T_v$  and  $T_w$ , and  $Z_r$  is a set of options at  $r$ .

**Definition 3:** The maximum  $R_{vw}$  among all possible merging pairs  $v$  and  $w$  in the set of roots  $ROOT$  of the current subtrees, denoted  $R_{max}(ROOT)$ , is defined as follows:

$$R_{max}(ROOT) = \max_{v, w \in ROOT} R_{vw}.$$

**Definition 4:** The distance between the source and the merging point for  $v$  and  $w$ , denoted  $D_{vw}$ , is defined as follows:

$$D_{vw} = \min(v_x, w_x) + \min(v_y, w_y).$$

This definition is for the case that  $v$  and  $w$  are in the first quadrant with  $s_0$  at the origin. Other cases can be defined in a similar way.

**Definition 5:** The maximum  $D_{vw}$  among all possible merging pairs  $v$  and  $w$  in the set of roots  $ROOT$  of the current subtrees, denoted  $D_{max}(ROOT)$ , is defined as follows:

$$D_{max}(ROOT) = \max_{v, w \in ROOT} D_{vw}.$$

Now, we can define the merging cost for  $v$  and  $w$ .

**Definition 6:** The merging cost for  $v$  and  $w$ , denoted  $mcost(v, w, ROOT)$ , is defined as follows:

$$mcost(v, w, ROOT) = \alpha * \frac{R_{vw}}{R_{max}(ROOT)} + (1 - \alpha) * \frac{D_{vw}}{D_{max}(ROOT)},$$

where  $\alpha$  is a fixed constant with  $0.0 \leq \alpha \leq 1.0$ .

Note that instead of using  $\alpha * R_{vw} + \beta * D_{vw}$  for the cost, we use the scaled objective above. The use of the scaled objective avoids the problem of choosing a different pairs of  $\alpha$  and  $\beta$  for each instance.

For the subtrees merging in BA-tree algorithm, we select  $T_v$  and  $T_w$  whose  $mcost(v, w, ROOT)$  is maximum among

all possible pairs of subtrees. Clearly, if we set  $\alpha = 0.0$ , our root selection criteria for merging is the same as that in the A-tree algorithm presented in Section 3.1.

By using  $mcost$  in the A-tree construction, required time maximization with buffer insertion (critical path isolation and balanced load decomposition) and wire length minimization can be achieved simultaneously. The second term in  $mcost$  contributes to the wire length minimization as the original A-tree algorithm. The first term contributes to the critical path isolation and balanced load decomposition as the fanout optimization in logic synthesis. When one or several sinks are timing-critical, those sinks are isolated since the merging for those sinks, whose  $R$  are smaller than the others, will be applied in the later stage. Figure 9(a) shows an example for this case, where sink  $s_1$  is the most critical among all the sinks. Sink  $s_1$  will be isolated since the merging for  $s_1$ , whose  $R$  is smaller than the others, will be applied after  $s_4$ ,  $s_3$ , and  $s_2$  are merged. On the other hand, if required times at sinks are within a small range, the merging will be performed so that the load is balanced, since  $R$  of the merging for those sinks are also within a small range. Figure 9(b) shows an example for this case, where required times at sinks  $s_1$ ,  $s_2$ ,  $s_3$ , and  $s_4$  are within a small range. The load will be balanced, since  $R$  of the merging for the sinks are within a small range.

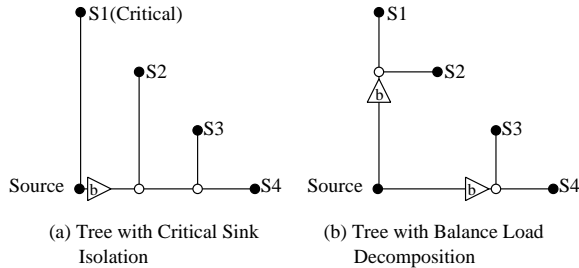


Figure 9. Example of BA-tree

#### 4.3. Overall Algorithm

The algorithm consists of two phases in the same way with the buffer insertion[6]: bottom up tree construction with option computation and top down buffer insertion.

Formal description for the first phase, bottom up tree construction with option computation, is shown in Figure 10. Option computation at each subtree's root by  $bottom\_up()$  and  $mcost(v, w, ROOT)$  evaluation at the merging are integrated into A-tree algorithm.

The second phase, top down buffer insertion, is the same with the one in the buffer insertion[6]. The option which gives the maximum required time at root is chosen, then traces back the computations of the first phase that led to this option. During the backtrace, the buffer positions are determined.

### 5. Experimental Results

We implemented BA-tree on a Sun SPARC 5 workstation under the C/UNIX environment, and tested it on signal nets with 10, 25, 50, and 100 sinks<sup>5</sup>. For each net size, 100 nets were randomly generated on a  $10mm \times 10mm$  routing

<sup>5</sup>The number of sinks in signal nets before buffer insertion is usually up to 100 and most of them are less than 25.

```

Procedure BA-tree_bottomup()
   $ROOT \leftarrow \{s_i \mid 0 \leq i \leq n\}$ ;
  foreach  $v \in ROOT$  do
     $bottom\_up(v)$ ;
    /*  $Z_v$  is computed for each sink */
  end for;
  while  $|ROOT| > 1$  do
    Find  $v, w \in ROOT$  with  $\max_{v, w \in ROOT} mcost(v, w, ROOT)$ ;
    /*  $Z_r$  is temporarily computed for its evaluation */
     $ROOT \leftarrow ROOT + \{r\} - \{v\} - \{w\}$ , where  $r$  is a node
      with coordinates  $(\min(v_x, w_x), \min(v_y, w_y))$ ;
    Merge  $T_v$  and  $T_w$  to  $T_r$ 
      adding edges from  $r$  to  $v$  and  $w$ , respectively;
     $bottom\_up(T_r)$ ;
    /*  $Z_r, Z_v, Z_w$  are re-computed here with pruning */
  end while;
end procedure

```

Figure 10. Algorithm for Simultaneous A-Tree Construction and Option Computation

region, and we evaluated the average results. The loading capacitances and required times at the sinks are also randomly chosen from the intervals  $[0.05pF, 0.15pF]$  and  $[5.0ns, 10.0ns]$ , respectively. The parameters used in the experiments are summarized in Table 1.

Table 1. Parameters for Experiments

Output Resistance of Gate	$R_{gate}$	1000 $\Omega$
Output Resistance of Buffer	$r_b$	800 $\Omega$
Intrinsic delay of Buffer	$d_b$	0.1ns
Wire Resistance	$r_0$	0.12 $\Omega/\mu m$
Wire Capacitance	$c_0$	0.15fF/ $\mu m$
Loading Capacitance of Sink	$c_g$	0.05pF - 0.15pF
Loading Capacitance of Buffer	$c_b$	0.05pF
Required Time at Sink	$q_s$	5.0ns - 10.0ns

We compared results obtained by the following two methods:

M1: A-tree[21] followed by buffer insertion[6].

M2: BA-tree construction ( $\alpha$  in  $mcost$ : 0.2, 0.4).

Table 2 shows average required times at the sources of the buffered Steiner trees generated by the two methods. The difference of the required time is increased as the number of sinks is increased. Although the difference is not so large for nets with 10 sinks, the required time at source of BA-tree is larger than that by M1 by 49% with  $\alpha = 0.2$  and 75% with  $\alpha = 0.4$  for nets with 100 sinks.

Table 2. Required Time at Source(ns)

#sinks	M1	M2( $\alpha : 0.2$ )	M2( $\alpha : 0.4$ )
10	3.05(1.00)	3.07(1.01)	3.10(1.02)
25	2.22(1.00)	2.29(1.03)	2.37(1.07)
50	1.65(1.00)	1.80(1.09)	1.94(1.18)
100	0.88(1.00)	1.31(1.49)	1.54(1.75)

Table 3 shows average runtimes, which is increased due to the merging pair evaluation by 1.5 times (10 sinks) to 40 times (100 sinks) in BA-tree.

**Table 3. Run Time(s)**

#sinks	M1	M2( $\alpha : 0.2$ )	M2( $\alpha : 0.4$ )
10	0.02(1.00)	0.03(1.5)	0.03(1.5)
25	0.03(1.00)	0.16(5.3)	0.15(5.0)
50	0.06(1.00)	1.02(17.0)	1.01(16.8)
100	0.18(1.00)	7.18(42.2)	6.95(38.6)

Table 4 shows wire length, which is increased by 0% (10 sinks) to 7% (100 sinks) with  $\alpha = 0.2$  and 5% (10 sinks) to 28% (100 sinks) with  $\alpha = 0.4$ .

**Table 4. Wire Length(mm)**

#sinks	M1	M2( $\alpha : 0.2$ )	M2( $\alpha : 0.4$ )
10	2.57(1.00)	2.58(1.00)	2.69(1.05)
25	4.25(1.00)	4.32(1.02)	4.70(1.11)
50	6.08(1.00)	6.29(1.03)	7.17(1.18)
100	8.64(1.00)	9.23(1.07)	11.05(1.28)

Table 5 shows the number of buffers inserted, which is also increased by 0% (10 sinks) to 8% (100 sinks) with  $\alpha = 0.2$  and 0% (10 sinks) to 15% (100 sinks) with  $\alpha = 0.4$ . Note that minimization for the number of buffers as in [6] is not considered here. Therefore, redundant buffers might be included in the results.

**Table 5. #Buffers Inserted**

#sinks	M1	M2( $\alpha : 0.2$ )	M2( $\alpha : 0.4$ )
10	8(1.00)	8(1.00)	8(1.00)
25	18(1.00)	18(1.00)	18(1.00)
50	31(1.00)	32(1.03)	33(1.06)
100	53(1.00)	57(1.08)	61(1.15)

Through Table 2 to 5, tradeoff between the required time, wire length and number of buffers can be seen with the different parameter  $\alpha$ .

## 6. Conclusions

In this paper, we have presented an algorithm, BA-tree, which derives buffered Steiner tree so that the required arrival time at the source is maximized. The algorithm achieves Steiner tree construction and buffer insertion simultaneously, while these two steps were carried out independently in the past. We have shown its efficiency and effectiveness experimentally.

Future work will include the total capacitance minimization and their trade-off with the required time at the source. We also plan to incorporate optimal wiresizing for further delay optimization. Our preliminary study shows that all these optimization techniques can be combined in the bottom-up dynamic programming paradigm employed in this paper. Interested readers may contact the authors for the follow-up work.

## References

- [1] C. L. Berman, J. L. Carter, and K. F. Day, "The fanout problem: From theory to practice," *Advanced Research in VLSI: Proc. 1989 Decennial Caltech Conf.*, pp.69-99, 1989.
- [2] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang, "Performance Oriented Technology Mapping," *Proc. sixth MIT VLSI Conf.*, pp.79-97, 1990.
- [3] K. J. Singh and A. Sangiovanni-Vincentelli, "A Heuristic Algorithm for the Fanout Problem," *Proc. ACM/IEEE Design Automation Conf.*, 1990, pp.357-360.
- [4] H. Vaishnav and M. Pedram, "Routability-Driven Fanout Optimization," *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp.230-235.
- [5] L. N. Kannan, P. R. Suaris, and H. G. Fang, "A Methodology and Algorithms for Post-Placement Delay Optimization," *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp.327-332.
- [6] L.P.P. van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay," *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp.865-868.
- [7] J. Lillis, C. K. Cheng, and T. T. Lin, "Optimal and Efficient Buffer Insertion and Wire Sizing," *Proc. IEEE Custom Integrated Circuits Conf.*, 1995, pp.259-262.
- [8] J. Lillis, C. K. Cheng, and T. T. Lin, "Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model," *Proc. IEEE Int. Conf. Computer-Aided Design*, 1995, pp.138-143.
- [9] D. Zhou, F. P. Preparata, and S. M. Kang, "Interconnection Delay in Very High-Speed VLSI," *IEEE Trans. Circuits Syst.*, 38(7), pp.779-790, July 1991.
- [10] J. Cong, K. S. Leung, and D. Zhou, "Performance-Driven Interconnect Design Based on Distributed RC Delay Mode," *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp.606-611.
- [11] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Provably Good Performance-Driven Global Routing," *IEEE Trans. Computer-Aided Design*, 11(6), pp.739-752, June 1992.
- [12] C. J. Alpert, T. C. Hu, H. Huang, and A. B. Kahng, "A Direct Combination of the Prim and Dijkstra Constructions for Improved Performance-Driven Routing," *Proc. IEEE Int. Symp. Circuits Syst.*, 1993, pp.1869-1872.
- [13] J. P. Cohoon and L. J. Randall, "Critical Net Routing," *Proc. IEEE Int. Conf. Computer Design*, 1991, pp.174-177.
- [14] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, "Rectilinear Steiner Trees with Minimum Elmore Delay" *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp.381-386.
- [15] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, "Near-Optimal Critical Sink Routing Tree Constructions," *IEEE Trans. Computer-Aided Design*, 14(12), pp.1417-1436, Dec. 1995.
- [16] X. Hong, T. Xue, E. S. Kuh, C. K. Cheng, and J. Huang, "Performance-Driven Steiner Tree Algorithms for Global Routing," *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp. 177-181.
- [17] W. C. Elmore, "The Transient Response of Damped Linear Network with Particular Regard to Wideband Amplifier," *J. Applied Physics*, 19, pp. 55-63, 1948.
- [18] J. Cong and C.-K. Koh, "Simultaneous Driver and Wire Sizing for Performance and Power Optimization," *IEEE Trans. VLSI*, 2(4), pp.408-423, Dec. 1994.
- [19] J. Cong and K.S. Leung, "Optimal Wiresizing Under the Distributed Elmore Delay Model," *IEEE Trans. Computer-Aided Design*, 14(3), pp.321-336, Mar. 1995.
- [20] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal Delay in RC Tree Networks," *IEEE Trans. Computer-Aided Design*, 2(3), pp.202-211, 1983.
- [21] S.K. Rao, P. Sadayappan, F.K. Hwang, and P.W. Shor, "The Rectilinear Steiner Arborescence Problem," *Algorithmica* 7, pp.277-288, 1992.