Delay Bounded Buffered Tree Construction for Timing Driven Floorplanning

Maggie Z.-W. Kang and Wayne W.-M. Dai
Tom Dillinger and David LaPotin

UCSC-CRL-97-08
April 11, 1997

Baskin Center for
Computer Engineering & Computer Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

As devices and lines shrink into the deep submicron range, the propagation delay of signals can be effectively improved by repowering the signals using intermediate buffers placed within the routing trees. Almost no existing timing driven floorplanning and placement approaches consider the option of buffer insertion. As such, they may exclude solutions, particularly early in the design process, with smaller overall area and better routability. In this paper, we propose a new methodology in which buffered trees are used to estimate wire delay during floorplanning and placement. Instead of treating delay as one of the objectives, as done by the majority of previous work, we formulate the problem in terms of Delay Bounded Buffered Trees (DBB-tree). The DBB formulation is as follows: Given a net and delay bounds on critical sinks, construct a tree with intermediate buffers inserted to minimize both the total wiring length and the number of buffers while satisfying the given delay bounds. Based on the Elmore delay model, we propose an efficient algorithm to construct a DBB spanning tree for use during floorplanning and placement. Experimental results show that the algorithm is very effective. Using buffer insertion at the floorplanning and placement stage yields significantly better solutions in terms of both chip area and total wire length.

Keywords: Elmore delay, total wire length, timing constraints, buffer insertion, DBB-tree, MST-tree, SPT-tree, floorplanning.

1 Introduction

In high speed design, long on-chip interconnects can be modeled as distributed delay lines, where the delay of the lines can often be reduced by wire sizing or intermediate buffer insertion. Simple wire sizing is one degree of freedom available to the designer, but often it is ineffective due to area, routability, and capacitance considerations. On the other hand, driver sizing and buffer insertion are powerful tools for reducing delay, given reasonable power constraints. Intermediate buffers can effectively decouple a large load off of a critical path or divide a long wire into smaller segments, each of which has less line resistance and makes the path delay more linear with overall length. As the devices and lines shrink into deep submicron, it is more effective, in terms of power, area, and routability, to insert intermediate buffers than to rely solely on wire sizing.

Because floorplanning and placement have a significant impact on critical path delay, research in the area has focused on timing driven approaches. Almost no existing floorplanning and placement techniques consider the option of buffer insertion, particularly early in the design cycle. Typically, only wire length or Elmore delay is used for delay calculation. This practice is too restrictive as evidenced by the reliance industry has placed on intermediate buffering as a means for achieving aggressive cycle times. It is commonplace for production chips to contain tens of thousands of buffers. This paper attempts to leverage the additional freedom gained by inserting buffers during floorplanning and placement. The resulting formulation provides an additional degree of freedom not present in past approaches and typically leads to solutions with smaller area and increased routability.

To incorporate buffer insertion into early planning stage, we propose a new methodology of floorplanning and placement using buffered trees to estimate the wiring delay. We formulate the Delay Bounded Buffered Tree (DBB-tree) problem as follows: Given a net with delay bounds on the critical sinks that are associated with critical paths, construct a tree with intermediate buffers inserted to minimize both the total wiring length and the number of buffers, while satisfying the delay bounds. We propose an efficient algorithm based on the Elmore delay model to construct DBB spanning trees for use during floorplanning and placement. The experimental results of the DBB spanning tree show that using buffer insertion at the floorplanning stage yields significantly better solutions in terms of both chip area and total wiring length.

The remainder of the paper is organized as follows. Section 2 reviews the related works on interconnect optimization and intermediate buffer insertion, and introduces the idea of our DBB spanning tree algorithm. Section 3 describes the DBB algorithm in detail. The experimental results of DBB spanning tree algorithm applied for signal nets and for general floorplanning are given in Section 4, followed by conclusions in Section 5.

2 Related Works and Overview of DBB-tree Algorithm

2.1 Elmore Delay Model

As VLSI design reaches deep submicron, interconnect delay models have evolved from the simplistic lumped RC model to the sophisticated high-order moment-matching delay model [1]. The *Elmore delay model* [2] provides a simple closed-form expression with greatly improved accuracy for delay compared to the lumped RC model. Elmore is the most commonly used delay model in recent works of interconnect design.

For each wire segment modeled as a $\pi-$type circuit, given the interconnect tree T, the Elmore delay from the source s_0 to sink s_i can be expressed as follows:

$$\tau(0,i) = R_0 C_0 + \sum_{e(u,v)\in Path(0,i)} rl_{u,v}\left(\frac{cl_{u,v}}{2} + C_v\right) \qquad (1)$$

where R_0 is the driver resistance at the source and C_0 is the total capacitance charged by the driver. $Path(0,i)$ denotes the path from s_0 to s_i and wire $e(u,v)$ connecting s_v to its parent s_u. Given a uniform wire width, r and c denote the unit resistance and unit capacitance respectively. The wire resistance $rl_{u,v}$ and wire capacitance $cl_{u,v}$ are proportional to the wire length $l_{u,v}$. Let C_v denote the total capacitance of a subtree rooted at s_v, which is charged through wire $e(u,v)$. The first term of $\tau(0,i)$ is linear with the total wire length of T, while the second term has quadratic dependence on the length of the path from the source to s_i.

2.2 Topology Optimization for Interconnect

From the previous discussion of Elmore delay, we can conclude that for interconnect topology optimization, two major concerns are the total wire length and the path length from the driver to the critical sinks. The early work of Cohoon Randall [3] and Cong et al. [4] observed the existence of conflicting min-cost and min-radius (the longest source-to-sink path length of the tree) objectives in performance-driven routing [5].

A number of algorithms have been proposed to make the trade-offs between the total wiring length and the radius of the Steiner or spanning tree [6, 7, 8, 9]. Cong et al.

proposed the "Bounded Radius, Bounded Cost" (BRBC) spanning tree algorithm which uses the shallow-light approach. BRBC constructs a routing tree with total wire length no greater than $(1 + 2/\epsilon)$ times that of a minimum spanning tree and radius no greater than $(1 + \epsilon)$ times that of a shortest path tree where $\epsilon \geq 0$. Alpert et al. [10] proposed AHHK trees as a direct trade-off between Prim's MST algorithm and Dijkstra's shortest path tree algorithm. They used a parameter $0 \leq c \leq 1$ to adjust the preference between tree length and path length.

For deep submicron design, path length is no longer an accurate estimate of path delay. Several attempts have been made to directly optimize Elmore delay taking into account different loading capacitances of the sinks. With exponential timing complexity, the branch and the bound algorithms proposed by Boese et al. [11, 12] provide the optimal and near-optimal solutions that minimize the delay from the source to an identified critical sink or a set of critical sinks of Steiner tree. For a set of critical sinks, it minimizes a linear combination of the sink delays. However it is very difficult to choose the proper weights, or the criticality, for this linear combination. Hong et al. [13] proposed a modified Dreyfus-Wagner Steiner tree algorithm for minimizing the maximal source-to-sink delay, The maximal source-to-sink delay is not necessarily interesting when the corresponding sink is off the critical path. Also, there may be more than one critical sink in the same net associated with multiple critical paths. Prasitjutrakul and Kubitz [14] proposed an algorithm for maximizing the minimal delay slack, where the delay slack is defined as the difference between the real delay and the given delay bound at a sink.

2.3 Buffered Tree Construction

Intermediate buffer insertion creates another degree of freedom for interconnect optimization. Early works on fanout optimization problem focused on the construction of buffered trees during logic synthesis [15, 16, 17] without taking into account the wiring effect. Recently, layout driven fanout optimization have been proposed [18, 19]. For a given Steiner tree, a polynomial time dynamic programming algorithm was proposed in [20] for the delay-optimal buffer insertion problem. Using dynamic programming, Lillis et al. [21] integrated wire sizing and power minimization with the tree construction under a more accurate delay model taking signal slew into account. Inspired by the same dynamic programming algorithm, Okamoto and Cong [22] proposed a simultaneous Steiner tree construction and buffer insertion algorithm. Later the work was extended to include wire sizing [23]. In the formulation of the problem [22, 23], the main objective is to maximize the required arrival time at the root of the tree, which is defined as the minimum among the differences between the arrival time of the sinks and the delay from the root to the sinks.

To achieve optimal delay, multiple buffers may be necessary for a single edge. An early work of S. Dhar and M. Franklin [24] developed the optimal solution for the size, number and position of buffers driving a uniform line that minimizes the delay of the line. The work further considered the area occupied by the buffers as a constraint. Recently C. Alpert and A. Devgan [25] calculated the optimal number of equally spaced buffers on a uniform wire to minimize the Elmore delay of the wire.

2.4 Delay Minimized vs. Delay Bounded

Since timing driven floorplanning and placement are usually iterated with static timing analysis tools, the critical path information is often available and the timing requirement for critical sinks converges as the design and layout progresses. It is sufficient to have bounded delay rather than minimized delay. On the other hand, the minimization of total wire length is of interest since total wire length contributes to circuit area and routing congestion. In addition, total wire capacitance contributes a significant factor to the switching power. The reduction of wire length reduces circuit area and improves routability, also reduces power consumption, which are important factors for manufacturing cost and fabrication yield [1]. In this paper, instead of minimizing the source to sink delays, we will present an algorithm that constructs buffered spanning trees to minimize the total wire length subject to timing constraints.

Zhu [26] proposed the "Delay Bounded Minimum Steiner Tree" (DBMST) algorithm to construct a low cost Steiner tree with bounded delay at critical sinks. The DBMST algorithm consists of two phases: (1) initialization of Steiner tree subject to timing constraints and (2) iterative refinement of the topology to reduce the wiring length while satisfying the delay bounds associated with critical sinks. Since the Elmore delays at sinks are very sensitive to topology and they have to be recomputed every time the topology is changed, DBMST algorithm searches all possible topological updates exhaustively at each iteration and so it is very time consuming.

2.5 Overview of DBB-tree Algorithm

In this paper, we formulate the new Delay Bounded Buffered tree (DBB-tree) problem as follows: Given a signal net and delay bounds associated with critical sinks, construct a routing tree with intermediate buffers inserted to minimize the total wiring length and the number of buffers while satisfying the delay bounds. Based on Elmore delay, we develop an efficient algorithm for DBB spanning tree construction.

The DBB-tree algorithm consists of three phases: (1) Calculate the minimum Elmore delay for each critical sink to allow immediate exclusion of floorplanning/placement solutions that are clearly infeasible from a timing perspective; (2) Construct a buffered spanning tree to minimize the total wire length subject to the bounded delay; (3) Based on the topology obtained in (2), delete unnecessary buffers without violating timing constraints to minimize the total number of buffers. The overall time complexity of DBB-tree algorithm is $O(kn^2)$, where k is the maximum number of buffers inserted on a single edge, and n the number of sinks in the net. Our DBB-tree algorithm makes the following three major contributions:

- Treating the delay bounds provided by static timing analysis tools as constraints rather than formulating the delay into the optimization objectives.
- Constructing a spanning tree and placing intermediate buffers simultaneously. The algorithm is very effective to minimize both wire length and the number of buffers.
- Allowing more than one buffer to be inserted on each single edge and calculating the precise buffer positions for the optimal solution. In contrast, most previous work assumes at most one buffer is inserted for each edge and the buffer location is fixed.

3 Description of DBB-tree Algorithm

For floorplanning purpose, we assume uniform wire width. In the DBB-tree algorithm presented here, we consider only non-inverting buffers. However, the algorithm can be easily extended to handle inverting buffers. Given a signal net $S = \{s_0, s_1, \cdots, s_n\}$, s_0 is the source and $s_1, \cdots, s_n$ sinks. The geometric location for each terminal of S is determined by floorplanning. Let $\vec{B} = (t_b, r_b, c_b)$ denote the vector describing the parameters of non-inverting buffers, in which t_b, r_b and c_b are the internal delay, resistance and capacitance of each buffer respectively. Before presenting the detailed DBB-tree algorithm, we first state some theoretical results developed by Alpert and Devgan [25] which will be used to calculate the number and position of identical buffers placed on a single edge to minimize the edge delay in DBB-tree algorithm:

Theorem 1 *Given a uniform line $e(0, i)$ connecting sink s_i to source s_0, and the parameter vector $\vec{B}$, the number of buffers placed on the wire to obtain the minimum Elmore delay of*

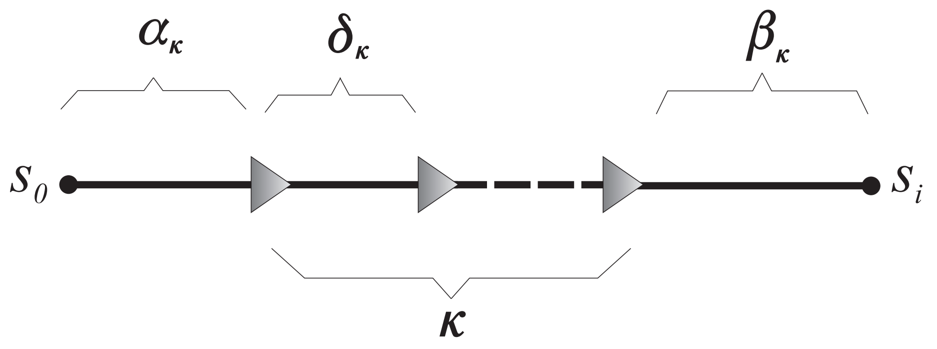

Figure 1: Given a uniform line $e(0,i)$ connecting sink s_i to source s_0, $\kappa(0,i)$ buffers are placed on the wire in such way that the wire delay is minimized: the first buffer is α_κ away from source s_0, the distance between two adjacent buffers equals to δ_κ and the last buffer is β_κ away from sink s_i.

e is given by:

$$\kappa(0,i) = \max(\lfloor -\frac{1}{2} + \sqrt{1 + \frac{2(r(cl_{0,i} + c_b - c_i) - c(r_b - R_0))^2}{rc(r_b c_b + t_b)}} \rfloor,\ 0). \tag{2}$$

where R_0 is the driver output resistance at source s_0 and c_i the loading capacitance at sink s_i. Given κ buffers inserted on $e(0,i)$, the optimal placement of buffers which obtains the minimum wire delay is places the buffers at equal spacing from each other. Let α_κ be the distance from the source to the first buffer, δ_κ the distance between two adjacent buffers, and β_κ the distance from the last buffer to sink s_i. They can be derived as follows:

$$\begin{aligned} \alpha_\kappa(0,i) &= \frac{1}{\kappa+1}\,(l_{0,i} + \frac{t_b(r_b - R_0)}{r} + \frac{c_i - c_b}{c}), \\ \delta_\kappa(0,i) &= \frac{1}{\kappa+1}(l_{0,i} - \frac{r_b - R_0}{r} + \frac{c_i - c_b}{c}), \\ \beta_\kappa(0,i) &= l_{0,i} - \alpha_\kappa - (\kappa - 1)\delta_\kappa. \end{aligned} \tag{3}$$

The minimized wire delay with κ buffers is given by:

$$\begin{aligned} \tau_\kappa(0,i) &= \kappa t_b + \frac{1}{\kappa+1}\,(rl_{0,i}(\kappa c_b + c_i) + cl_{0,i}(R_0 + \kappa r_b) +\ (\kappa c_b + c_i)(\kappa r_b + R_0)) + \\ & \frac{1}{2(\kappa+1)}\,(rcl_{0,i}^2 + \frac{\kappa r(c_b - c_i)^2}{c} - \frac{\kappa c(r_b - R_0)^2}{r}). \end{aligned} \tag{4}$$

If $\kappa - 1$ buffers instead of κ buffers are placed on wire e the wire delay will be increased by:

$$\Delta\tau_\kappa(0,i) = \frac{(rcl_{0,i} + r(c_i - c_b) + c(R_0 - r_b))^2}{2\kappa(\kappa+1)rc} - t_b - r_b c_b. \tag{5}$$

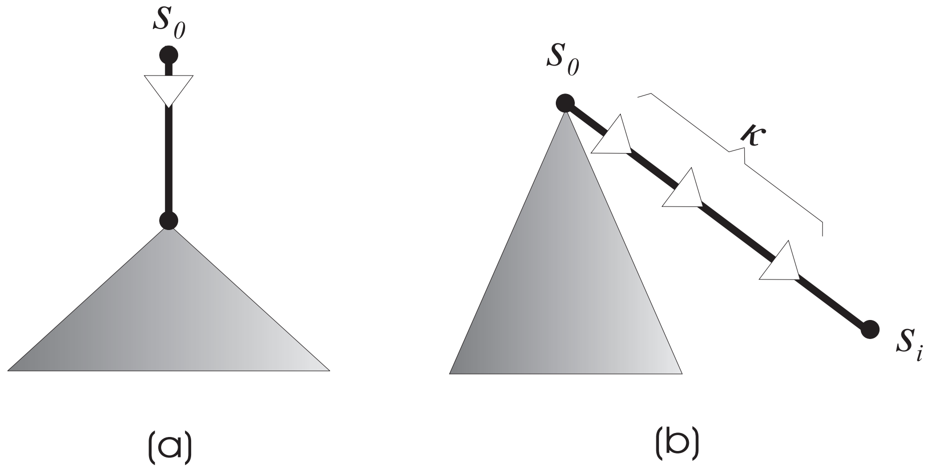

Figure 2: If we place a buffer right after s_0 as in (a), the total capacitance driven by the driver at source is reduced to c_b and the first term of $\tau(0,i)$ equals to $R_0 c_b$. The second term, the propagation delay of the path from source to s_i, can be minimized by directly connecting s_i to the source and placing $\kappa(0,i)$ buffers on the wire as in (b). Combining (a) and (b), we calculate the lower bound of Elmore delay for s_i.

By replacing R_0 with 0, Equations 2 – 5 can be applied to the wire connecting any two sinks in routing tree T. Based on the theoretical results discussed above, we will present the detailed DBB-tree algorithm in the following section.

3.1 Lower Bound of Elmore Delay for Critical Sinks

The first phase of DBB-tree algorithm calculates the lower bound of Elmore delay for each sink s_i. It may not be possible to achieve this delay simultaneously for all sinks, but no achievable delay will exceed it. The floorplanning is timing infeasible if there exists s_i in S such that the lower bound $\tau^*(0,i)$ is greater than the given delay bound D_i: $\tau^*(0,i) > D_i$. The first term in Eq.1, $R_0 C_0$, can be reduced to $R_0 c_b$ by placing a buffer right after s_0 as shown in Fig. 2 (a). And the second term, the propagation delay of the path from source to s_i, can be minimized by directly connecting source to s_i and placing buffers as shown in Fig. 2 (b). Formally, the lower bound of Elmore delay for s_i can be given by:

$$\tau^*(0,i) = \begin{cases} R_0 c_b + \tau_\kappa(0,i) & \text{if } c_b \le (cl_{0,i} + c_i) \\ R_0(cl_{0,i} + c_i) + \tau_\kappa(0,i) & \text{otherwise} \end{cases} \qquad (6)$$

If for all sinks in S, the lower bound of Elmore delay is less than the given delay bound, then the algorithm continues to phases 2 and 3, otherwise the timing constraints are too

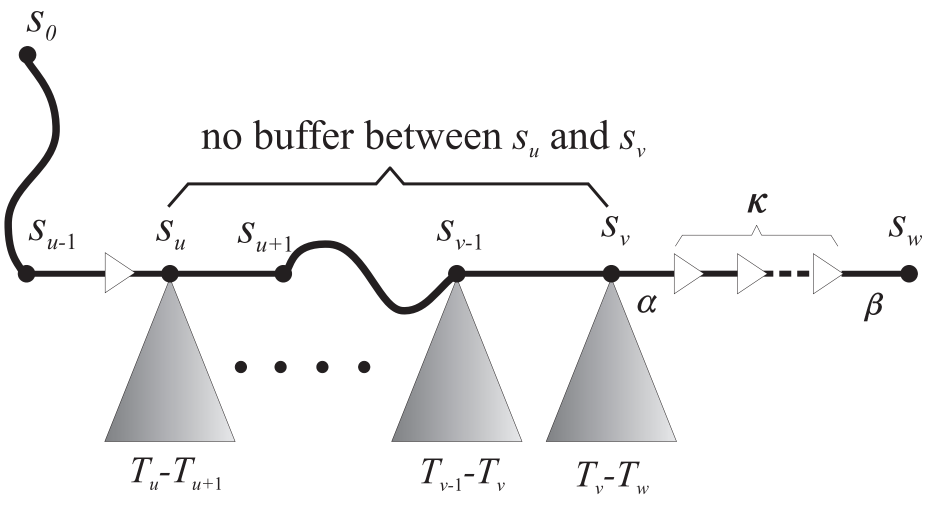

Figure 3: For particular sink $s_v \in T$, edge $e(u-1,u)$ is the last buffered edge on the path from the source to s_v and the last buffer on edge $e(u-1,u)$ drives T_v through the resistance between the buffer and s_v, defined as driving resistance of T_v, denoted by $R(T_v)$. Since there is no buffer between s_u and s_v, the driver of T_v also drives T_i for $i = u, u+1, \cdots, v-1$, where $s_{u+1}, \cdots, s_{v-1}$ are the intermediate sinks from s_u to s_v. After adding the new edge $e(v,w)$, the loading capacitance of T_v is increased by ΔC_v, the Elmore delay of sinks in $T_i - T_{i+1}$ for $i = u, u+1, \cdots, v$, will be increased by $R(T_i)\Delta C_v$. On the other hand, due to the buffers on edge $e(u-1,u)$, ΔC_v will not affect on the delay of sinks which are not in T_u. Therefore the timing constraints of T will be satisfied if and only if the timing constraints of T_u are satisfied.

tight for the given floorplanning and the solution is excluded.

3.2 DBB Spanning Tree Construction

The second phase of DBB-tree algorithm constructs a buffered spanning tree to minimize the total wire length subject to the timing constraints. Similar with Prim's MST algorithm, it starts with the trivial tree: $T = \{s_0\}$. Iteratively edge $e(v,w)$ with $\kappa(v,w)$ buffers is added into T, where $s_v \in T$ and $s_w \in S - T$ are chosen such that $l_{v,w}$ is minimized and timing constraints are satisfied. T grows incrementally until it spans all terminals of S, or there is no edge $e(v,w)$ that can be added without violating the timing constraints. In the later case, the floorplanning is considered to be timing infeasible and the solution is excluded.

For the incremental construction of the DBB-tree, the key issue is how to quickly evaluate the timing constraints each time a new edge is added, i.e. whether or not the delay bound at each critical sink is satisfied. For particular edge $e(v, w)$ where $s_v \in T$ and $s_w \in S - T$, the number and the precise positions of buffers inserted on the edge which minimize the edge delay can be calculated according to Equations 2 and 3. Let T_v denote the subtree rooted at s_v, after adding edge $e(v, w)$ into T, the loading capacitance of T_v, is increased by ΔC_v:

$$\Delta C_v = \begin{cases} cl_{v,w} + c_w & \text{if } \kappa(v, w) = 0, \\ c\alpha_\kappa(v, w) + c_b & \text{otherwise.} \end{cases} \tag{7}$$

Let $e(u-1, u)$ denote the last buffered edge on the path from the source to s_v as shown in Fig. 3, the last buffer on edge $e(u-1, u)$ drives T_v. If there is no buffer from the source to s_v, the source drives T_v. According to Elmore delay, T_v is driven through the resistance between the driver and s_v, defined as *driving resistance* of T_v, denoted by $R(T_v)$. Given s_{v-1} is the parent of s_v, $R(T_v)$ can be calculated as follows:

$$R(T_v) = \begin{cases} R(T_{v-1}) + rl_{v-1,v} & \text{if } \kappa(v-1, v) = 0, \\ r_b + r\beta_\kappa(v-1, v) & \text{otherwise.} \end{cases} \tag{8}$$

Since there is no buffer on the path from s_u to s_v, the driver of T_v also drives T_i for $i = u, u+1, \cdots, v-1$, where $s_{u+1}, \cdots, s_{v-1}$ are the intermediate sinks from s_u to s_v as shown in Fig. 3. Let $T_i - T_{i+1}$ denote the set of sinks in subtree T_i but not in T_{i+1}. Due to the increased loading capacitance ΔC_v of T_v, the Elmore delay of sinks in $T_i - T_{i+1}$ for $i = u, u+1, \cdots, v$, is given by:

$$\forall s \in T_i - T_{i+1}, \quad \Delta\tau(0, s) = R(T_i)\Delta C_v \quad \textit{for } i = u, u+1, \cdots, v. \tag{9}$$

On the other hand, due to the buffers on edge $e(u-1, u)$, the increased loading capacitance of T_v will not affect on the delay of sinks which are not in T_u. We define the *delay slack* of a sink $s \in T$ as:

$$\lambda(s) = D_s - \tau(0, s), \tag{10}$$

and the delay slack of T_i to be:

$$\lambda(T_i) = \min_{s \in T_i} \lambda(s) \tag{11}$$

the timing constraints will be satisfied for the sinks in $T_u - \{s_w\}$ if and only if the following condition holds:

$$\lambda(T_i) \geq R(T_i)\Delta C_v \quad \textit{for } i = u, u+1, \cdots, v. \tag{12}$$

By introducing the *loading capacitance slack* of each subtree T_i:

$$\sigma(T_i) = \frac{\lambda(T_i)}{R(T_i)} \tag{13}$$

Eq. 12 can be rewritten as:

$$\sigma(T_i) \geq \Delta C_v \quad for\ i = u, u+1, \cdots, v. \tag{14}$$

Let $\sigma^*(v)$ denote the minimum slack of loading capacitance among the subtrees T_i for $i = u, u+1, \cdots, v$:

$$\sigma^*(v) = \min_{i=u,u+1,\cdots,v} \sigma(T_i). \tag{15}$$

the condition in Eq. 14 can be simply rewritten as:

$$\sigma^*(v) \geq \Delta C_v. \tag{16}$$

By keeping track of $\sigma^*(v)$, this condition can be checked in constant time. The Elmore delay of s_w can be calculated from the Elmore delay of s_v:

$$\tau(0, w) = \tau(0, v) + R(T_v)\Delta C_v + \tau_\kappa(v, w) \tag{17}$$

where $\tau_\kappa(v, w)$ is calculated from Eq. 4 and the timing bound at s_w can also be checked in constant time. From above analysis, we can conclude that the **necessary and sufficient** condition for satisfying the timing constraints of T after adding the new edge $e(v, w)$ is:

$$\sigma^*(v) \geq \Delta C_v \quad and \quad D_w \geq \tau(0, w), \tag{18}$$

and this condition can be checked in constant time.

At each iterative step of DBB-tree construction, $s_v \in T$ and $s_w \in S - T$ can be selected in linear time such that $l_{v,w}$ is minimum and the timing constraints are satisfied. After adding the new edge $e(v, w)$, a two-pass traversal of T is sufficient to update the delay slack and loading capacitance slack of each subtree in T: (1) traverse T bottom up and calculate the delay slack and loading capacitance slack of each subtree T_i according to Equations 11 and 13; (2) traverse T top down and calculate $\sigma^*(i)$ from $\sigma^*(i-1)$, given s_{i-1} is the parent of s_i:

$$\sigma^*(i) = \begin{cases} \sigma(i) & \text{if } \kappa(i-1, i) > 0, \\ \min(\sigma^*(i-1),\ \sigma(i)) & \text{otherwise.} \end{cases} \tag{19}$$

Since each new edge can be added into T in linear time, the overall DBB spanning tree can be constructed in $O(n^2)$ time for net S with n sinks.

3.3 Buffer Deletion

In phase 2, one or more buffers are inserted on each edge to minimize wire delay. Some of the buffers may not be necessary for meeting the delay bound. The third phase of the

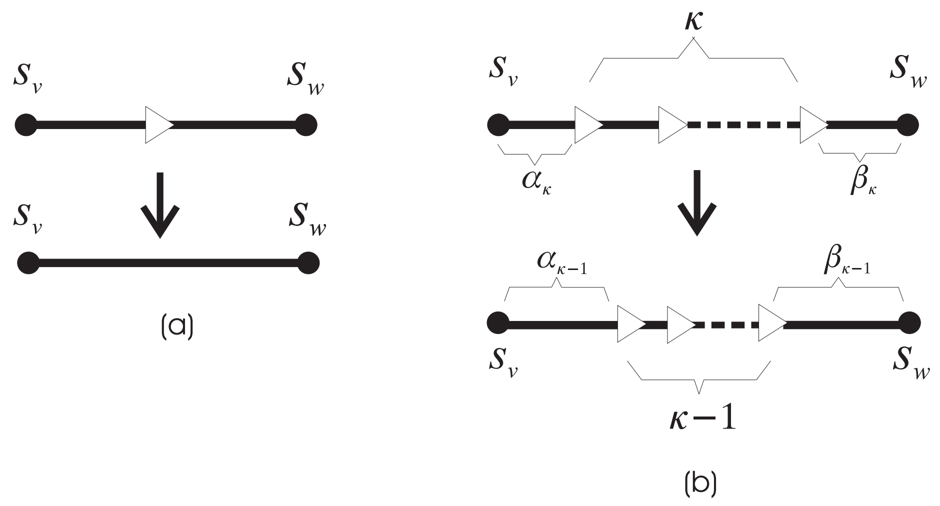

Figure 4: In case of $\kappa(v,w)=1$ as shown in (a), edge $e(v,w)$ becomes unbuffered edge after deleting the buffer, the load capacitance of subtree T_v is increased by: $\Delta C_v = cl_{v,w} + C_w - c\alpha_\kappa - c_b$; otherwise $\kappa(v,w)-1>0$ buffers are re-inserted on $e(v,w)$, as shown (b): $\Delta C_v = c(\alpha_{\kappa-1}-\alpha_\kappa)$.

DBB-tree algorithm deletes buffers from the spanning tree obtained in the second phase to reduce the total number of buffers. In general the buffers closest to the source can unload the critical path the most. The algorithm traverses T bottom up and deletes one buffer at a time without violating timing constraints. The deletion continues until all the buffers left in T are necessary, that is, the timing constraints would not be satisfied if one more buffer is deleted.

For particular edge $e(v,w)$ with $\kappa>0$ buffers, if one buffer is deleted from $e(v,w)$, this wire delay will be increased by $\Delta\tau_\kappa(v,w)$ according to Eq. 5, and $\kappa-1$ buffers will be re-inserted: $\alpha_\kappa \to \alpha_{\kappa-1}$ and $\beta_\kappa \to \beta_{\kappa-1}$. In case of $\kappa=1$ as shown in Fig. 4 (a), wire $e(v,w)$ becomes unbuffered edge after deleting the buffer, the load capacitance of subtree T_v is increased by: $\Delta C_v = cl_{v,w}+C_w - c\alpha_\kappa - c_b$; otherwise $\kappa-1>0$ buffers are re-inserted on edge $e(v,w)$, as shown in Fig. 4 (b): $\Delta C_v = c(\alpha_{\kappa-1}-\alpha_\kappa)$.

Similar to phase 2, let $e(u-1,u)$ denote the last buffered edge from the source to s_v. The delay of the sinks in subtree T_u will be increased due to the increased loading capacitance of T_v. In addition, the delay of sinks in subtree T_w will be further increased due to the increased edge delay of $e(v,w)$. Based on the analysis in phase 2, a buffer can be deleted without causing timing violation if and only if following condition holds:

$$\sigma^*(v) \geq \Delta C_v \quad and \quad \lambda(T_w) \geq R(T_v)\Delta C_v + \Delta\tau_\kappa(v,w) \tag{20}$$

Table 1: Experimental Parameters of DBB-tree Algorithm on Signal Nets

Output Resistance of Driver	R_0	$500\Omega - 1000\Omega$
Unit Wire Resistance	c	$0.12\Omega/\mu m$
Unit Wire Capacitance	r	$0.15fF/\mu m$
Output Resistance of Buffer	r_b	500Ω
Loading Capacitance of Buffer	c_b	$0.05pF$
Intrinsic Delay of Buffer	t_b	$0.1ns$
Loading Capacitance of Sink	c_i	$0.05pF - 0.15pF$

Therefore the timing constraints of T can be evaluated in constant time for deleting a buffer from edge $e(v, w)$. The buffer can be found by searching at most $n-1$ edges. After deleting a buffer, the delay slack and loading capacitance slack of subtrees in T are incrementally updated in $O(n)$ time as in phase 2. So one buffer will be deleted in linear time. There are at most kn buffers in T where k is the maximum number of buffers on single edge, the timing complexity of buffer deletion is $O(kn^2)$ which dominates the overall DBB-tree algorithm. Following experimental results show that the buffer deletion effectively minimizes the total number of buffers and it can delete more than 90% of the buffers inserted in the previous phase.

4 Experimental Results

In the first part of the experiments, we implemented the DBB spanning tree algorithm on a Sun SPARC 20 workstation under the C/UNIX environment. The algorithm was tested on signal nets with $2, 5, 10, 25, 50$ and 100 pins. For each net size, 100 nets were randomly generated on a $10mm \times 10mm$ routing region, and we report the average results. The driver output resistance at the source and the loading capacitances of sinks are randomly chosen from the intervals $[500\Omega, 1000\Omega]$ and $[0.05pF, 0.15pF]$ respectively. The parameters used in the experiments are based on [22], which are summarized in Table 1.

The average results of the DBB spanning tree construction are shown in Table 2. The delay bounds of critical sinks for each net size are randomly chosen from the interval titled "Delay Bounds". The average wire length and number of buffers for DBB spanning tree are reported in this table. The average CPU time consumed per net shows that DBB spanning tree algorithm is fast enough that can be applied during the stochastic optimization.

Table 2: Experimental Results of DBB Spanning Trees on Signal Nets

Pins(#)	Delay Bounds(ns)	Wire Length(mm)	Buffers(#)	CPU ($sec.$)
2	1.0 - 5.0	4.94	0.23	0.0004
5	1.0 - 5.0	13.80	1.43	0.0019
10	1.5 - 5.0	25.07	2.82	0.0049
25	2.0 - 5.0	45.79	4.57	0.0816
50	2.5 - 5.0	78.12	7.15	0.6259
100	3.0 - 5.0	123.30	10.53	4.9228

To evaluate the DBB spanning trees generated by the experiments, we constructed both minimum spanning tree (MST) and shortest path tree (SPT) for the same signal nets using the same parameters. The comparison of the average results is shown in Table 3. "DBB/MST" and "DBB/SPT" is the average length ratio of DBB-tree to MST and DBB-tree to SPT respectively. The column "% sinks meeting bound" gives the average percentage of critical sinks which satisfy the delay bounds. For the nets with small number of terminals, the length of DBB-tree is very close to MST. As the number of terminals in the nets increases, the length of DBB-tree to MST is increased, but only 9% through 0% critical sinks can meet the bound in MST for 25-pin through 100-pin nets. It can be concluded that it is very difficult to satisfy the timing constraints using MST especially for the large nets. On the other hand, the length ratio of DBB-tree to SPT is decreased from 1.0 down to 0.24, and SPT is also not ideal to meet the delay bounds for the large nets. The DBB-tree approach can achieve the short wire length with 100% critical sinks meeting the delay bounds.

In Table 4, the average number of buffers inserted in DBB spanning trees are listed and the result is very reasonable considering the number of terminals in the net. To evaluate the buffer deletion algorithm, we compare the average number of buffers inserted in DBB spanning tree before and after buffer deletion. The percentage of buffers reduced by the third phase of DBB-tree algorithm is as high as 79% through 93%. The results presented in Table 4 demonstrate that the third phase of the algorithm is quite effective at removing any unnecessary buffers estimated during phase 2 and the DBB-tree algorithm will not lead to unrealistic, impractical results.

In the second part of the experiments, we apply DBB-tree to evaluate the wiring delay of floorplanning solutions considered by the Genetic Simulated Annealing method [27]. Table

Table 3: Comparison of DBB-tree, MST and SPT of Signal Nets.

Pins (#)	Legnth (mm)					% sinks meeting bound		
	DBB	MST	DBB/MST	SPT	DBB/SPT	DBB	MST	SPT
2	4.94	4.94	1.00	4.94	1.00	100	100	100
5	13.80	12.96	1.07	20.08	0.70	100	93	96
10	25.07	20.54	1.22	44.98	0.56	100	76	92
25	45.79	34.15	1.34	118.26	0.39	100	9	31
50	78.12	48.01	1.63	249.79	0.32	100	0	0.06
100	123.30	67.50	1.83	513.72	0.24	100	0	0

Table 4: Average Number of Buffers Before vs. After Buffer Deletion.

Pins(#)	w/o Deletion	with Deletion	Reduced (%)
2	3.24	0.23	92.90
5	7.09	1.43	79.83
10	13.99	2.82	79.84
25	36.34	4.57	87.42
50	79.61	7.15	91.02
100	171.24	10.53	93.85

Table 5: Four Examples of Floorplanning Applying DBB-tree Algorithm.

Blocks (#)	Block size (mm)	Aspect ratio of blocks	Nets (#)	Net size (#pins/net)	Delay bound (ns)	CPU ($min.$)
10	0.5 - 2.0	0.8 - 1.2	50	2 - 10	0.5 - 2.0	3.8
25	0.5 - 2.0	0.8 - 1.2	75	2 - 20	1.0 - 5.0	10.7
50	0.5 - 2.0	0.8 - 1.2	150	2 - 25	1.5 - 7.5	105.4
100	0.5 - 2.0	0.8 - 1.2	250	2 - 50	2.5 - 10.0	477.2

Table 6: Achieved Floorplanning Solutions by Using DBB-tree, MST and SPT Approaches.

Blocks	Area(mm^2)			Length(mm)			% sinks meeting bound		
(#)	DBB	MST	SPT	DBB	MST	SPT	DBB	MST	SPT
10	18.55	26.92	23.48	112.08	128.60	211.62	100	96.2	96.9
25	52.30	72.38	61.01	511.27	579.87	910.53	100	88.1	92.5
50	112.59	148.62	124.38	1455.47	1801.92	2696.10	100	94.4	97.7
100	213.57	274.77	274.02	6039.93	7037.06	16339.61	100	90.82	95.61

5 presents four examples which includes 10, 25, 50 and 100 rectangular blocks, respectively. The sizes (widths and heights) and aspect ratios of blocks are randomly chosen within a nominal range. Netlists are also randomly generated for the four examples. The technology parameters are consistent with those shown in Table 1.

To compare with the traditional approaches which do not consider buffer insertion during the floorplanning, we also apply MST and SPT methods to evaluate the floorplanning solution in the same examples. Based on the same stochastic search strategy, the floorplanning solutions achieved by the three methods are shown in Table 6. Similarly, the column "% sinks meeting bound" measures the percentage of critical sinks which satisfy the tim-

Table 7: The Improvement by Considering Buffer Insertion in Floorplanning Stage.

Blocks	Area Improvement(%)		Wire Length Improvement(%)		Buffers(#)
(#)	DBB vs. MST	DBB vs. SPT	DBB vs. MST	DBB vs. SPT	in DBB
10	31.10	21.01	14.71	47.04	18
25	27.74	14.28	13.42	43.85	18
50	24.24	9.48	19.23	46.02	24
100	22.27	22.06	14.17	63.04	15

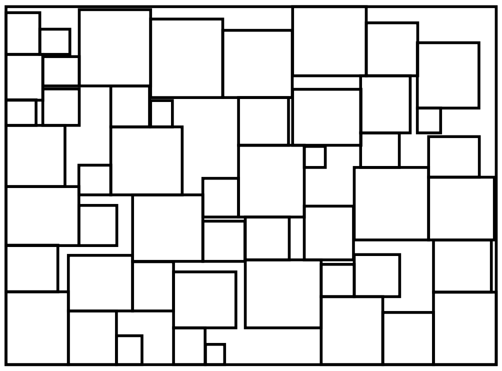

Figure 5: Floorplanning of 50 blocks with 150 nets sized from 2-pin to 25-pin. SPT is applied to evaluated the wiring delay. Achieved chip area is $124.38mm^2$ and total wire length $2696.10mm$ with 97.7% critical sinks meeting the delay bounds.

ing bounds. Table 7 calculates the improvement of both chip area and total wire length by using DBB-tree method. For the examples, the area can be improved up to 31% over MST and 22% over SPT, respectively. On the other hand, the total wire length can be improved up to 19% over MST and 63% over SPT, respectively. This substantial improvement demonstrates that using buffer insertion at the floorplanning stage yields significantly better solutions in terms of both chip area and total wire length. In addition, the total number of buffers estimated by the DBB-tree approach are also shown in this table. Figures 5 and 6 show the floorplanning solution with 50 blocks by using SPT and DBB-tree algorithm, respectively. In addition, Fig. 6 also displays the buffers estimated by DBB-tree approach. It should be noted that future research is needed to extend the approach to distribute buffers into the empty space between macros subject to timing constraints. However, the area of such buffers is typically a small fraction of a given macro area and can be typically accommodated.

5 Conclusion

In this paper, we propose a new methodology of floorplanning and placement where intermediate buffer insertion is used as another degree of freedom in the delay calculation. An efficient algorithm to construct Delay Bounded Buffered(DBB) spanning trees has been developed. One of the key reasons this approach is effective is that we treat the delay bounds as constraints rather than formulating the delay into the optimization objectives as is done

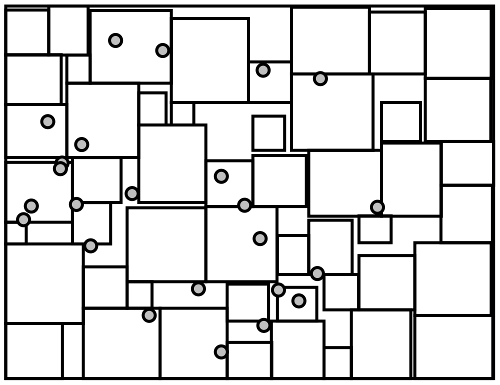

Figure 6: Floorplanning of the same example in Fig. 5. DBB-tree is applied to evaluate the wiring delay. Achieved chip area is $112.59mm^2$ and total wire length $1455.47mm$ with 100% critical sinks meeting the delay bounds. The area and total wire length are improved by 9.48% and 46.02% respectively. The dots shown in the figure represent the buffers estimated by DBB-tree.

in most of the previous work. In fact, our problem formulation is more realistic for the path based timing driven layout design. The timing constraints of a floorplan are evaluated many times during our stochastic optimization process. The efficient DBB spanning tree algorithm made our buffered tree based floorplanning and placement highly effective and practically applicable to industrial problems.

References

[1] J. Cong, L. He, C.-K. Koh, and P. H. Madden, *Integration, the VLSI Journal 21 (1996) 1-94.* Elsevier Science B.V., 1996.

[2] W. C. Elmore, "The transient response of damped linear networks with particular regard to wide-band amplifiers," *J. Appl. Phys.*, vol. 19, pp. 55–63, 1948.

[3] J. P. Cohoon and L. J. Randall, "Critical net routing," in *Proc. IEEE Intl. Conf. on Computer Design*, pp. 174–177, 1991.

[4] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Performance-Driven global routing for cell based IC's," in *Proc. IEEE Intl. Conf. Computer Design*, (Cambridege, MA), pp. 170–173, October 1991.

[5] A. B. Kahng and G. Robins, "A new class of iterative steiner tree heuristics with good performance," *IEEE Trans. on Computer-Aided Design*, vol. 11, pp. 893–902, July 1992.

[6] 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,” in *Proc. of IEEE Intl. Symp. on Circuits and Systems*, pp. 1869–1872, 1993.

[7] J. Cong, K. Shing Leung, and D. Zhou, “Performance-Driven interconnect design based on distributed RC delay model,” in *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 606–611, June 1993.

[8] A. Lim and S. Wing Cheng, “Performance oriented rectilinear steiner trees,” in *Proc. of 30th Design Automation Conf.*, pp. 171–176, June 1992.

[9] J. M. Ho, D. J. Lee, C. H. Chang, and C. K. Wong, “Bounded-diameter spanning tree and related problems,” in *Proc. ACM Symp. on Computational Geometry*, pp. 276–282, 1989.

[10] C. J. Alpert, T. C. Hu, J. H. Huang, A. B. Kahng, and D. Karger, “Prim-Dijkstra tradeoffs for improved performance-Driven routing tree design,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 14, pp. 890–896, July 1995.

[11] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, “Rectilinear steiner trees with minimum elmore delay,” in *Proc. 31st ACM/IEEE Design Automation Conf.*, pp. 381–387, June 1994.

[12] K. D. Boese, A. B. Kahng, and G. Robins, “High-Performance routing trees with identified critical sinks,” in *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 182–187, June 1993.

[13] X. Hong, T. Xue, E. S. Kuh, C. K. Cheng, and J. Huang, “Performance-Driven steiner tree algorithms for global routing,” in *Proc. 30th ACM/IEEE Design Automation Conf.*, (Baltimore, MD), pp. 177–181, June 1993.

[14] S. Prasitjutrakul and W. J. Kubitz, “A timing-Driven global router for custom chip design,” in *IEEE Intl. Conf. on Computer Aided Design*, pp. 48–51, 1990.

[15] K. J. Singh and A. Sangiovanni-Vincentelli, “A heuristic algorithm for the fanout problem,” in *Proc. ACM/IEEE Design Automation Conf.*, pp. 357–360, 1990.

[16] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang, “Performance oriented technology mapping,” in *Proc. 6th MIT VLSI Conf.*, pp. 79–97, 1990.

[17] C. L. Berman, J. L. Carter, and K. F. Day, “The fanout problem: From theory to practice,” in *Proc. 1989 Decennial Caltech Conf.*, pp. 69–99, 1989.

[18] L. N. Kannan, P. R. Suaris, and H. G. Fang, “A methodology and algorithms for post-Placement delay optimization,” in *Proc. ACM/IEEE Design Automation Conf.*, pp. 327–332, 1994.

[19] H. Vaishnav and M. Pedram, “Routability-Driven fanout optimization,” in *Proc. ACM/IEEE Design Automation Conf.*, pp. 230–235, 1993.

[20] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal elmore delay," in *Proc. International Symposium on Circuits and Systems*, pp. 865–868, 1990.

[21] J. Lillis, C. Kuan Cheng, and T. Ting Y. Lin, "Optimal and efficient buffer insertion and wire sizing," in *Proc. IEEE 1995 Custom Integrated Circuits Conf.*, pp. 259–262, 1995.

[22] T. Okamoto and J. Cong, "Interconnect layout optimization by simultaneous steiner tree construction and buffer insertion," in *Proc. 5th ACM/SIGDA Physical Design Workshop*, (Reston, Virginia), pp. 1–6, April 1996.

[23] T. Okamoto and J. Cong, "Buffered steiner tree construction with wire sizing for interconnect layout optimization," in *Proc. 1996 IEEE/ACM International Conf. on Computer Aided Design*, (San Jose, CA), pp. 44–49, Nov. 1996.

[24] S. Dhar and M. A. Franklin, "Optimum buffer circuits for driving long uniform lines," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 32–40, January 1991.

[25] C. Alpert and A. Devgan, "Wire segmenting for improved buffer insertion," in *to appear in Proc. of 34th Design Automation Conf.*, June 1997.

[26] Q. Zhu, *Chip and Package Co-Synthesis of Clock Networks.* PhD thesis, Univ. of California, Santa Cruz, Santa Cruz, CA, June 1995.

[27] S. Koakutsu, M. Kang, and W. W.-M. Dai, "Genetic simulated annealing and application to non-slicing floorplan design," in *Proc. 5th ACM/SIGDA Physical Design Workshop*, (Virginia, USA), pp. 134–141, April 1996.