On the Skew-Bounded Minimum-Buffer Routing Tree Problem*

Christoph Albrecht,† Andrew B. Kahng, Bao Liu, Ion Măndoiu, and Alexander Zelikovsky‡

CSE Department, UCSD, La Jolla, CA 92093-0114

†Research Institute for Discrete Mathematics, University of Bonn, Lennstr. 2, 53113 Bonn, Germany

‡CS Department, Georgia State University, Atlanta, GA 30303

albrecht@or.uni-bonn.de, {abk,bliu,mandoiu}@cs.ucsd.edu, alexz@cs.gsu.edu

Abstract—Bounding the load capacitance at gate outputs is a standard element in today's electrical correctness methodologies for high-speed digital VLSI design. Bounds on load caps improve coupling noise immunity, reduce degradation of signal transition edges, and reduce delay uncertainty due to coupling noise [6]. For clock and test distribution, an additional design requirement is bounding the *buffer skew*, i.e., the difference between the maximum and the minimum number of buffers over all source-to-sink paths in the routing tree, since buffer skew is one of the main factors affecting sink delay skew [10]. In this paper we consider algorithms for buffering a given tree with the minimum number of buffers under given load cap and buffer skew constraints. We show that the greedy algorithm proposed by Tellez and Sarrafzadeh [10] is suboptimal for non-zero buffer skew bounds and give examples showing that no bottom-up greedy algorithm can achieve optimality. The main contribution of the paper is an optimal dynamic programming algorithm for the problem. Experiments on test cases extracted from recent industrial designs show that the dynamic programming algorithm has practical running time and inserts up to 5–10% fewer buffers compared to the algorithm in [10].

I Introduction

For high-speed digital VLSI design, bounding the load capacitance at gate outputs is a standard element in today's *electrical correctness* methodologies. Bounds on load caps improve coupling noise immunity, reduce degradation of signal transition edges, and reduce delay uncertainty due to coupling noise [6].[1] According to [9], commercial EDA methodologies and tools for signal integrity rely heavily on upper-bounding the capacitive loads on driver and buffer outputs (to prevent very long slew times on signal transitions). Essentially, the load capacitance bounds serve as *proxies* for bounds on input rise/fall times at buffers and sinks (Tellez and Sarrafzadeh [10] formally prove this equivalence using a simple linear model). We assume that such capacitive load bounds are inherent to any buffered routing tree design task. It is natural to propose a *minimum-buffer* formulation, so as to minimize changes made to the routing tree in meeting the load bounds.

Buffering to control slew times is also critical to *early timing analysis*. With lookup-table based modeling of gate delays and output transition times, very long input slews tend to be propagated inaccurately, resulting in extremely slow transitions. Static timing analyses that are based on the associated delay calculations will be utterly compromised, and useless for driving performance optimizations. Thus, early timing analysis must start with a buffering solution that bounds the capacitive loads of all buffers and of the source driver. Again, a *minimum-buffer* objective is appropriate.

Last, we observe that buffering of some large routing trees (e.g., for clock and test distribution) is further constrained with respect to the *buffer skew*, i.e., the difference between the maximum and the minimum number of buffers over all source-to-sink paths in the routing tree [10]. This is because buffer skew reflects the actual buffered clock tree skew after routing. To accurately estimate tradeoffs between alternative clock tree topologies in the early stages of clock distribution design, the key problem is to bound the number of buffers needed by a given tree to satisfy given constraints on *both slew rate* (input rise/fall times) *and buffer skew*. Good bounds (or, good constructions that minimize the number of buffers while controlling the buffer skew) will enable accurate estimation and tradeoff of such system resources as power and area.

From the above context and assumptions, we obtain the following problem formulation:

Bounded Skew Buffering Problem (BSBP): Given a (clock) net N, per-unit length wire capacitance, sink and buffer input capacitances, capacitive load bounds for

*This work was partially supported by Cadence Design Systems, Inc., the MARCO Gigascale Silicon Research Center and NSF Grant CCR-9988331.

[1] Such bounds also improve reliability with respect to hot-carrier oxide breakdown (hot electrons) [4, 5] and AC self-heating in interconnects [8], and facilitate technology migration since designs are more balanced.

buffers and for the tree source, and an upper bound Δ on buffer skew, find a buffering of N that satisfies all bounds while using the minimum number of buffers.

The BSBP was first formulated by Tellez and Sarrafzadeh [10], who suggested a greedy algorithm with runtime $O(n+k)$, where n is the number of sinks in the net N and k is the number of inserted buffers. In this paper, we make the following contributions:

- We give examples showing the sub-optimality of the Tellez-Sarrafzadeh algorithm for BSBP with non-zero skew bounds, and further prove that no bottom-up greedy algorithm can achieve optimality (Section III).
- We give a non-trivial dynamic programming algorithm which guarantees optimum solutions for BSBP in $O(n(\Delta+1)^3 NB^2)$ time, where n, Δ, and NB are the number of sinks, the given skew bound, and an upper-bound on the optimum number of inserted buffers, respectively (Section IV).
- We present experimental results on test cases extracted from recent industrial designs, showing that the dynamic programming algorithm has practical running time and inserts significantly fewer buffers compared to the algorithm in [10] (Section V).

II Notations and Problem Formulation

We start with a few definitions and notations. Let N be a *net* consisting of a *source* r and a set of *sinks* S.

- A *routing tree* for the net N is a binary[2] tree $T = (r, V, E)$ rooted at r such that each sink of S is a leaf in T.
- A *buffered routing tree* for the net N is a tree $T' = (r, V, E, B)$ such that $T = (r, V, E)$ is a routing tree for N and B is a set of buffers located on the edges[3] of T.
- For any $b \in B \cup \{r\}$, the *subtree driven by* b, D_b, (also referred to as the *stage* of b [10]) is the maximal subtree of T which is rooted at b and has no internal buffers; a buffered routing tree $T = (r, V, E, B)$ has $|B| + 1$ stages including a *source stage* driven by the source.

Throughout the paper we will use the following notations:

$n = |S|$ number of sinks

[2] In this paper we restrict ourselves to binary routing trees. Every routing tree can be made binary by duplicating nodes and inserting zero-length edges.

[3] We assume that buffers have a single input and a single output and thus are inserted only on the edges of T.

$C_w =$ capacitance of a wire of unit length, which is assumed to be the same for all wires

$C_b =$ buffer input capacitance, assumed to be the same for all buffers[4]

$C_U =$ given upper-bound on the capacitive load of each buffer and of the source driver

$c_v =$ input capacitance of sink or buffer v

$l_e =$ length of wire segment e

$c_e =$ capacitance of wire segment e, i.e., $c_e = C_w l_e$

$T_v =$ subtree of T rooted at v

$c(T_v) =$ lumped capacitance of T_v, i.e., $c(T_v) = \sum_{e \in T_v} c_e + \sum_{v \in leaves(T_v)} c_v$

$l(T_v) =$ maximum number of buffers on a path from v to a sink $s \in T_v$ (called longest path for short)

$s(T_v) =$ minimum number of buffers on a path from v to a sink $s \in T_v$ (shortest path)

$\delta(T_v) = l(T_v) - s(T_v)$ (buffer skew of T_v)

Load Constraints
As noted in [10], bounded slew rate can be ensured by upper-bounding the lumped capacitive load of each buffer and of the source driver. The *lumped capacitive load* of $b \in B \cup \{r\}$ is given by

$$c(D_b) = \sum_{e \in D_b} c_e + \sum_{v \in leaves(D_b)} c_v \qquad (1)$$

Thus, to ensure bounded slew rate we require that

$$c(D_b) \le C_U \quad \text{for every} \quad b \in B \cup \{r\} \qquad (2)$$

Buffer Skew Constraints
Tellez and Sarrafzadeh [10] also note that the buffer skew is a significant factor affecting sink delay skew. Other sources of sink delay skew, such as propagation delays, have been well studied (heuristics and approximation algorithms for constructing *unbuffered* trees with zero- or bounded-skew can be found, e.g., in [3, 12]). To guarantee bounded sink delay skew after buffering we need to ensure that the difference in the number of buffers of the longest and shortest path from the root r to the sinks is at most a given buffer skew bound Δ, i.e.,

$$\delta(T) = l(T) - s(T) \le \Delta \qquad (3)$$

A buffering satisfying both the load constraint (2) and the buffer skew constraint (3) will be called *feasible*. In

[4] We assume that a single type of buffer is used. Using a single buffer type is a widely accepted design strategy since it reduces process variation sensitivity and facilitates technology migration.

this paper we consider the problem of finding a feasible buffering with minimum number of buffers, formally defined as follows:

Bounded Skew Buffering Problem (BSBP)
Given: (1) net N with source r and set of sinks S, (2) binary routing tree $T = (r, V, E)$ for N, (3) sink input capacitances c_s, $s \in S$, (4) buffer input capacitance C_b, (5) unit-length wire capacitance C_w, (6) load upper-bound C_U, and (7) buffer-skew bound Δ,
Find: buffering $T' = (r, V, E, B)$ of T such that:

(a) $c(D_b) \leq C_U$ for every $b \in B \cup \{r\}$,

(b) $\delta(T') \leq \Delta$, and

(c) the total number of inserted buffers, $|B|$, is minimum subject to (a) and (b).

For every $v \in V$ the *branch* of v, denoted $br(v)$, is $T_v \cup (v, parent(v))$ (where $parent(r) = r$). For each buffering X of a branch $br(v)$, we denote by $nb(X)$, $l(X)$, $s(X)$, $cap(X)$, and $\delta(X)$ the total number of buffers, the number of buffers on the longest path, the number of buffers on the shortest path, the residual capacitance (i.e., the capacitance of the stage driven by $parent(v)$), and the buffer skew in the branch $br(v)$, respectively. Also, if X is a buffering of a subtree containing vertex v, we denote by X_v the buffering X restricted to the branch $br(v)$.

III Why Greedy Does Not Work

The BSBP has been previously studied by Tellez and Sarrafzadeh [10]. In [10], a greedy algorithm is first presented for minimum buffering *without* buffer skew constraints and then the algorithm is modified to handle such constraints. Below we describe the two algorithms for the case of binary trees; the description in [10] is given for arbitrary trees.

When there are no constraints on buffer skew, the algorithm in [10] starts with an empty buffering $X = \emptyset$ and then performs the following two steps for each node u, in bottom-up order:

1. $packNode(u)$: while $cap(X_v) + cap(X_w) > C_U$ (where v and w are the two children of u), add a buffer at the topmost position of the child branch with the largest residual capacitance (the greedy choice).
2. $packEdge(u)$: while $cap(X_u) > C_U$, add a buffer on the edge $(u, parent(u))$, at the highest possible position still meeting the load cap bound C_U.

With buffer skew constraints, $packEdge$ remains the same while the modified *packNode-BS(u)* consists of the following four steps:

1. Balance T_u as follows. If $l(X_v) < l(X_w)$ then swap v and w. If $l(X_v) - s(X_w) > \Delta$ then insert $l(X_v) - s(X_w) - \Delta$ buffers at the topmost position of $br(w)$. Exit if $cap(X_u) \leq C_U$.

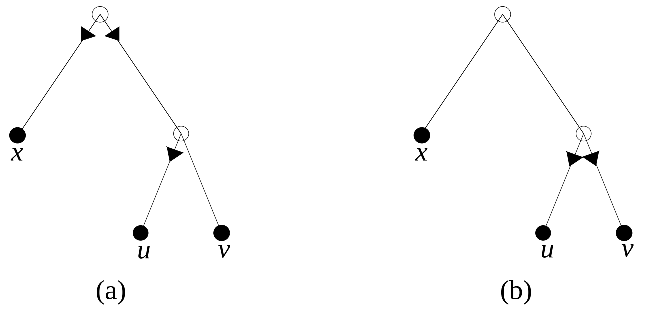

Fig. 1. A counterexample for the greedy BSBP algorithm in [10].

2. Perform $packNode(u)$ excluding the child branches with maximum longest path, i.e., if $l(X_w) < l(X_v)$, then add a buffer at the topmost position in $br(w)$. Exit if $cap(X_u) \leq C_U$.
3. Insert buffers at the topmost position of all child branches with shortest path equal to $l(u) - \Delta$ (in order to maintain buffer skew at most Δ when we insert buffers on the longest paths in the next step). Exit if the load constraint is satisfied.
4. Perform $packNode(u)$ considering only child branches with maximum longest path, i.e., longest path equal to $s(u) + \Delta + 1$.

The modified greedy algorithm finds the optimum solution of any given tree when the skew bound Δ is zero. However, contrary to the claim made in [10], the modified greedy algorithm may give suboptimal solutions for $\Delta \geq 1$. There several reasons for its sub-optimality. One reason is that child branches with maximum longest path are considered for buffering *after* considering the other branches, regardless of their residual capacitance. This may cause the algorithm to return a suboptimal solution, e.g., when the skew bound Δ is so large that the buffer skew constraint never becomes a constraint (in this case the optimum is found by always choosing the branch with the largest residual capacitance in $packNode$).

Figure 1 shows another small instance for which the Tellez-Sarrafzadeh algorithm fails to find the optimal buffering. In this instance we have $\Delta = 1$, $C_w = C_b = 0$, and sink input capacitances are given by $c_u = C_U$ and $c_x = c_v = \frac{3}{4}C_U$. Figure 1 (a) shows the solution computed by the greedy algorithm and Figure 1 (b) shows the optimal solution which has one less buffer. This instance points to a more basic reason for the sub-optimality of the modified greedy algorithm: the optimum buffering of a given tree may be suboptimal when restricted to subtrees.

A natural question prompted by the example in Figure 1 is whether or not there exists a bottom-up algorithm that computes a *fixed* number of solutions for each branch and still guarantees global optimality. Below, we give two series of examples showing that the answer to this question is negative.

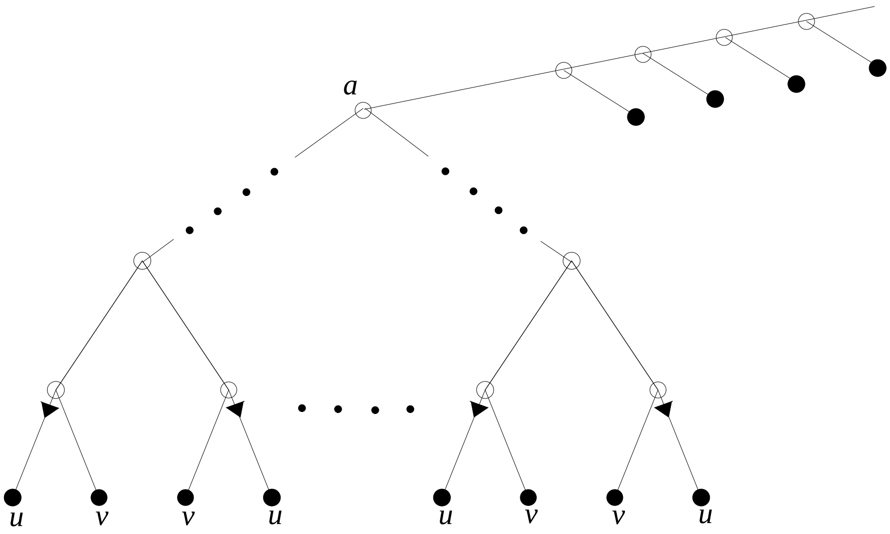

Fig. 2. Proof of Claim 1.

Claim 1 *To guarantee optimality, every bottom-up buffering algorithm may need to compute branch bufferings with $m, m+1, \ldots, m+k$ buffers respectively, where m is the minimum number of buffers for the branch, and unbounded k.*

Claim 1 follows from the example in Figure 2, in which $\Delta = 1$ and $C_w = C_b = 0$. Each pair of sibling leaves contains a "u" leaf and a "v" leaf, with capacitances of $c_u = C_U$ and c_v, respectively, where $c_v 2^{d-2} < C_U$ and $c_v(2^{d-2}+1) > C_U$ and d is the depth of T_a.

The minimum number of buffers for each of the two branches into a is 2^{d-2}, since buffers are only required by the "u" leaves. If we start with minimum-number bufferings for both branches into a, we will have to insert a buffer right below a on one of them (in order to meet the load constraints). This in turn triggers the insertion of a very large number of buffers upstream due to the skew constraint. The optimum overall solution is to insert buffers right above 2^{d-2} of the "v" leaves. This leads to buffering one of the branches into a with at least $\frac{3}{2}2^{d-2}$ buffers.

Claim 2 *To guarantee optimality, every bottom-up buffering algorithm may need to compute branch bufferings with longest path equal to $l, l+1, \ldots, l+\Delta-1$, respectively, where l is the minimum longest path.*

Claim 1 follows from the example in Figure 3, in which there are $n = \Delta$ "u" leaves, each with capacitance $c_u = C_U - \epsilon$, and one additional "v" leaf, with capacitance $c_v = \epsilon$. Assume that the wire capacitance C_w and buffer input capacitance C_b are both 0. In this case, any optimum bottom-up buffering algorithm must compute the n bufferings shown in the figure, with longest paths of $n, n-1, \ldots 1$ buffers, respectively. None of these solutions is dominated by the others since those with smaller longer path have larger residual capacitance, and, depending on the upstream tree topology, each of them may be the only way to complete the optimal solution.

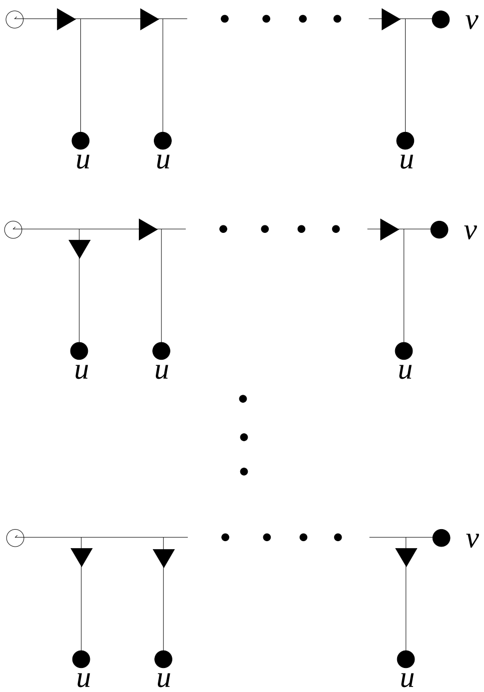

Fig. 3. Proof of Claim 2.

IV Dynamic Programming Algorithm

In this section we give a dynamic programming algorithm for the bounded skew buffering problem. The dynamic programming technique has been applied in the past to timing-driven buffer insertion (see e.g., [1, 7, 11]), but its application to BSBP presents specific challenges.

Input: Net N with source r and set of sinks S, binary routing tree $T=(r,V,E)$ for N, input capacitances c_s, $s \in S$, buffer input capacitance C_b, unit-length wire capacitance C_w, load upper-bound C_U, buffer-skew bound Δ, and upper bound NB on the number of buffers in an optimal solution
Output: Minimum size feasible buffering of T

For each $u \in V$, $\mathcal{L}(u) \leftarrow \emptyset$
For each sink $s \in S$ do EdgeBuffering($\emptyset$, s)

For each $u \in V-S$, in bottom-up order (postorder), do

(1) Let v and w be the children of u

(2) For each $X \in \mathcal{L}(v)$ and $Y \in \mathcal{L}(w)$, $l(X) \geq l(Y)$, do

- **(a)** Let Z be $X \cup Y$ with $\max\{0, l(X)-s(Y)\}$ buffers added at the top of $br(w)$
- **(b)** For $i=0,\ldots,\min\{\max\{0,s(X)-s(Y)\}, l(X)-l(Y)\}$ do
  - Let Z_i be Z with i buffers at the top of $br(w)$
  - EdgeBuffering(Z_i, u)

(3) Remove from $\mathcal{L}(u)$ bufferings with more than NB buffers

(4) For each buffering with parameters (nb,l,s), remove from $\mathcal{L}(u)$ all bufferings with parameters $(nb+k, l+k, s+k)$, where $k \geq 2$

Return the buffering $X \in \mathcal{L}(r)$ with minimum $nb(X)$

Procedure EdgeBuffering(X, u **)**

While $cap(X) > C_U$, add a buffer on edge $(u, parent(u))$ at the highest position meeting the load cap bound C_U

$\mathcal{L}(u) \leftarrow \mathcal{L}(u) + X$

If $cap(X) > C_b$, then $\mathcal{L}(u) \leftarrow \mathcal{L}(u) + X'$, where X' is X with an additional buffer just below $parent(u)$)

Fig. 4. Dynamic programming BSBP algorithm.

We start by introducing a few more definitions. A buffering X of $br(v)$ is said to be *redundant* if it contains a topmost buffer whose deletion does not increase the residual capacitance. Let X and Y be two bufferings of the same branch. We say that Y *dominates* X if $nb(Y) \leq nb(X)$, $l(Y) \leq l(X)$, $s(Y) \geq s(X)$, and $cap(Y) \leq cap(X)$. In other words, if Y dominates X, then we can always replace X with Y without loosing feasibility or increasing the number of buffers.

The algorithm (see Figure 4) computes for each node $v \neq r$ a set $\mathcal{L}(u)$ of feasible bufferings for $br(v)$. The set $\mathcal{L}(u)$ is constructed by considering all combinations of bufferings from $\mathcal{L}(v_1) \times \mathcal{L}(v_2)$, where v_1 and v_2 are the two children of u. To keep the size of $\mathcal{L}(u)$ bounded the algorithm eliminates redundant bufferings and bufferings using more than NB buffers, where NB is a given upper-bound on the number of buffers in an optimal buffering. An upper-bound NB can be computed in linear time by running the algorithm of Tellez and Sarrafzadeh [10] with skew-bound set to zero (recall that the algorithm in [10] is guaranteed to find the optimum when $\Delta = 0$).

Lemma 1 *For each buffering X of $br(u)$ there exists $k \geq 0$ and buffering $Y \in \mathcal{L}(u)$ such that X is dominated by Y with k buffers added at the top.*

Proof. The proof is by induction on the depth of u. The claim is trivially true if u is a sink, i.e., a leaf of T. Assume that $u \in V-S$. Let v_1, v_2 be the two children of u, and let X_i, $i=1,2$, be the restriction of X to $br(v_i)$. By induction, there exist $k_i \geq 0$ and $Y_i \in \mathcal{L}(v_i)$, $i=1,2$, such that we can replace X_i by Y_i with k_i buffers added on edge (v_i,u) just below u.

If each branch $br(v_i)$ has at least 2 buffers at the top then we can replace the top buffer from each branch with a single buffer on $(u, parent(u))$ just below $parent(u)$. By repeating this procedure we may assume that one of the branches, say $br(v_1)$, has at most one buffer at the top. We claim that the resulting buffering Z of $br(v_1)$ is in $\mathcal{L}(v_1)$. Indeed, if Z does not have a buffer at the top, then $Z = Y_1$. Otherwise, Z is either Y_1 or Y_1 with one buffer at the top, and in the latter case procedure **EdgeBuffering** ensures that $Z \in \mathcal{L}(v_1)$.

Finally, when Algorithm 4 combines $Z \in \mathcal{L}(v_1)$ with $Y_2 \in \mathcal{L}(v_2)$, steps (2a) and (2b) generate bufferings with all feasible skews. □

Lemma 2 *For each node u of T, the set $\mathcal{L}(u)$ computed by Algorithm 4 contains at most $2(\Delta+1)NB$ bufferings.*

Proof. Consider a feasible buffering X with parameters (nb,l,s) and a feasible buffering Y with parameters $(nb+k, l+k, s+k)$, $k \geq 2$. Then Y is dominated by a redundant buffering obtained from X by adding k buffers at the topmost position, and hence Y is removed by step (4) of the algorithm. Thus, $\mathcal{L}(u)$ contains at most two non-redundant bufferings with parameters $(nb+i, l+i)$ for every fixed $nb-l$ and skew $\delta \in \{0,\ldots,\Delta\}$. The lemma follows by observing that all bufferings remaining in $\mathcal{L}(u)$ satisfy $nb - l \leq nb \leq NB$, where the last inequality is ensured by step (3) of the algorithm. □

Theorem 1 *Algorithm 4 returns a feasible buffering with the minimum number of buffers,* OPT. *The running time of the algorithm is $O(n(\Delta+1)^3 NB^2)$, where n, Δ, and NB are the number of sinks, the given skew bound, and the given upper-bound on* OPT.

Proof. Correctness follows from Lemma 1. The running time follows by observing that, for each of the $n-1$ vertices $u \in V-S$, the algorithm needs $O((\Delta+1)^3 n^2)$ time to compute the set $\mathcal{L}(u)$. Indeed, the time needed by steps (2a) and (2b) is $O((\Delta+1)\cdot|\mathcal{L}(v_1)|\cdot|\mathcal{L}(v_2)|)$, where v_1 and v_2 are the two children of u, and by Lemma 2, $|\mathcal{L}(v_i)| = O((\Delta+1)NB)$ for $i = 1, 2$. Step (3) requires $O((\Delta+1)^3 n^2)$ time since at the end of step 2 the size of $\mathcal{L}(u)$ is $O((\Delta+1)^3 n^2)$. Finally, step (4) can be implemented in $O((\Delta+1)^3 n^2)$ by using radix sort to lexicographically order all bufferings in $\mathcal{L}(u)$ with respect to $(nb-l, l-s)$, and then removing all dominated bufferings in one traversal of the sorted list. □

Remark. We conjecture that Lemma 2 can be strengthen by showing that the set $\mathcal{L}(u)$ has size at most $4n(\Delta+1)$. This will imply that the runtime of Algorithm 4 is $O((\Delta+1)^3 n^3 + \mathrm{OPT})$. In practice, significantly shorter lists $\mathcal{L}$ (and hence significantly improved runtime) is achieved by deleting from the lists $\mathcal{L}$ all dominated bufferings (note that Steps (3) and (4) only remove bufferings with more than NB buffers or those dominated by bufferings with $k \geq 2$ buffers added at the top).

V Experimental Results

Both our dynamic programming algorithm and the greedy algorithm of [10] have been implemented in C. Table I gives the results obtained by running the two algorithms on 5 testcases from [2]. In all experiments the initial tree was computed using the Greedy-DME algorithm [3]. The unit wire capacitance was $C_w = 0.177 fF/\mu$m, buffer input capacitance was $C_b = 37.5 fF$, and sink input capacitance varied between $2.04 fF$ and $200 fF$.

The first observation is that, although slower than the greedy algorithm of [10], the dynamic programming has very practical runtime (all testcases finish in less than one second on a SUN Ultra 60 running SunOS 5.7). As expected, both algorithms find the optimum solution when a buffer skew bound of 0 is imposed. For non-zero skew bounds the dynamic programming algorithm is almost always strictly better than the greedy algorithm – the number of saved buffers is often in the 5–10% range.

Table I also shows that a significant reduction in the number of inserted buffers can be achieved with a small increase in buffer skew, e.g., when going from zero buffer skew to a buffer skew of 1. For comparison, we have also included in the table a lower bound on the number of buffers, which is the minimum number of buffers needed to meet the load cap constraints while disregarding buffer skew constraints.[5] In all but one case, the lower bound is matched by the optimum buffering with $\Delta = 4$, and often it is matched with a buffer skew as small as 2.

VI Conclusions and Future Research

In this paper we have addressed the problem of finding the minimum-buffered routing of a given tree under buffer load and skew constraints. We have shown that a greedy algorithm previously proposed for this problem in [10] may fail to find the optimum solution, and we have proposed an exact dynamic programming algorithm. Experimental results on test cases extracted from recent industrial designs show that the dynamic programming algorithm has practical running time and inserts significantly fewer buffers compared to the greedy algorithm of [10].

Our future research will address

(i) multi-constraint formulations, in which, e.g., input capacitance and fanout must be upper-bounded simultaneously,

(ii) minimum inverter insertion in a given tree subject to sink polarity constraints, in addition to inverter load and skew constraints, and

(iii) simultaneous tree construction and buffering under given buffer load and skew constraints.

[5] This lower bound can be computed by running the dynamic programming algorithm with a very large Δ, but we used the much faster (linear time) algorithm given in [2].

References

[1] C. Alpert and A. Devgan. Wire segmenting for improved buffer insertion. In *ACM/IEEE Design Automation Conference*, pages 588–593, 1997.

[2] C. Alpert, A.B. Kahng, B. Liu, I.I. Măndoiu, and A.Z. Zelikovsky. Minimum-buffered routing for slew and reliability control. In *Proceedings IEEE-ACM International Conference on Computer-Aided Design*, 2001 (to appear).

[3] M. Edahiro. Delay minimization for zero-skew routing. In *Proceedings IEEE-ACM International Conference on Computer-Aided Design*, pages 563–567, 1993.

[4] P. Fang, J. Tao, J.F. Chen, and C. Hu. Design in hot-carrier reliability for high performance logic applications. In *IEEE Custom Integrated Circuits Conference*, pages 525–532, 1998.

[5] C. Hu. Hot carrier effects. In N.G. Einspruch, editor, *Advanced MOS Device Physics*, pages 119–160. Academic Press, 1989.

[6] A.B. Kahng, S. Muddu, E. Sarto, and R. Sharma. Interconnect tuning strategies for high-performance ICs. In *Proc. Conference on Design Automation and Test in Europe*, February 1998.

[7] J. Lillis, C.-K. Cheng, and T.-T. Lin. Optimal wire sizing and buffer insertion for low power and a generalized delay model. *IEEE J. Solid-State Circuits*, 31:437–447, 1996.

[8] S. Rzepka, K. Banerjee, E. Meusel, and Chenming Hu. Characterization of self-heating in advanced vlsi interconnect lines based on thermal finite element simulation. *IEEE Transactions on Components, Packaging, and Manufacturing Technology, Part A,*, 21:406–411, 1998.

[9] L. Scheffer. Personal communication, April 2000.

[10] G.E. Tellez and M. Sarrafzadeh. Minimal buffer insertion in clock trees with skew and slew rate constraints. *IEEE Transactions on Computer-Aided Design*, 16:333–342, 1997.

[11] L.P.P.P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 865–868, 1990.

[12] A.Z. Zelikovsky and I.I. Măndoiu. Practical approximation algorithms for zero- and bounded-skew trees. In *Proceedings 12th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 407–416, 2001.

Testcase		$\Delta = 0$		$\Delta = 1$		$\Delta = 2$		$\Delta = 3$		$\Delta = 4$		Lower
#sinks	C_U	Greedy	DP	Greedy	DP	Greedy	DP	Greedy	DP	Greedy	DP	Bound
330	500	**34**	**34**	**31**	**26**	**27**	**25**	**27**	**25**	**27**	**25**	**25**
		0.01	0.03	0.00	0.03	0.00	0.05	0.01	0.06	0.01	0.07	
	1000	**19**	**19**	**17**	**14**	**15**	**12**	**16**	**12**	**16**	**12**	**12**
		0.01	0.01	0.01	0.02	0.00	0.05	0.00	0.06	0.01	0.08	
	2000	**8**	**8**	**7**	**6**	**7**	**5**	**7**	**5**	**7**	**5**	**5**
		0.01	0.01	0.00	0.02	0.00	0.05	0.00	0.06	0.01	0.07	
	4000	**5**	**5**	**4**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
		0.01	0.01	0.01	0.03	0.00	0.04	0.00	0.05	0.01	0.07	
	8000	**2**	**2**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
		0.01	0.01	0.01	0.03	0.00	0.05	0.01	0.06	0.01	0.07	
830	500	**101**	**101**	**91**	**87**	**90**	**83**	**89**	**81**	**88**	**81**	**81**
		0.02	0.03	0.01	0.04	0.01	0.06	0.01	0.09	0.01	0.09	
	1000	**46**	**46**	**45**	**41**	**45**	**38**	**43**	**38**	**42**	**38**	**38**
		0.01	0.02	0.01	0.04	0.01	0.06	0.01	0.07	0.01	0.09	
	2000	**25**	**25**	**23**	**20**	**24**	**19**	**23**	**19**	**23**	**19**	**19**
		0.01	0.02	0.01	0.05	0.01	0.07	0.01	0.09	0.01	0.10	
	4000	**11**	**11**	**9**	**9**	**9**	**9**	**10**	**9**	**10**	**9**	**9**
		0.01	0.02	0.01	0.04	0.01	0.06	0.01	0.07	0.01	0.08	
	8000	**5**	**5**	**5**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
		0.01	0.02	0.01	0.04	0.01	0.06	0.01	0.08	0.01	0.09	
1900	500	**105**	**105**	**90**	**84**	**87**	**79**	**87**	**79**	**85**	**78**	**78**
		0.03	0.07	0.03	0.13	0.02	0.21	0.03	0.27	0.02	0.33	
	1000	**53**	**53**	**46**	**42**	**43**	**40**	**43**	**39**	**44**	**39**	**39**
		0.03	0.05	0.03	0.11	0.03	0.19	0.03	0.26	0.03	0.31	
	2000	**26**	**26**	**23**	**20**	**22**	**19**	**21**	**19**	**21**	**19**	**19**
		0.02	0.05	0.03	0.12	0.03	0.19	0.03	0.24	0.03	0.29	
	4000	**14**	**14**	**12**	**10**	**11**	**9**	**11**	**9**	**11**	**9**	**9**
		0.03	0.05	0.03	0.11	0.03	0.17	0.03	0.24	0.03	0.29	
	8000	**6**	**6**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
		0.03	0.05	0.03	0.12	0.03	0.17	0.03	0.23	0.03	0.28	
2400	500	**133**	**133**	**121**	**105**	**115**	**102**	**116**	**100**	**116**	**99**	**99**
		0.04	0.09	0.03	0.16	0.03	0.26	0.03	0.36	0.03	0.43	
	1000	**62**	**62**	**54**	**50**	**52**	**48**	**52**	**47**	**52**	**47**	**47**
		0.03	0.06	0.03	0.15	0.03	0.24	0.03	0.33	0.04	0.40	
	2000	**29**	**29**	**26**	**24**	**26**	**23**	**25**	**23**	**25**	**23**	**23**
		0.03	0.06	0.03	0.15	0.04	0.22	0.03	0.31	0.04	0.39	
	4000	**16**	**16**	**15**	**12**	**15**	**10**	**14**	**10**	**14**	**10**	**10**
		0.04	0.07	0.04	0.14	0.04	0.22	0.04	0.31	0.04	0.38	
	8000	**9**	**9**	**8**	**5**	**7**	**5**	**7**	**5**	**7**	**5**	**5**
		0.03	0.07	0.04	0.14	0.04	0.22	0.04	0.30	0.04	0.37	
2600	500	**266**	**266**	**238**	**211**	**229**	**204**	**226**	**198**	**227**	**196**	**196**
		0.04	0.14	0.03	0.33	0.04	0.60	0.03	0.82	0.04	1.02	
	1000	**125**	**125**	**117**	**104**	**109**	**99**	**106**	**98**	**106**	**98**	**97**
		0.03	0.10	0.04	0.27	0.03	0.50	0.04	0.71	0.04	0.87	
	2000	**64**	**64**	**55**	**50**	**52**	**49**	**52**	**48**	**52**	**48**	**48**
		0.03	0.10	0.04	0.29	0.03	0.50	0.04	0.69	0.04	0.86	
	4000	**34**	**34**	**30**	**26**	**29**	**23**	**28**	**22**	**28**	**22**	**22**
		0.03	0.10	0.04	0.28	0.04	0.50	0.04	0.70	0.04	0.88	
	8000	**15**	**15**	**15**	**12**	**13**	**11**	**13**	**11**	**13**	**11**	**11**
		0.04	0.11	0.04	0.28	0.06	0.48	0.04	0.66	0.05	0.81	

TABLE I Number of buffers (boldface) and runtime for the dynamic programming and greedy BSBP algorithms.