Minimum Buffered Routing With Bounded Capacitive Load for Slew Rate and Reliability Control

Charles J. Alpert, *Member, IEEE*, Andrew B. Kahng, Bao Liu, Ion I. Măndoiu, and Alexander Z. Zelikovsky

***Abstract*—In high-speed digital VLSI design, bounding the load capacitance at gate outputs is a well-known methodology to improve coupling noise immunity, reduce degradation of signal transition edges, and reduce delay uncertainty due to coupling noise. Bounding load capacitance also improves reliability with respect to hot-carrier oxide breakdown and AC self-heating in interconnects, and guarantees bounded input rise/fall times at buffers and sinks.**

This paper introduces a new *minimum-buffer routing problem* (MBRP) formulation which requires that the capacitive load of each buffer, and of the source driver, be upper-bounded by a given constant. Our contributions are as follows:

- **We give linear-time algorithms for optimal buffering of a given routing tree with a single (inverting or noninverting) buffer type.**
- **For simultaneous routing and buffering with a single noninverting buffer type, we prove that no algorithm can guarantee a factor smaller than 2 unless P = NP and give an algorithm with approximation factor slightly larger than 2 for typical buffers. For the case of a single inverting buffer type, we give an algorithm with approximation factor slightly larger than 4.**
- **We give local-improvement and clustering based MBRP heuristics with improved practical performance, and present a comprehensive experimental study comparing the runtime/quality tradeoffs of the proposed MBRP heuristics on test cases extracted from recent industrial designs.**

NOMENCLATURE

C_w	Capacitance of a wire segment of unit length, assumed to be the same for all wires.
C_b	Input capacitance of the given buffer type.
c_v	Input capacitance of sink or buffer v.
σ_v	Input signal polarity of sink or inverting 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., $c(T_v) = \sum_{e \in T_v} c_e + \sum_{v \in leaves(T_v)} c_v$.
C_U	Given upper-bound on the capacitive load of each buffer.

Manuscript received March 6, 2002; revised June 25, 2002. This work was supported in part by Cadence Design Systems, Inc., in part by the MARCO Gigascale Silicon Research Center, in part by the National Science Foundation under Grant CCR-9988331, in part by the Moldovan Research and Development Association (MRDA) under Award MM2-3018, in part by the U.S. Civilian Research & Development Foundation for the Independent States of the Former Soviet Union (CRDF), and in part by the State of Georgia's Yamacraw Initiative. This paper was recommended by Associate Editor M. D. F. Wong.

C. J. Alpert is with the IBM Corporation, Austin, TX 78758 USA (e-mail: alpert@austin.ibm.com).

A. B. Kahng, B. Liu, and I. I. Măndoiu are with the Departments of Computer Science and Engineering, and of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: abk@cs.ucsd.edu; bliu@cs.ucsd.edu; mandoiu@cs.ucsd.edu).

A. Z. Zelikovsky is with the Computer Science Department, Georgia State University, Atlanta, GA 30303 USA (e-mail: alexz@cs.gsu.edu).

Digital Object Identifier 10.1109/TCAD.2002.807888

I. INTRODUCTION

IN HIGH-SPEED digital VLSI design, bounding the load capacitance at gate outputs is a well-known part of 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 [13]. According to [21], commercial EDA methodologies and tools for signal integrity rely heavily on upper-bounding the load caps of drivers and buffers to prevent very long slew times on signal transitions. Such buffer insertions for long or high-fanout nets *are for electrical—not timing optimization—reasons*.[1] Essentially, load cap bounds serve as proxies for bounds on input rise/fall times at buffers and sinks. Although slew time is not completely determined by capacitive loads, Tellez and Sarrafzadeh [24] show experimentally the strong correlation between them. Bounded capacitive loads also improve reliability with respect to hot-carrier oxide breakdown (hot electrons) [10], [12] and AC self-heating in interconnects [20], and facilitate technology migration since designs are more balanced.

In this work, we do not address the well-studied problem of buffer insertion for timing optimization. Instead, we focus on the very practical and immediate requirement of *electrical correctness in large interconnects*—a requirement that arises *before* timing optimization even starts. The motivating observation is that any design flow requires early elimination of all electrical violations (i.e., load cap or slew)—*even for noncritical nets*—as a prerequisite to initiating meaningful placement and timing optimizations. In other words, until electrical correctness is established, timing analyses are meaningless and layout/timing optimizations cannot begin. Several reasons for this are as follows. 1) Gates are well-characterized only for particular cap load ranges, and applying table lookups plus extrapolations in the timing tools will result in garbage transition times for loads outside these ranges. 2) Any inaccurate slew time caused by a cap load violation will propagate through the timing graph and cause misleading values downstream. 3) Until all slew time and cap load violations are fixed, static timing analysis results

[1]For signal integrity purposes buffer insertion should also *lower bound* the capacitive load of drivers and buffers, since a driver that is too strong relative to its load will result in too sharp a transition, creating a stronger aggressor to neighboring potential victim nets. Our algorithms can be extended to simultaneously ensure that the capacitive load of each buffer is *at least half* the given load upper-bound (see Lemma 3).

cannot be trusted and the quality of a floorplan or placement cannot even be evaluated meaningfully.

To make progress with any methodology, it is crucial to have a fast and resource-efficient method for fixing electrical violations. Of particular interest are practical methods for otherwise noncritical nets that have up to *tens of thousands of sinks* (e.g., scan enable). Again, such nets are not timing-critical, but timing and layout optimizations require their efficient buffering for electrical correctness. We observe the following.

- Even if buffers have been inserted by synthesis to honor cap load bounds, the synthesis tool's buffer insertion is layout-oblivious. These buffers must be ripped out and recalculated from the placement, analogous to how synthesized clock and scan structures are treated in modern flows.
- In buffering for electrical correctness, it suffices to use a single buffer and/or inverter type with reasonable drive strength. One buffer type has been shown to be sufficient to yield good results in timing optimization [4]. (Optimization of buffer drive strengths can also be performed during later power/timing optimization phases.)
- Since one just wants to quickly fix violations without using too many resources, minimizing the total wire and buffer area is a suitable objective. A simplified objective is to minimize the number of inserted buffers, which also minimizes the number of placement perturbations required to accommodate the buffers.

These observations motivate the problem addressed in this paper, informally formulated as follows:

Minimum-Buffered Routing Problem (MBRP): Given a net N, sink input capacitances, and an (inverting) buffer type, find a minimum-cost (polarity obeying) buffered routing tree for N such that the capacitive load of each buffer and of the source is at most a given upper bound.

A. Previous Work

The vast amount of research on buffer insertion can be roughly divided into three categories.

Fanout Optimization During Logic Synthesis: Works in this category (see, e.g., [6], [8], [17], [23]) seek buffered routing *topologies* and focus on timing optimization. Since placement information is not available at the logic synthesis stage, the delay models used in these works consist mainly of gate delay and statistically inferred interconnect delay. In contrast, our work is targeted to the early postplacement phases of the design cycle.

Timing-Driven Buffer Insertion During Routing: Works in this category concentrate on buffering *timing-critical* nets, e.g., maximizing the required arrival time (RAT) at the source, often with no bounds on the number of buffers, power consumption, or area. The seminal work of Van Ginneken [25] proposed a dynamic programming approach to finding the optimum buffering of an already routed net, using identical buffers and at most one buffer per wire. Lillis *et al.* [15], [16] extended the dynamic programming approach by incorporating slew effects into the delay model and performing simultaneous buffer insertion and wire sizing; they also considered formulations that seek to minimize area or power consumption subject to meeting given timing constraints. More recently, Alpert and Devgan [1] gave extensions to multiple buffers per wire, and Alpert, Devgan and Quay [2] extended the approach to simultaneous noise and delay optimization. Okamoto and Cong [18] considered simultaneous routing and buffer insertion, showing that significant delay reductions can be achieved over previous approaches which insert buffers into an already routed net. These techniques are appropriate for buffered routing of (relatively small) timing-critical nets, but not for upper-bounding slew rates in *noncritical* nets: (1) quadratic or worse runtimes reduce their applicability to large (tens of thousands of sinks) instances; (2) timing-driven objectives such as max RAT at the source, and reliance on unavailable or meaningless timing analyzes and constraints, lead to wasted resources (too many buffers inserted); and (3) minimizing area or power subject to RAT constraints as in [15], [16] cannot guarantee that slew constraints will be met.

Clock-Tree Buffering: Work on buffered clock trees has focused on delay [22] and skew minimization [9], [19]. Tellez and Sarrafzadeh [24] considered minimal buffer insertion in *routed* clock trees with skew and slew constraints. They argued that slew upper-bounds can be met by upper-bounding the lumped capacitive loads of the buffers, and gave a linear time algorithm for buffering a routed clock tree with a single noninverting buffer type under these constraints. We differ from [24] in several respects. 1) We seek simultaneous routing and buffering, while [24] considers only the problem of buffering an already routed clock tree. 2) Besides noninverting buffering, we also consider buffering with a single inverting buffer type, which requires handling additional sink polarity constraints (the number of inverting buffers on each source-to-sink path must be consistent with the given polarity of the sink). 3) Clock trees in [24] require bounded buffer skew—this constraint is not necessary in our application.

B. Our Contributions

Our contributions are as follows.

- We give linear-time algorithms for optimal buffering of a given routing tree with a single (inverting or noninverting) buffer type.[2]
- For simultaneous routing and buffering with a single noninverting buffer type, we prove that no algorithm can guarantee a factor smaller than 2 unless P $=$ NP and give an algorithm with approximation factor slightly larger than 2 for typical buffers. For the case of a single inverting buffer type, we give an algorithm with approximation factor slightly larger than 4.
- We give local-improvement and clustering based MBRP heuristics with improved practical performance, and present a comprehensive experimental study comparing the runtime/quality tradeoffs of the proposed MBRP heuristics on test cases extracted from recent industrial designs.

C. Organization of the Paper

We formally define MBRP in Section II. Then, in Section III, we describe two exact *linear-time* algorithms for buffering a given routing tree: a greedy algorithm for buffering with a

[2]A different algorithm for noninverting buffers was previously given in [24].

noninverting buffer type and a dynamic programming algorithm for buffering with an inverting buffer type. In Section IV we analyze the approximation complexity of MBRP and give provably-good approximation algorithms for both inverting and noninverting buffer types. We give local-improvement and clustering heuristics with improved practical performance in Section V, and present experimental results comparing the runtime/quality tradeoffs of the proposed heuristics in Section VI. We conclude in Section VII with directions for future research.

II. Problem Formulation

We start with basic 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 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 of T.[3]
- For any $b \in B \cup \{r\}$, the *subtree driven by* b, also referred to as the *stage* of b [24], is the maximal subtree D_b 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.

Load Model: We use the *lumped capacitive load* model, in which the load of a buffer b is given by

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

Load Constraints: As noted in [24], bounded slew rate can be ensured by upper-bounding the lumped capacitive load of each buffer $b \in B$ and of the source driver r. Formally, we require that

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

Cost Functions: The cost of a buffered routing tree T is measured by the total wire and buffer area. Denoting the area of each buffer by a, the *combined cost* of the buffered routing $T = (r, V, E, B)$ can be expressed as follows:

$$combined\_cost(T) = wire\_area(T) + |B| \cdot a. \quad (1)$$

The wire area of T depends on the wirelength in each metal layer and the number of vias. During early post-placement phases of the design cycle the wire area still cannot be estimated very accurately, since layer assignment and via information is not yet available. Therefore, we assume that each stage requires the same amount of routing resources and define the simplified routing cost as the number of stages in the buffered routing T, i.e.,

$$cost(T) = |B| + 1. \quad (2)$$

Thus, in this paper we adopt the simplified cost measure (2):

Minimum-Buffered Routing Problem (MBRP): Given a net N with source r and set of sinks S (with prescribed polarities), input capacitance c_s for every sink $s \in S$, buffer input capacitance C_b, unit-length wire capacitance C_w, and load upper-bound $C_U > 2C_b$,[4] find a buffered routing tree $T = (r, V, E, B)$ for N such that

a) $c(D_b) \le C_U$ for every $b \in B \cup \{r\}$;
b) (for inverting buffer type) the parity of the number of buffers on each path from the source to any positive sink is the same, and opposite from the parity of the number of buffers on the paths from the source to any negative sink;
c) $cost(T) = |B|+1$ is minimum among all buffered routing trees satisfying conditions a) and b).

III. Exact Algorithms for Buffering Routed Nets

In this section, we present two algorithms for optimally buffering an already routed net using a single inverting or noninverting buffer type. The running time of each algorithm is linear in the number of sinks and the number of inserted buffers.

A. Single Noninverting Buffer Type

Our algorithm for buffering a given routing tree with a single noninverting buffer type is a generalization of a greedy algorithm for partitioning node-weighted trees due to Kundu and Misra [14]. Like in [14], we traverse the tree in bottom-up order, inserting "fully loaded" buffers, i.e. buffers that drive a subtree with total capacitance equal to C_U. If no fully loaded buffer can be inserted then we must have reached a node p with subtree capacitance greater than C_U such that the capacitance of each child branch is strictly less than C_U. In this case we greedily insert the most loaded buffer, i.e., the buffer at the top of the child branch with highest capacitance.

Before formally describing the algorithm we need to introduce two more definitions. Let $T = (r, V, E)$ be a routing tree. A vertex p of T is called *critical* if p is a bottom-most point of T such that T_p cannot be driven by a single buffer. Formally, p is critical if $c(T_p) > C_U$ and $c(T_u) \le C_U$ for every child u of p. A *heaviest child* u of p is one which accumulates more capacitance than any other child of p. Formally, u is a heaviest child of p if $c(T_u) + c_{(u,p)} \ge c(T_v) + c_{(v,p)}$ for every other child v of p.

The algorithm (see Fig. 1) finds critical vertices by a post-order traversal of the input tree. Then, for every such critical vertex p, the algorithm repeatedly inserts buffers on the edge connecting p to its heaviest child, until p is no longer critical. For simplicity of analysis we give here a recursive implementation of the algorithm.

Remark: The runtime of the algorithm in Fig. 1 is $O(|S| \cdot |B|)$ (since the tree is traversed once for each inserted buffer). An optimal $O(|S| + |B|)$ time implementation inserts all buffers in a single bottom-up traversal; see [3] for the full details.

Theorem 1: The algorithm in Fig. 1 finds an optimum buffering of the input tree T with the given noninverting buffer type.

The proof of the theorem follows from the following two lemmas, corresponding to the two possible cases in Step 3 of the algorithm.

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

[4]We require that $C_U > 2C_b$ since otherwise buffering is impossible for some trees.

Input: Routing tree $T = (r, V, E)$ for net N with source r and sinks S, sink input capacitances c_s, load upper-bound C_U
Output: Optimum buffering of T with $c(D_b) \leq C_U$ for each $b \in \{r\} \cup B$

1. Find a critical vertex p by a post-order traversal of T
2. Find a heaviest child, u, of p
3. Insert a buffer b on the edge (u, p) such that
   $c_{(u,b)} = \min\{C_U - c(T_u), c_{(u,p)}\}$
4. Recursively find an optimum buffering B' of $T \setminus T_b$
5. Return $B = B' \cup \{b\}$

Fig. 1. Routed Net Buffering (RNB) algorithm.

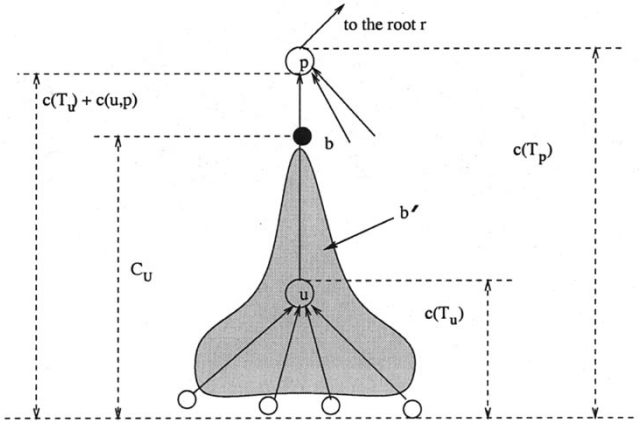

Fig. 2. Since $c(T_b) = C_U$, the tree T_b (shaded area) must contain a buffer b' in any optimum buffering B_{opt}. $(B_{\text{opt}} \setminus \{b'\}) \cup \{b\}$ is then an optimum buffering of T containing b.

Lemma 1: If p is a critical vertex of T and u is a child of p with $C_U - c(T_u) \leq c_{(u,p)}$, then there exists an optimum buffering of T containing a buffer b located on the edge (u, p) such that $c_{(u,b)} = C_U - c(T_u)$ (see Fig. 2).

Proof: Let the optimum buffering of T consist of the set of buffers B_{opt}. The subtree of T rooted at b must contain at least one buffer b' from B_{opt} since it has total capacitance equal to C_U. The lemma follows by observing that $(B_{\text{opt}} \setminus \{b'\}) \cup \{b\}$ is a feasible buffering of T. ■

Lemma 2: If p is a critical vertex of T and $c_{(u,p)} < C_U - c(T_u)$ for the heaviest child u of p, then there exists an optimum buffering of T that contains a buffer b placed immediately below p on the edge (u, p) (see Fig. 3).

Proof: Let the optimum buffering of T consist of the set of buffers B_{opt}. Since p is critical, T_p must contain at least one buffer b' of B_{opt}. We claim that $(B_{\text{opt}} \setminus \{b'\}) \cup \{b\}$ is an optimum buffering of T. The claim follows as in Lemma 1 if b' is located in T_b. Otherwise, the claim follows by observing that (i) by optimality, there is no buffer of B_{opt} on the path connecting b' to p in T, and (ii) $c(T_u) + c_{(u,p)} \leq c(D_{b'})$, since u is the heaviest child of p. ■

Notice that the capacitive load of each buffer inserted in Step 3 when $c_{(u,p)} \geq C_U - c(T_u)$ is exactly C_U, i.e., these buffers are "fully filled." Although this is not true for the buffers inserted when $c_{(u,p)} < C_U - c(T_u)$, it is easy to see that in this case inserted buffers have a capacitive load of at least C_U/k, where k is the degree of p. In particular, when the routing tree T is binary, we obtain Lemma 3.

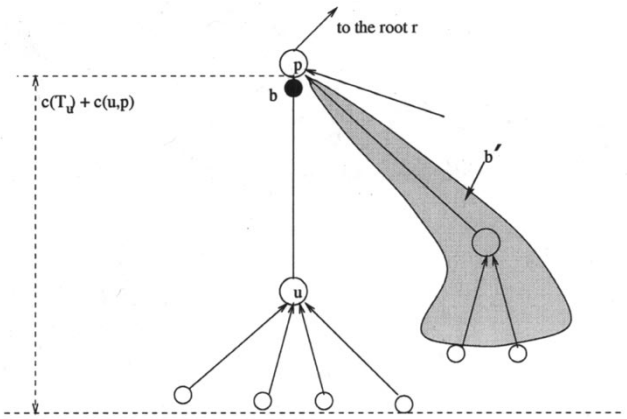

Fig. 3. When b' is located on a different branch (shaded area) than that of the heaviest child u, $c(T_u) + c_{(u,p)} \geq c(D_{b'})$. Hence, $(B_{\text{opt}} \setminus \{b'\}) \cup \{b\}$ is an optimum buffering of T containing b.

Lemma 3: If the input to the algorithm in Fig. 1 is a binary routing tree, then the load of each inserted buffer is at least $C_U/2$.

Lemma 3 will be used in proving the approximation guarantee for the algorithms in Section IV. It also gives a way to satisfy the simultaneous lower- and upper-bound constraints on buffer loads referred to in Footnote 1, since every routing tree can be converted to a binary tree by inserting zero-length edges.

B. Single Inverting Buffer Type

Optimal buffering with a single inverting buffer type is more complex than buffering with a noninverting buffer type. The greedy approach does not work in this case, and we must use dynamic programming. In bottom-up order, the algorithm (see Fig. 4) computes two solutions for each subtree of T, one for positive and one for negative topmost buffer input polarity. Then, after choosing the best output polarity for the source, it determines the position of the buffers by a top-down traversal. The running time of the algorithm is linear assuming that the degree of the routing tree T is bounded; in the rectilinear plane this assumption holds for all standard routing tree constructions, including the minimum spanning tree, the minimum-length Steiner tree, and approximations of the latter one.

For simplicity, we give the algorithm for binary trees, i.e., we assume that all vertices other than the source (which is the root of the tree) and the sinks (which are leaves) have outdegree 2. Without loss of generality, we assume that sink input capacitances are all equal to 0—nonzero sink capacitances can be compensated by increasing the length of the edges incident to the sinks. By scaling, we also assume that the unit wirelength capacitance, C_w, is equal to 1. The algorithm associates with each leaf v of the tree T two *labels*, $l^+(v)$ and $l^-(v)$, such that one of them belongs to $[0, C_U]$ and the other is 0. The labels $l^+(v)$ and $l^-(v)$ represent the penalty capacitance incurred in assuming that the sink has the opposite polarity. Initially, for each sink s,

$$l^+(s) = \begin{cases} 0, & \text{if } \sigma(s) = + \\ C_U, & \text{otherwise} \end{cases}$$

and $l^-(s) = C_U - l^+(s)$.

For each tree leaf v, define the *stem* of v to be the edge connecting v to its parent. Also, define a *fork* of T to be a set of four vertices (u, v, x_1, x_2), where x_1 and x_2 are two leaves, v is the common parent of x_1 and x_2, and u is the parent of v. The bottom-up phase of the algorithm consists of two main procedures: **Reduce_stem** and **Collapse_fork**. The procedure **Reduce_stem** simply reduces the length of the stem of a leaf v until it becomes strictly less than C_U. The procedure also counts the number of buffers inserted on the stem of v, referred to as $n^+(v)$ and $n^-(v)$, depending on the polarity of the topmost buffer.

The procedure **Collapse_fork** replaces a fork (u, v, x_1, x_2) with the single edge (u, v), computes the appropriate labels for v, and modifies the number of buffers inserted on the edges (v, x_1) and (v, x_2) as needed. The labels of v depend on the labels of x_1 and x_2 and the length of the edges (v, x_1) and (v, x_2). To guarantee optimality, **Collapse_fork** checks all possibilities of inserting buffers on the stems (v, x_1) and (v, x_2). Among the feasible bufferings of these two stems it chooses the one with the least buffers inserted, breaking ties according to the residual capacitance. Note that after the stems (v, x_1) and (v, x_2) have been reduced, the maximum number of buffers that may be inserted on each stem is at most 2. Thus, no more than 9 cases need to be checked in **Collapse_fork**, depending on whether 0, 1, or 2 buffers are inserted on each stem. In fact, since inserting 2 buffers in each of the two stems is always a dominated solution, we never need to check more than 8 cases.

Theorem 2: The algorithm in Fig. 4 finds an optimum buffering of the input tree T with the given inverting buffer type.

Input: Binary routing tree $T = (r, V, E)$ for net N with source r and sinks S, sink input capacitances c_s and polarities σ_s, upper-bound C_U
Output: Optimum buffering B of T consistent with sink polarities such that $c(D_b) \le C_U$ for every $b \in \{r\} \cup B$

1. $T' = T$
2. For each $s \in S$ do:
   If $\sigma_s = +$ then $l^+(s) = 0$, else $l^+(s) = C_U$
   $l^-(s) = C_U - l^+(s)$
   Reduce_stem(s)
3. While there is a fork (u, v, x_1, x_2) in T', **Collapse_fork**(u, v, x_1, x_2)
4. Insert buffers in T in top-down order:
   Let v be the single remaining leaf v in T', and $\mu \in \{+, -\}$ s.t. $l^\mu(v) = 0$
   Insert $n^\mu(v)$ buffers on the edge (r, v)
   For each fork (r, v, x_1, x_2), in reverse order of collapsing, do:
      Insert $n^\sigma(x_i)$ buffers on edges (v, x_i), $i = 1, 2$, where $\sigma = \mu$ if $n^\mu(v)$ is odd and $\sigma = -\mu$ if $n^\mu(v)$ is even
5. Return the set B of inserted buffers

Procedure **Reduce_stem**(v)
1. $n^+(v) = n^-(v) = 0$ // Initialize # of buffers on v's stem
2. While $l_{(u,v)} > C_U$ do:
   For each $\sigma \in \{+, -\}$, $n^\sigma(v) = n^\sigma(v) + 1$
   $l_{(u,v)} = l_{(u,v)} - (C_U - C_b)$
   Swap $l^-(v)$ with $l^+(v)$ // Switch topmost buffer polarity

Procedure **Collapse_fork**(u, v, x_1, x_2)

// Check all feasible bufferings of the stems (v, x_1) and (v, x_2)
1. For each $(i, j) \in \{0, 1, 2\} \times \{0, 1, 2\}$ and $\sigma \in \{+, -\}$ do:
   $l_{ij}^\sigma = \max\{0, l_{(v,x_1)} + l^\sigma(x_1) - i \cdot (C_U - C_b)\}$
   $+ \max\{0, l_{(v,x_2)} + l^\sigma(x_2) - j \cdot (C_U - C_b)\}$
   If $l_{ij}^\sigma \le C_U$ then $l_{ij}^\sigma = l_{ij}^\sigma + (i + j)C_U$
   Else $l_{ij}^\sigma = \infty$ // $i + j$ buffers are not sufficient

// Choose the topmost buffer positions
2. For each $\sigma \in \{+, -\}$ do:
   $l^\sigma(v) = \min\{l_{ij}^\sigma | i, j = 0, 1, 2\}$
   $(i^\sigma, j^\sigma) = \operatorname{argmin}\{l_{ij}^\sigma | i, j = 0, 1, 2\}$

// Find minimal label and normalize the opposite polarity label
3. $l^\mu(v) = \min\{l^+(v), l^-(v)\}$
   If $l^{-\mu}(v) > l^\mu(v) + C_U$, then $(i^{-\mu}, j^{-\mu}) = (i^\mu, j^\mu)$, $l^{-\mu}(v) = l^\mu(v) + C_U$

// Increment # of buffers for both stems and restore v's labels
4. For each $\sigma \in \{+, -\}$ do:
   $n^\sigma(x_1) = n^\sigma(x_1) + i^\sigma$, $n^\sigma(x_2) = n^\sigma(x_1) + j^\sigma$
   $l^\sigma(v) = l^\sigma(v) - (i^\sigma + j^\sigma)C_u$

// Reduce minimal label of v to 0, remove leaves x_1 and x_2, and reduce v's stem
5. $l_{(u,v)} = l_{(u,v)} + l^\mu(v)$, $l^{-\mu}(v) = l^{-\mu}(v) - l^\mu(v)$, $l^\mu(v) = 0$
6. $T' = T' \setminus \{x_1, x_2\}$
7. **Reduce_stem**(v)

Fig. 4. Routed Net Inverting Buffering (RNIB) algorithm.

IV. Approximating MBRP

The approximation factor of an algorithm A for a minimization problem P is the worst-case performance of A. Formally, the approximation factor of A is defined as $(A(I))/(OPT(I))$, where the supremum is taken over all instances I of the problem P, $A(I)$ is the output value of the algorithm A on input I, and $OPT(I)$ is the optimal value for the instance I. In this section we prove that, unless P = NP, no algorithm can guarantee a factor smaller than 2 for MBRP with single (inverting or noninverting) buffer type. On the positive side, for any $\varepsilon > 0$, we give a factor $2(1+\varepsilon)(1+(1/(C_U/(C_b-2))))$ approximation algorithm for MBRP with single noninverting buffer type and a factor $4(1+\varepsilon)(1+(1/(C_U/(C_b-2))))$ approximation algorithm for MBRP with single inverting buffer type.

A. Approximation Complexity of MBRP

Theorem 3: For any $\varepsilon > 0$, approximating MBRP within a factor of $2 - \varepsilon$ is NP-hard.

Proof: The proof is by reduction from the rectilinear Steiner minimum tree (RSMT) problem, which is NP-hard [11]. An RSMT instance consists of a set R of terminals and a number K, and the problem is to decide if terminals in R can be interconnected via a rectilinear Steiner tree of length K or less. Let r be an arbitrary terminal in R and let $S = R \setminus \{r\}$. Consider the MBRP instance in which all sinks have input capacitance 0, $C_b = 0$; $C_w = 1$, and $C_U = K$. Then, there exists a rectilinear Steiner tree of length at most K for the terminals in R if and only if the above MBRP instance has optimum cost equal to 1, and any $(2-\varepsilon)$-approximation algorithm for MBRP would find the optimum solution if this is the case. ■

Remark: Fig. 5 gives an example showing that MBRP is inherently more difficult than the RSMT problem since, in general, the Steiner points for MBRP do not belong to the Hanan grid, i.e., to the grid formed by the vertical and horizontal lines passing through terminals. In this example the input capacitance

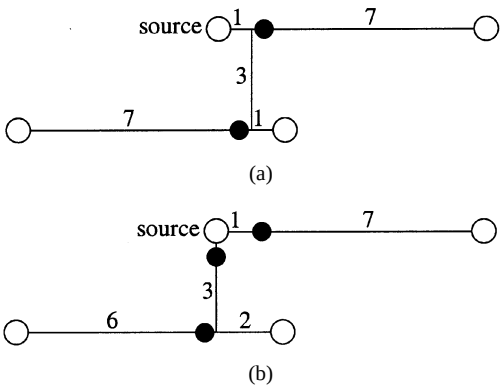

Fig. 5. (a) Four terminal net and optimum buffered routing using a non-Hanan grid edge. (b) Best buffered routing on the Hanan grid for the same net.

Input: Net N with source r and set of sinks S, sink input capacitances c_s, upper-bound C_U
Output: Buffered routing tree $T = (r, V, E, B)$ for N such that $c(D_b) \leq C_U$ for every $b \in \{r\} \cup B$

1. Find an α-approximate Steiner tree T for $\{r\} \cup S$
2. Transform T into a binary tree in which all sinks are leaves by duplicating internal nodes of degree > 3 and sinks of degree > 1 and adding zero-length edges between duplicated nodes
3. Add buffers to T using the RNB algorithm (see Fig. 1)

Fig. 6. Steiner Tree Buffering (STB) algorithm.

of each sink and of the buffers is 1, the unit wirelength capacitance C_w is 1, and the buffer load upper-bound C_U is eight. Any routing along the Hanan grid must use at least 3 buffers, while the optimum buffered routing, which uses a non-Hanan edge, has only two buffers.

B. Approximating MBRP With Single Noninverting Buffer Type

In this section we show that optimal buffering of an approximate rectilinear Steiner minimum tree over the terminals (see Fig. 6) comes within a constant factor of the MBRP optimum. Below, the output of a polynomial-time RSMT algorithm with approximation factor of α will be referred to as an *α-approximate Steiner tree*.

Theorem 4: The algorithm in Fig. 6 approximates the MBRP with single noninverting buffer type within a factor of $2\alpha(1 + (1/(C_U/(C_b - 2))))$ for every net with total sink capacitance of at least C_b.[5]

Proof: Let OPT be the number of stages in an optimum buffered routed net T_{opt}, and let CAP be the capacitance of T_{opt} before buffering, i.e.,

$$CAP = \sum_{s \in S} c_s + C_w \cdot \left(\sum_{e \in T_{\text{opt}}} l_e \right).$$

[5]In practice, the total sink capacitance is greater than C_b for almost all multipin nets. Also, the ratio C_U/C_b is typically much greater than 2 (recall that $C_U/C_b > 2$ to guarantee that every tree can be buffered). In our benchmarks C_U/C_b varies between 12 and 200, which corresponds to an approximation factor between 2.1α and 2.005α in Theorem 4.

Input: Net N with source r and set of sinks S, sink input capacitances c_s and polarities σ_s, upper-bound C_U
Output: Buffered routing tree $T = (r, V, E, B)$ for N consistent with sink polarities such that $c(D_b) \leq C_U$ for every $b \in \{r\} \cup B$

1. Find a buffered routing tree $T' = (r, V', E', B')$ using the STB algorithm
2. For each $b \in B' \cup \{r\}$, in the order given by a postorder traversal of T', do:
   Replace b with two inverters b^+ and b^- such that
   - the parent of b^- is b^+ and $l_{(b^-, b^+)} = 0$
   - the parent of b^+ is the parent p of b in T' and $l_{(b^+, p)} = l_{(b,p)}$
   For each $\sigma \in \{+, -\}$ add to T a Steiner tree rooted at b^σ and spanning all sinks with polarity σ in D_b
   $T' = T' \setminus D_b$
3. Return T

Fig. 7. Steiner Tree Inverting Buffering (STIB) algorithm.

Input: Net N with source r and set of sinks S, sink input capacitances c_s, upper-bound C_U
Output: Buffered routing tree $T = (r, V, E, B)$ for N such that $c(D_b) \leq C_U$ for every $b \in \{r\} \cup B$

1. $T = \emptyset$; $B = \emptyset$
2. T' = Steiner tree for $S \cup \{r\}$, rooted at r
3. While $c(T') > C_U$ do:
   Find the position of the first buffer b inserted by the RNB algorithm in T'
   If $c(T'_b) < C_U$ then
   // Fill b's capacitive load by joining a subtree to T'_b
   For each node i which is neither ancestor nor descendant of b, do:
   Compute T'_p by joining T'_i to T'_b, where p is either b or the point closest to $parent(b)$ on the shortest path between i and T'_b, whichever of the two is closer to $parent(b)$
   If $c(T'_p) < C_U$ then
   Set $b'(i)$ at distance $(C_U - c(T'_p))/C_w$ from p towards $parent(b)$
   Set $gain(i) = c(T'_i)/C_w - distance(b, b'(i))$
   End if
   End for
   Find i^* with maximum gain and join T'_{i^*} to T'_b
   Move buffer b to position $b'(i^*)$
   End if
   $B = B \cup \{b\}$, $T = T \cup T'_b$, $T' = T' \setminus T'_b$
   End while
4. Return $T \cup T'$, with buffer set B

Fig. 8. Cut&Connect algorithm.

In the optimum buffering of T_{opt}, each of the OPT stages has a capacitance of at most C_U. Since the total capacitance of the buffered tree T_{opt} is $CAP + (OPT - 1)C_b$, we get that $OPT \cdot C_U \geq CAP + (OPT - 1)C_b$, i.e.,

$$OPT \geq \frac{CAP - C_b}{C_U - C_b}. \tag{3}$$

Let CAP' be the capacitance before buffering of the α-approximate Steiner tree constructed by the algorithm in Fig. 6. Then $CAP' - s \leq \alpha(CAP - s)$, where $s = \sum_{s \in S} c_s$ is the total input capacitance of the sinks. Since we assume that $s \geq C_b$, this gives $CAP' \leq \alpha CAP - (\alpha - 1)s \leq \alpha(CAP - C_b) + C_b$, i.e.,

$$CAP' - C_b \leq \alpha(CAP - C_b). \tag{4}$$

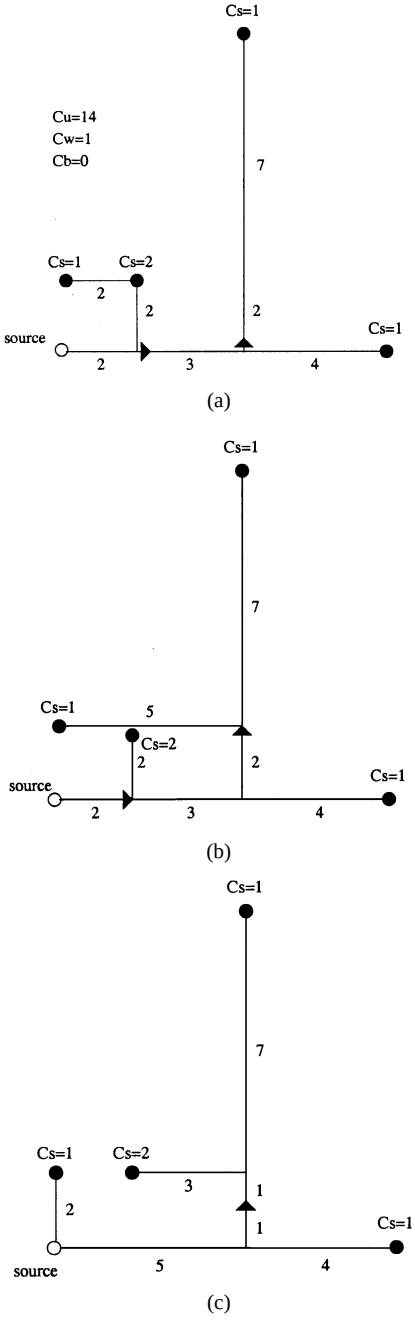

Fig. 9. Example execution of the Cut&Connect heuristic, where $C_U = 14$, $C_b = 0$, and $C_w = 1$. (a) The original routing tree after applying RNB. (b) The modified routing tree after cutting the leftmost terminal and "filling" the first buffer. (c) The optimal buffered routed tree with a single buffer.

Let A be the number of stages in the buffering produced by the algorithm. Since T is a binary tree, by Lemma 3 every buffer inserted by the algorithm in Fig. 1 has a minimum load of $C_U/2$. Furthermore, the total capacitance of the source stage and of the stage driven by the last inserted buffer is greater than C_U (otherwise the source can drive alone both stages). Thus, $CAP' + (A-1)C_b \geq A \cdot (C_U/2)$, i.e.,

$$A \leq \frac{CAP' - C_b}{\frac{C_U}{2} - C_b} = 2\frac{CAP' - C_b}{C_U - 2 \cdot C_b}. \tag{5}$$

Input: Net N with source r and set of sinks S, sink input capacitances c_s, upper-bound C_U
Output: Buffered routing tree $T = (r, V, E, B)$ for N such that $c(D_b) \leq C_U$ for every $b \in \{r\} \cup B$

1. $T = \emptyset$; $B = \emptyset$
2. T' = Steiner tree for $S \cup \{r\}$, rooted at r
3. While $c(T') > C_U$ do:
   // Find a critical node with maximum subtree capacitance
   Find $v \in T'$ with maximum $c(T'_v)$ s.t. $c(T'_v) < C_U$ and $c(T'_{parent(v)}) > C_U$
   // Fill the load of the subtree by connecting neighboring sinks
   $subtree\_load = c(T'_v)$; $S' = T'_v \cap S$; $T = T \cup T'_v$
   q = sink in $S \setminus S'$ closest to S'; p = sink of S' closest to q
   While $subtree\_load + C_w l_{(p,q)} + c_q < C_U$ do:
      $subtree\_load = subtree\_load + C_w l_{(p,q)} + c_q$
      $S' = S' \cup \{q\}$; $T = T + (p,q)$
      q = sink in $S \setminus S'$ closest to S'; p = sink of S' closest to q
   End while
   Place buffer b at distance $(C_U - subtree\_load)/C_w$ from p, towards q
   $B = B \cup \{b\}$; $S = (S \setminus S') \cup \{b\}$
   T' = Steiner tree for $S \cup \{r\}$, rooted at r
   End while
4. Return $T \cup T'$, with buffer set B

Fig. 10. Clustering algorithm.

Finally, inequalities (3)–(5) give

$$\frac{A}{\text{opt}} \leq 2\frac{CAP' - C_b}{CAP - C_b} \cdot \frac{C_U - C_b}{C_U - 2 \cdot C_b} \leq 2\alpha \cdot \left(1 + \frac{1}{\frac{C_U}{C_b - 2}}\right).$$

■

Since the rectilinear Steiner tree for a given set of terminals can be approximated in polynomial time to within any desired accuracy using Arora's PTAS [5], Theorem 4 gives the following.

Corollary 1: For any $\varepsilon > 0$, the MBRP with single noninverting buffer type can be approximated within a factor of $2(1+\varepsilon)(1+(1/(C_U/(C_b-2))))$ in time $O(|S|(\log|S|)^{O(1/\varepsilon)} + |B|)$.

C. Approximating MBRP With Single Inverting Buffer Type

A naive solution to handling sink polarities is to make the polarity of all sinks the same by inserting one inverter for each sink of the minority polarity, and then use noninverting buffers to route the signal from the source. In the worst case this solution may require as many as $|S|/2$ inverters, plus the noninverter buffers needed to drive a Steiner tree spanning all terminals. A better solution is to construct two separate Steiner trees, one for the positive sinks and one for the negative sinks, buffer them optimally with noninverting buffers using the RNB algorithm, and then insert a single inverter at the top of one of them.

If an inverting buffer occupies less than half the area of a noninverting buffer with the same driving strength, an even better solution is provided by algorithm in Fig. 7. In this algorithm, we construct a routing tree for all sinks, buffer it with noninverting buffers, and then make it consistent with sink polarities by replacing each noninverting buffer by two inverters.

Input: Net N with source r and set of sinks S, sink input capacitances c_s and polarities σ_s, upper-bound C_U
Output: Buffered routing tree $T = (r, V, E, B)$ for N consistent with sink polarities such that $c(D_b) \leq C_U$ for every $b \in \{r\} \cup B$

1. Find a buffered routing tree $T' = (r, V', E', B')$ using the STB algorithm
2. For each $b \in B' \cup \{r\}$, in the order given by a postorder traversal of T', do:
   If b drives sinks or non-swappable inverters with both polarities then
      For every swappable "$-$" inverter q^- driven by b, reconnect q^- as a child of its sibling q^+
      Replace b by two siblings, which are swappable inverters b^+ and b^- with polarity "$+$", resp. "$-$"
      Delete b's stage D'_b from T', then add to T a Steiner tree rooted at b^+ having as leaves all "$-$" sinks and non-swappable inverters in D'_b and a Steiner tree rooted at b^- having as leaves all "$+$" sinks/inverters in D'_b
   Else, if b drives no sink or non-swappable inverter with "$-$" polarity, then
      For every swappable "$-$" inverter q^- driven by b^-, reconnect q^- as a child of its sibling q^+
      Replace b by a non-swappable inverter b^- with "$-$" polarity, delete b's stage from T' and add it to T
   Else // b drives no sink or non-swappable inverter with "$+$" polarity
      For every swappable "$+$" inverter q^+ driven by b^+, reconnect q^+ as a child of its sibling q^-
      Replace b by a non-swappable inverter b^+ with "$+$" polarity, delete b's stage from T' and add it to T
3. Return T

Fig. 11. Steiner Tree Inverting Buffering with Swapping (STIB-S) algorithm.

Input: Net N with source r and set of sinks S, sink input capacitances c_s and polarities σ_s, upper-bound C_U
Output: Buffered routing tree $T = (r, V, E, B)$ for N consistent with sink polarities such that $c(D_b) \leq C_U$ for every $b \in \{r\} \cup B$

1. Find an α-approximate Steiner tree T for $\{r\} \cup S$
2. Transform T into a binary tree in which all sinks are leaves by duplicating internal nodes of degree > 3 and sinks of degree > 1 and adding zero-length edges between duplicated nodes
3. For each node v of T, in postorder, do:
   Repeat forever
      If $S^+(v) > C_U - C_b$ and $S^-(v) > C_U - C_b$ then insert an inverter with appropriate polarity in the highest position on the branch with maximum capacitance among $Br^+_{u_1}$, $Br^+_{u_2}$, $Br^-_{u_1}$, $Br^-_{u_2}$, where u_1 and u_2 are v's children
      If $S^+(v) > C_U$ then insert inverter with "$-$" polarity in the highest feasible position on the maximum capacitance branch among $Br^+_{u_1}$, $Br^+_{u_2}$
      If $S^-(v) > C_U$ then insert inverter with "$+$" polarity in the highest feasible position on the maximum capacitance branch among $Br^-_{u_1}$, $Br^-_{u_2}$
      Else exit repeat loop
5. Return T

Fig. 12. Steiner Tree Inverting Buffering with Load Filling (STIB-LF) algorithm.

Theorem 5: The algorithm in Fig. 7 approximates the MBRP with single inverting buffer type within a factor of at most $4\alpha(1 + (1/(C_U/(C_b - 2))))$.

Proof: First we show that T is a feasible solution. Indeed, by construction, each inserted inverter drives sinks or inverters of the same polarity. Also, the load of each inverter inserted in T is at most C_U, since this load is never larger than the load of the corresponding stage D_b of T'.[6]

The key observation is that the optimum number of inverting buffers, OPT, is no less than the optimum number of noninverting buffers OPT'. Let A' and A be the number of buffers inserted by the algorithms STB and STIB, respectively. Then, by Theorem 4, $A \leq 2 \cdot A' \leq 4\alpha(1 + (1/(C_U/(C_b - 2))))OPT' \leq 4\alpha(1 + (1/(C_U/(C_b - 2))))OPT$. ■

Using Arora's PTAS [5], Theorem 5 gives the following.

Corollary 2: For any $\varepsilon > 0$, the MBRP with single inverting buffer type can be approximated within a factor of $4(1+\varepsilon)(1+(1/(C_U/(C_b - 2))))$ in time $O(|S|(\log|S|)^{O(1/\varepsilon)} + |B|)$.

By Theorem 3, no approximation algorithm with a factor better than 2 exists for MBRP with single inverting buffer type. Closing the gap between Corollary 2 and this hardness result is an interesting open problem.

[6]For simplicity we assume that the buffer input capacitance C_b is less than any sink capacitance. The algorithm in Fig. 7 can be modified such that this assumption is not necessary.

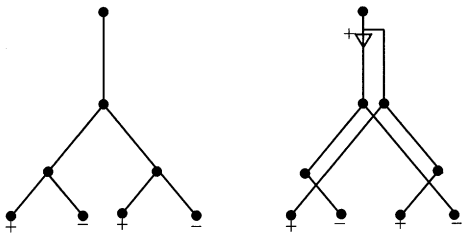

Fig. 13. Inverter insertion with the algorithm in Fig. 12. (a) Tree before inverter insertion. (b) Tree after inserting an inverter driving the "—" sinks.

V. MBRP Heuristics With Improved Practical Performance

A. Noninverting Buffer Type

Theorems 3 and 4 imply that the STB algorithm is essentially the best possible from the point of view of worst case approximation guarantee. In this section we describe two MBRP heuristics which, by changing the topology of the Steiner tree, improve upon the STB algorithm on practical instances.

The first heuristic, called Cut&Connect (see Fig. 8), modifies the Steiner tree constructed by STB in a bottom-up fashion, starting from the sinks and working toward the root. When

TABLE I
NUMBER OF BUFFERS, WIRELENGTH (mm), AND RUNTIME (CPU s) FOR THE RNB, CUT & CONNECT, AND CLUSTERING HEURISTICS FOR NONINVERTING BUFFER INSERTION

Benchmark		MST+RNB			MST+Cut&Conn.			MST+Cluster			Lower
#term.	C_U	#b	WL	time	#b	WL	time	#b	WL	time	bound
330	500	17	25	0.81	16	26	0.83	16	25	0.94	15
	1000	8	25	0.81	8	26	0.82	7	26	0.83	7
	2000	4	25	0.81	4	28	0.83	3	26	0.81	2
	4000	2	25	0.81	1	26	0.82	1	25	0.78	0
	8000	0	25	0.81	0	25	0.82	0	25	0.78	0
830	500	34	68	0.97	34	69	0.97	33	69	2.05	32
	1000	17	68	0.97	16	69	0.96	16	70	1.32	15
	2000	8	68	0.97	8	71	0.96	8	72	1.06	6
	4000	3	68	0.97	3	68	0.96	3	68	0.95	2
	8000	1	68	0.97	1	68	0.96	1	68	0.88	0
1900	500	56	45	1.02	54	50	1.33	51	46	3.27	49
	1000	28	45	1.02	26	53	1.43	24	46	1.93	23
	2000	13	45	1.01	12	51	1.33	12	47	1.36	11
	4000	6	45	1.02	6	47	1.18	5	46	1.06	5
	8000	2	45	1.01	2	45	1.05	2	45	0.94	1
2400	500	74	58	1.07	70	65	1.61	64	59	4.93	62
	1000	33	58	1.06	32	62	1.43	31	59	2.65	29
	2000	17	58	1.06	17	64	1.87	15	59	1.73	14
	4000	8	58	1.07	8	60	1.45	7	58	1.29	7
	8000	4	58	1.07	3	60	1.35	3	59	1.07	2
2600	500	147	89	1.15	144	98	1.73	134	92	10.39	128
	1000	70	89	1.14	67	99	1.99	63	91	5.40	61
	2000	33	89	1.14	32	99	1.72	31	91	3.02	30
	4000	17	89	1.14	16	96	1.92	15	90	1.96	14
	8000	8	89	1.15	8	92	1.61	7	91	1.39	6
12000	500	244	266	2.63	236	285	9.27	222	272	106.83	184
	1000	116	266	2.63	113	283	11.54	106	272	46.90	88
	2000	56	266	2.63	55	279	12.42	52	271	21.25	42
	4000	28	266	2.64	28	290	13.32	25	269	10.59	20
	8000	13	266	2.63	13	274	8.10	12	268	5.82	9
22000	500	1418	1396	4.39	1395	1551	21.62	1305	1476	1172.75	1197
	1000	674	1396	4.39	656	1524	30.36	613	1444	540.28	575
	2000	330	1396	4.39	319	1506	49.58	298	1437	257.99	282
	4000	164	1396	4.39	159	1471	95.40	146	1426	121.24	139
	8000	80	1396	4.39	78	1448	106.98	72	1420	60.33	68
34000	500	806	990	6.59	778	1068	39.13	729	1016	890.01	591
	1000	388	990	6.58	374	1071	58.55	350	1011	424.81	283
	2000	191	990	6.58	153	1058	89.04	171	1009	208.79	138
	4000	95	990	6.57	92	1065	147.62	84	1002	103.59	68
	8000	45	990	6.57	44	1036	113.80	42	1000	49.25	33

finding a buffer b whose load is smaller than C_U, the heuristic tries to fill b's load up to C_U by cutting a subtree from some other part of the tree and reconnecting it to the closest point in T_b. If the resulted modified stage of the buffer b has the capacitive load still no more than C_U, then such reconnect is accepted, otherwise the reconnect is reversed. If needed, the position of b may be adjusted to ensure that its load remains at most C_U (see Fig. 9(a) and (b)).

Similar to Cut&Connect, the Clustering heuristic (see Fig. 10) repeatedly chops off buffer stages from a Steiner tree over terminals. The Clustering heuristic is essentially a greedy algorithm which tries the reconnected vertex with the largest gain. There are two main differences between Clustering and Cut&Connect. The first difference is in the way buffer loads are filled: Clustering always adds one sink at a time, while Cut&Connect adds whole subtrees. For example, Clustering constructs the tree in Fig. 9(c), while Cut&Connect cannot. The second difference is in the fact that Clustering recomputes the Steiner tree after chopping off each buffer stage. Tree recomputation improves solution quality, but also leads to a much higher time complexity, of $O(|B|T_{rsmt})$, where T_{rsmt} is the time needed to compute a Steiner tree. To achieve a competitive running time, our implementation of Clustering uses minimum spanning trees as approximate Steiner trees.

B. Inverting Buffer Type

Both algorithms in this section are improved versions of the STIB algorithm in Section IV-C. The improvement in the first algorithm (see Fig. 11) is based on the following observations: The STIB algorithm replaces each buffer inserted by the STB algorithm by a pair of inverters, but if all sinks driven by a buffer have the same polarity then a single inverter replacement is sufficient. Furthermore, new saving opportunities can be created for higher levels by *swapping* the two inverters in an inserted pair such that the most appropriate polarity comes on top.

A significant limitation of the STIB-S algorithm is that it inserts inverters only at locations of buffers inserted by the STB algorithm. In order to avoid leaving too much unused driving capacity, the STIB-LF algorithm in Fig. 12 computes the placement of inverters in bottom-up order as the highest position which can still drive all positive (respectively negative) sinks below, thus in effect "filling" the load of each inverter as close as possible to the its full capacity. Similar to the STIB and STIB-S algorithms, whenever an inverter is inserted by the algorithm in Fig. 12 the driven sinks/buffers are connected to the inverter by duplicating paths of the routing tree (see Fig. 13).

In the algorithm in Fig. 12, we use some additional notation. For every node v of a tree T, let $D_v^+(D_v^-)$ be the tree

TABLE II
NUMBER OF BUFFERS, WIRELENGTH (mm), AND RUNTIME (CPU s) FOR RNIB, STIB-S, AND STIB-LF HEURISTICS ON TESTCASES WITH ALL SINKS OF THE SAME POLARITY

Benchmark		MST+RNIB			MST+STIB-S			MST+STIB-LF		
#term.	C_U	#b	WL	time	#b	WL	time	#b	WL	time
330	500	34	25	0.99	26	31	0.96	21	36	0.97
	1000	16	25	0.99	11	30	0.96	9	32	0.97
	2000	8	25	0.98	6	27	0.96	5	30	0.97
	4000	4	25	0.99	3	27	0.96	3	28	0.97
	8000	0	25	0.99	0	25	0.96	0	25	0.97
830	500	64	68	1.09	50	92	1.04	50	106	1.06
	1000	34	68	1.08	26	93	1.04	22	100	1.05
	2000	16	68	1.08	12	89	1.04	11	103	1.05
	4000	6	68	1.08	5	90	1.04	4	90	1.05
	8000	2	68	1.08	2	76	1.05	2	76	1.05
1900	500	112	45	1.10	80	51	1.07	61	55	1.06
	1000	56	45	1.09	40	50	1.07	30	53	1.05
	2000	26	45	1.09	19	49	1.07	14	51	1.05
	4000	12	45	1.10	9	47	1.06	7	50	1.05
	8000	4	45	1.09	3	48	1.06	3	48	1.06
2400	500	148	58	1.14	109	66	1.09	79	71	1.12
	1000	66	58	1.15	47	65	1.09	35	68	1.11
	2000	34	58	1.15	25	62	1.09	18	65	1.12
	4000	16	58	1.14	11	61	1.09	9	63	1.11
	8000	8	58	1.15	6	61	1.09	5	62	1.11
2600	500	292	89	1.17	215	107	1.12	165	118	1.11
	1000	140	89	1.17	102	102	1.12	76	111	1.12
	2000	66	89	1.16	49	101	1.12	36	108	1.11
	4000	34	89	1.17	25	98	1.12	18	104	1.11
	8000	16	89	1.17	12	97	1.13	9	99	1.12
12000	500	484	266	2.02	354	305	1.82	276	328	1.83
	1000	232	266	2.01	168	298	1.82	128	317	1.82
	2000	112	266	2.02	82	288	1.81	62	303	1.83
	4000	56	266	2.02	41	286	1.81	29	299	1.83
	8000	26	266	2.02	19	282	1.81	14	292	1.82
22000	500	2832	1396	3.23	2069	1706	2.69	1664	1912	2.72
	1000	1344	1396	3.14	974	1646	2.67	756	1807	2.72
	2000	660	1396	3.09	476	1583	2.68	360	1709	2.71
	4000	328	1396	3.07	236	1550	2.68	175	1646	2.71
	8000	160	1396	3.07	114	1509	2.68	86	1585	2.71
34000	500	1606	990	4.31	1173	1133	3.68	899	1233	3.72
	1000	774	990	4.26	563	1109	3.67	427	1190	3.72
	2000	382	990	4.22	274	1087	3.67	206	1157	3.71
	4000	190	990	4.23	138	1067	3.66	103	1130	3.71
	8000	90	990	4.21	67	1068	3.67	48	1104	3.71

rooted at v which is the union of all paths from v to the positive (respectively negative) driven sinks/buffers in D_v, and denote by $S^+(v)(S^-(v))$ the total capacitance of D_v^+ (respectively D_v^-), e.g., $S^+(v) = c_v$ if v is positive sink and $S^+(v) = 0$ if v is a negative sink. Also, let $Br_v^+ = D_v^+ + (v, parent(v))$ if $S^+(v) > 0$ and $Br^+(v) = \emptyset$ otherwise, and, similarly, $Br_v^- = D_v^- + (v, parent(v))$ if $S^-(v) > 0$ and $Br^-(v) = \emptyset$ otherwise.

VI. EXPERIMENTAL RESULTS

We have implemented the RNB and RNIB algorithms for optimally buffering a given tree with a single noninverting, respectively inverting, buffer type, the Cut&Connect and Clustering heuristics for MBRP with single noninverting buffer type, as well as the STIB-S and the STIB-LF heuristics for MBRP with single inverting buffer type. Tables I–III give the results obtained by these heuristics on eight large nets extracted from recent industrial designs. For all heuristics, the initial tree is a minimum spanning tree over the terminals. The runtime is in CPU seconds on a SUN Ultra 60 and includes the time for computing the initial minimum spanning tree. For all datasets, $C_w = 0.177\ fF/\mu$m, $C_b = 37.5\ fF$, while sink input capacitances are varying between $2.04\ fF$ and $200\ fF$.

Table I gives the results obtained by the three heuristics for noninverting buffering. For comparison, Table I includes a lower bound on the optimum number of buffers, calculated according to (3) with RSMT length estimated using the edge-based heuristic of [7]. The lower-bound estimates the number of buffers by assuming that (a) the tree is shortest possible, and (b) each buffer is fully loaded. Since the optimum solution is unlikely to meet these two conditions simultaneously, the lower-bound may significantly under-estimate the optimum number of buffers.

Results in Table I show that, on the average, the Cut&Connect heuristic inserts 5.81% fewer buffers than the RNB algorithm, while increasing the wirelength by 6.52%. The Clustering heuristic inserts 10.43% fewer buffers than RNB on the average, with an average wirelength increase of only 2.02%. In fact, Clustering solutions are almost always better than Cut&Connect results *both* in number of inserted buffers and total wirelength. As

TABLE III
NUMBER OF BUFFERS, WIRELENGTH (mm), AND RUNTIME (CPU s) FOR RNIB, STIB-S, AND STIB-LF HEURISTICS ON TESTCASES WITH RANDOM SINK POLARITIES. SPLITMST VARIANTS CORRESPOND TO INDEPENDENTLY BUFFERING MINIMUM SPANNING TREES FOR THE POSITIVE AND NEGATIVE SINKS

Benchmark		MST+RNIB			SplitMST+RNIB			MST+STIB-S			SplitMST+STIB-S			MST+STIB-LF			SplitMST+STIB-LF		
#term.	C_U	#b	WL	time	#b	WL	time	#b	WL	time	#b	WL	time	#b	WL	time	#b	WL	time
330	500	177	25	0.99	45	42	1.94	34	43	0.98	34	55	1.93	25	46	0.97	31	60	1.91
	1000	177	25	0.99	21	42	1.94	17	45	0.98	17	50	1.93	12	45	0.97	15	53	1.90
	2000	177	25	0.99	9	42	1.95	9	46	0.98	7	46	1.93	5	46	0.97	7	49	1.90
	4000	177	25	0.99	5	42	1.94	5	46	0.98	5	45	1.93	3	46	0.97	5	45	1.90
	8000	177	25	0.99	1	42	1.94	1	46	0.98	1	42	1.94	1	46	0.97	1	42	1.90
830	500	420	68	1.07	99	117	2.00	67	115	1.05	79	161	1.96	56	120	1.06	81	184	1.99
	1000	420	68	1.08	49	117	2.00	35	121	1.05	39	161	1.96	27	123	1.05	38	177	1.99
	2000	420	68	1.07	25	117	1.99	17	122	1.05	19	155	1.96	14	123	1.05	19	165	1.99
	4000	420	68	1.08	13	117	2.00	7	122	1.05	10	146	1.96	7	123	1.05	10	156	1.99
	8000	420	68	1.07	5	117	1.99	3	123	1.05	5	133	1.97	3	123	1.06	5	133	2.01
1900	500	984	45	1.10	131	67	2.06	113	76	1.06	95	75	2.00	70	76	1.08	75	82	2.00
	1000	984	45	1.10	63	67	2.06	57	76	1.06	46	75	2.00	34	76	1.07	35	78	2.00
	2000	984	45	1.10	31	67	2.06	27	76	1.05	23	72	1.99	17	76	1.07	18	75	2.00
	4000	984	45	1.10	13	67	2.06	13	76	1.05	11	71	1.99	9	76	1.08	9	71	2.00
	8000	984	45	1.10	5	67	2.06	5	76	1.05	5	67	1.99	3	76	1.08	5	67	2.00
2400	500	1245	58	1.15	163	83	2.09	149	96	1.09	120	96	2.06	93	97	1.09	91	105	2.04
	1000	1245	58	1.15	79	83	2.09	67	97	1.09	59	92	2.06	41	97	1.08	44	99	2.03
	2000	1245	58	1.15	37	83	2.09	35	97	1.08	28	91	2.06	20	97	1.09	21	95	2.03
	4000	1245	58	1.15	17	83	2.09	17	97	1.09	13	89	2.07	11	97	1.08	11	92	2.03
	8000	1245	58	1.15	9	83	2.09	9	97	1.09	7	85	2.06	5	97	1.09	7	85	2.03
2600	500	1359	89	1.19	323	130	2.12	295	146	1.12	237	154	2.07	179	150	1.12	183	173	2.07
	1000	1355	89	1.19	153	130	2.11	141	149	1.11	111	149	2.07	83	150	1.11	85	163	2.06
	2000	1355	89	1.19	75	130	2.11	67	150	1.11	56	147	2.07	40	150	1.12	41	159	2.06
	4000	1355	89	1.19	37	130	2.10	35	150	1.12	29	144	2.06	21	150	1.11	20	150	2.06
	8000	1355	89	1.19	17	130	2.12	17	150	1.11	13	135	2.07	11	150	1.12	11	141	2.06
12000	500	6000	266	2.24	583	381	2.95	488	445	1.84	425	437	2.74	335	449	1.84	336	476	2.76
	1000	6000	266	2.24	285	381	2.93	233	448	1.83	209	432	2.73	156	449	1.84	158	459	2.75
	2000	6000	266	2.23	139	381	2.93	113	448	1.84	100	416	2.73	76	449	1.84	76	440	2.74
	4000	6000	266	2.25	65	381	2.93	57	449	1.84	49	409	2.74	36	449	1.84	38	431	2.75
	8000	6000	266	2.24	31	381	2.94	27	449	1.84	22	406	2.74	19	449	1.84	18	420	2.75
22000	500	11366	1396	3.96	3350	2008	4.11	2815	2289	2.71	2456	2461	3.61	1849	2350	2.73	2007	2764	3.63
	1000	11284	1396	3.94	1596	2008	4.03	1349	2337	2.71	1160	2369	3.61	875	2358	2.72	928	2605	3.63
	2000	11284	1396	3.95	781	2008	3.98	661	2353	2.71	565	2297	3.60	429	2360	2.73	442	2482	3.61
	4000	11284	1396	3.96	383	2008	3.98	329	2358	2.71	280	2235	3.59	211	2361	2.73	211	2360	3.62
	8000	11284	1396	3.96	189	2008	3.99	161	2360	2.71	138	2160	3.59	106	2361	2.73	104	2281	3.61
34000	500	17252	990	5.91	1983	1434	5.09	1613	1657	3.68	1453	1659	4.47	1110	1670	3.72	1163	1831	4.52
	1000	17252	990	5.91	939	1434	5.03	777	1665	3.68	687	1635	4.47	533	1671	3.72	541	1776	4.52
	2000	17252	990	5.92	471	1434	5.06	383	1670	3.67	339	1594	4.47	256	1672	3.72	261	1718	4.51
	4000	17252	990	5.96	227	1434	5.04	191	1672	3.67	163	1577	4.46	125	1672	3.72	127	1684	4.50
	8000	17252	990	5.94	111	1434	5.02	91	1672	3.68	81	1548	4.46	63	1672	3.72	61	1619	4.51

expected, the Clustering heuristic—which recomputes a minimum spanning tree after each buffer insertion—has the slowest runtime, being as much as 267 times slower than RNB and 24 times slower than Cut&Connect. However, Clustering runtime remains practical: even for the nets with tens of thousands of sinks Clustering takes just a little over one second of CPU time per inserted buffer.

We have compared the inverting buffering heuristics on two sets of datasets. In one set (Table II) all sinks are assigned the same polarity, while in the second (Table III) sink polarities are assigned at random. The results indicate that optimal inverting buffering of a minimum length spanning or Steiner tree can be very far from optimal, and that heuristics for simultaneous tree construction and buffering are particularly important in this case.

The results for uniform sink polarities given in Table II show that the STIB-S heuristic inserts on the average 25.74% fewer buffers compared to the MST buffered optimally using RNIB; the STIB-S wirelength is larger than the MST wirelength by an average of 13.38%. With the same or even smaller runtime, the STIB-LF heuristic reduces the number of buffers by an average of 57.23% compared to RNIB, with an average wirelength increase of 20.84%.

Table III gives the results obtained by the inverting buffering heuristics on testcases with random sink polarities. We have included in comparison two variants of each heuristic: the first variant buffers (or starts with) an MST spanning *all* sinks, while the second variant computes separate MSTs for the sinks of each polarity and buffers each tree independently. Such a "split" construction proves to be particularly important for RNIB buffering, since on the average half of the sinks require an inverter when RNIB is run on the MST over all sinks.[7] The split MST construction also helps the STIB-S heuristic in most cases, reducing the number of buffers by an average of 8.21% compared to the running STIB-S on the MST over all sinks. Interestingly, however, the split MST construction *hurts* the STIB-LF heuristic in most cases, increasing the number of buffers by an average of 13.64% and the wirelength by 6.44%. The STIB-LF heuristic on the MST for all sinks gives the best results on the average, with 42.31% fewer buffers and 13.90% wirelength increase compared to RNIB over the split MST, respectively 25.30% fewer buffers and 1.05% wirelength increase compared to STIB-S over the split MST.

VII. CONCLUSION AND FUTURE RESEARCH

In this paper we have addressed a minimum-buffered routing problem which asks for bounded input rise/fall time for all

[7]The number of inverters inserted by RNIB is almost the same for the whole range of driving strengths since most inverters are inserted to meet polarity, not load cap, constraints.

buffers and sinks. We have analyzed the approximation complexity of this problem and given provably-good algorithms for buffering with a single inverting or noninverting buffer type. We have also proposed local-improvement and clustering heuristics with improved practical performance; experiments conducted on industrial datasets show that our heuristics are efficient and insert a near-optimum number of buffers.

A natural research direction is to extend the results in this paper to MBRP with multiple buffer/invertor types. If the buffer library can be arbitrary the problem becomes considerably harder than the single buffer type case considered in this paper. For example, a direct reduction from the subset sum problem shows that even finding the optimum buffering of a routed two-pin net is NP-hard. Our ongoing research addresses the case of libraries with small number of buffer types. We also investigate multisource formulations, in which the buffer solution should be legal for multiple rooted orientations of the tree, and multi-constraint formulations, in which, e.g., input capacitance and fanout must be upper-bounded simultaneously.

REFERENCES

[1] C. Alpert and A. Devgan, "Wire segmenting for improved buffer insertion," in *ACM/IEEE Design Automation Conf.*, 1997, pp. 588–593.

[2] C. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion for noise and delay optimization," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 1633–1645, Nov. 1999.

[3] C. Alpert, A. B. Kahng, B. Liu, I. Măndoiu, and A. Zelikovsky, "Minimum-Buffered Routing of Non-Critical Nets for Slew Rate and Reliability Control," Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA, Tech. Rep. CS2001-0681, 2001.

[4] C. J. Alpert, R. G. Gandham, J. L. Neves, and S. T. Quay, "Buffer library selection," in *IEEE Int. Conf. Computer Design*, 2000, pp. 221–226.

[5] S. Arora, "Polynomial time approximation scheme for Euclidean TSP and other geometric problems," *J. ACM*, vol. 45, pp. 753–782, 1998.

[6] C. L. Berman, J. L. Carter, and K. F. Day, "The fanout problem: from theory to practice," in *Advanced Research in VLSI: Proc. 1989 Decennial Caltech Conf.*, 1989, pp. 69–99.

[7] M. Borah, R. M. Owens, and M. J. Irwin, "A fast and simple Steiner routing heuristic," *Discrete Appl. Math.*, vol. 90, pp. 51–67, 1999.

[8] W. Chen, C.-T. Hsieh, and M. Pedram, "Simultaneous gate sizing and fanout optimization," in *Proc. IEEE-ACM Int. Conf. Computer-Aided Design*, 2000, pp. 374–378.

[9] M. Edahiro and R. J. Lipton, "Clock buffer placement algorithm for wire-delay-dominated timing model," in *Proc. Great Lakes Symp. VLSI*, 1996, pp. 143–147.

[10] 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 Conf.*, 1998, pp. 525–532.

[11] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM J. Appl. Math.*, vol. 32, pp. 826–834, 1977.

[12] C. Hu, "Hot carrier effects," in *Advanced MOS Device Physics*, N. G. Einspruch, Ed. New York: Academic, 1989, pp. 119–160.

[13] A. B. Kahng, S. Muddu, E. Sarto, and R. Sharma, "Interconnect tuning strategies for high-performance ICs," in *Proc. Conf. Design Automation and Test in Europe*, Feb. 1998.

[14] S. Kundu and J. Misra, "A linear tree partitioning algorithm," *SIAM J. Comput.*, vol. 6, pp. 151–154, 1977.

[15] 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*, vol. 31, pp. 437–447, Mar. 1996.

[16] ——, "Similtaneous routing and buffer insertion for high performance interconnect," in *Proc. Great Lakes Symp. VLSI*, 1996, pp. 148–153.

[17] S. Lin and M. Marek-Sadowska, "A fast and efficient algorithm for determining fanout trees in large networks," in *Proc. Eur. Design Automation Conf.*, 1991, pp. 539–544.

[18] T. Okamoto and J. Cong, "Buffered steiner tree construction with wire sizing for interconnect layout optimization," in *Proc. IEEE-ACM Int. Conf. Computer-Aided Design*, 1996, pp. 44–49.

[19] S. Pullela, N. Menezes, J. Omar, and L. T. Pillage, "Skew and delay optimization for reliable buffered clock trees," in *Proc. IEEE-ACM Int. Conf. Computer-Aided Design*, 1993, pp. 556–559.

[20] S. Rzepka, K. Banerjee, E. Meusel, and C.Chenming Hu, "Characterization of self-heating in advanced vlsi interconnect lines based on thermal finite element simulation," .

[21] L. Scheffer, Personal Communication, Apr. 2000.

[22] N. A. Sherwani and B. Wu, "Effective buffer insertion of clock tree for high speed VLSI circuits," *Microelectronics J.*, vol. 23, pp. 291–300, 1992.

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

[24] G. E. Tellez and M. Sarrafzadeh, "Minimal buffer insertion in clock trees with skew and slew rate constraints," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 333–342, Apr. 1997.

[25] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. IEEE Int. Symp. Circuits and Systems*, 1990, pp. 865–868.

Charles J. Alpert (S'92–M'96) received the B.S. degree in math and computational sciences and the B.A. degree in history from Stanford University, Stanford, CA, in 1991 and the Ph.D. degree in computer science from the University of California at Los Angeles in 1996.

He currently works as a Research Staff Member at the IBM Austin Research Laboratory, Austin, TX. His research interests include placement, buffer insertion, delay metrics, and anything under the realm of physical synthesis.

Dr. Alpert received the Best Paper Award at the 1994, 1995, and 2001 ACM/IEEE Design Automation Conferences. He also received the SRC's Mahboob Khan Mentor Award in 2001. Charles is the technical program chair for the ACM Tau Workshop on Timing Issues in the Specification and Synthesis of Digital Systems 2002 and the International Symposium on Physical Design 2003. Charles has served on the technical program committees for ICCAD, DAC, and ISPD and as a guest editor for TCAD.

Andrew B. Kahng received the A.B. degree in applied mathematics/physics from Harvard College, Cambridge, MA, and the M.S. and Ph.D. degrees in computer science from the University of California at San Diego (UCSD).

He was with the UCLA computer science department from 1989 to 2000, most recently as Professor and Vice-Chair. Since January 2001, he is Professor of CSE and ECE at UCSD. He has published over 160 papers in the VLSI CAD literature, centering on physical layout and performance analysis. His research interests include VLSI physical layout design and performance analysis, combinatorial and graph algorithms, and stochastic global optimization.

Prof. Kahng received the National Science Foundation Young Investigator award and a Design Automation Conference Best Paper award. He was the founding General Chair of the ACM/IEEE International Symposium on Physical Design, co-founder of the ACM Symposium on System-Level Interconnect Prediction, and since 1997 has defined the physical design roadmap for the SIA International Technology Roadmap for Semiconductors. He is currently Chair of the U.S. Design Technical Working Group for the 2001 ITRS, and technical program chair of EDP-2001 (the Electronic Design Processes symposium of the IEEE DATC). He is also on the steering committees of ISPD-2001 and SLIP-2001.

Bao Liu was born in Guilin, China, in 1973. He received the B.S. and the M.S. degrees in electrical engineering from Fudan University, Shanghai, China, in 1993 and 1996, respectively. He is currently working toward the Ph.D. degree in the Computer Science And Engineering Department, University of California San Diego.

He was with China IC Design Center, Beijing, China, from 1996 to 1998, Cadence Design Systems, Inc., San Jose, CA, in 1999 and Conexant Systems, Inc., Newport Beach, CA, in 2000. His research includes VLSI interconnect estimation, construction, and optimization.

Ion I. Măndoiu received the M.S. degree from Bucharest University, Bucharest, Romania, in 1992 and the Ph.D. degree from Georgia Institute of Technology, Atlanta, in 2000, both in computer science.

He worked as a Research Assistant at Bucharest University (1992–1995), Instructor at Georgia Institute of Technology (2000–2001), and Postgraduate Researcher at University of California, Los Angeles (2000–2001). Since 2001 he has been a Research Scientist at University of California, San Diego. He is the author of over 25 refereed scientific publications, including an Asia-South Pacific Design Automation Conference Best Paper. His research interests include approximation algorithms, VLSI physical layout design, and combinatorial optimization.

Alexander Z. Zelikovsky received the Ph.D. degree in computer science from the Institute of Mathematics of the Belorussian Academy of Sciences, Minsk, U.S.S.R. (now Belarus), in 1989.

He worked at the Institute of Mathematics in Kishinev as a senior research scholar from 1989 to 1995. Between 1992 and 1995, he visited Bonn University and the Institut fur Informatik, Saarbrueken, Germany. He was a Research Scientist at the University of Virginia (1995–1997) and a Postdoctoral Scholar at UCLA (1997–1998). Since 1999, he has been an Assistant Professor at the Computer Science Department of Georgia State University. He is the author of over 60 refereed publications. His research interests include VLSI physical layout design, discrete and approximation algorithms, combinatorial optimization, and *ad hoc* wireless networks.