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

Charles J. Alpert,† Andrew B. Kahng, Bao Liu, Ion I. Mândoiu, and Alexander Z. Zelikovsky‡

CSE Department, UCSD, La Jolla, CA 92093-0114
†IBM Corporation, 11400 Burnet Road, Austin, TX 78758
‡CS Department, Georgia State University, Atlanta, GA 30303
alpert@austin.ibm.com, {abk,bluiu,mandoiu}@cs.ucsd.edu, alexz@cs.gsu.edu

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 non-inverting) buffer type.
- For simultaneous routing and buffering with a single non-inverting 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.

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.

This work was partially supported by Cadence Design Systems, Inc., the MARCO Gigascale Silicon Research Center, NSF Grant CCR-9988331, Award No. MM2-3018 of the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research & Development Foundation for the Independent States of the Former Soviet Union (CRDF), and by the State of Georgia’s Yamacraw Initiative.

¹For 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).
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 non-critical 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 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 non-critical 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 post-placement 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 non-critical 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 analyses 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 non-inverting 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 non-inverting 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 non-inverting) buffer type.\(^2\)
- For simultaneous routing and buffering with a single non-inverting 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

\(^2\)A different algorithm for non-inverting buffers was previously given in [24].
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 non-inverting 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 non-inverting 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 $T$.

- 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.

Throughout the paper we use the following notations:

- $C_w =$ 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 $v$
- $\sigma_v =$ 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., $c(T_v) = \sum_{e \in T_v} c_e + \sum_{v \in \text{leaves}(T_v)} c_v$
- $C_U =$ given upper-bound on the capacitive load of each buffer.

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

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

---

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

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

\[
c(D_b) \leq C_U \text{ for every } b \in B \cup \{r\}
\]

### 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:

\[
\text{combined\ cost}(T) = \text{wire\ area}(T) + |B| \cdot a
\]

(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.,

\[
\text{cost}(T) = |B| + 1
\]

(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) \( c(D_b) \leq C_U \) 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, and

(c) \( \text{cost}(T) = |B| + 1 \) 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 non-inverting buffer type. The running time of each algorithm is linear in the number of sinks and the number of inserted buffers.

#### A. Single Non-Inverting Buffer Type

Our algorithm for buffering a given routing tree with a single non-inverting 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

\(^4\)We require that \( C_U > 2C_b \) since otherwise buffering is impossible for some trees.
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 $c(T_u) \leq C_U$ 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 $c(T_u) + c_{u,p} \geq c(T_v) + c_{v,p}$ for every other child $v$ of $p$.

The algorithm (see Algorithm 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.

**Algorithm 1: Routed Net Buffering (RNB)**

**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 $B$ of $T$ such that $c_b \leq C_U$ for every $b \in \{r\} \cup B$

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 $B = B' \cup \{b\}$

**Remark.** The runtime of Algorithm 1 is $O(|S| + |B|)$ (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:** Algorithm 1 finds an optimum buffering of the input tree $T$ with the given non-inverting 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.
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 Figure 1).

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 \( (B_{opt} \setminus \{b'\}) \cup \{b\} \) 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 Figure 2).

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 \( b' \) from \( B_{opt} \) since it has total capacitance equal to \( C_U \). 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) \( c(T_u) + c_{(u,p)} \leq c(D_p) \), 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: If the input to Algorithm 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 non-inverting buffer type. The greedy approach does not work in this case, and we must use dynamic programming. In bottom-up order, the algorithm (see Algorithm 2) 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 4 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:** Algorithm 2 finds an optimum buffering of the input tree $T$ with the given inverting buffer type.
Algorithm 2: Routed Net Inverting Buffering (RNIB)

**Input:** Binary routing tree \( T = (r, V, E) \) for net \( N \) with source \( r \) and sinks \( S \), sink input capacitances \( c_s \) and polarities \( \sigma_i \), upper-bound \( C_U \)

**Output:** Optimum buffering \( B \) of \( T \) consistent with sink polarities such that \( c(D_b) \leq C_U \) for every \( b \in \{r\} \cup B \)

1. \( T' = T \)
2. For each \( s \in S \) do:
   - If \( \sigma_s = + \) then \( I^+(s) = 0 \), else \( I^+(s) = C_U \)
   - \( l^-(s) = C_U - I^+(s) \)
   - **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. \( F_v(\mu) = 0 \)
   - Insert \( n^\mu(\nu) \) buffers on the edge \( (r, v) \)
   - For each fork \( (v, i, x_1, x_2) \), in reverse order of collapsing, insert \( n^\mu(\nu) \) buffers on edges \( (v, u_i) \), \( i = 1, 2 \), where \( \sigma = \mu \) if \( n^\mu(\nu) \) is odd and \( \sigma = -\mu \) if \( n^\mu(\nu) \) 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 \( (u, x_1) \) and \( (v, x_2) \)
1. For each \( (i, j) \in \{0, 1, 2\} \times \{0, 1, 2\} \) and \( \sigma \in \{+\, -\} \) do:
   - \( b_{ij} = \max(0, l_{(u,x_1)} + F_x(x_1) - i \cdot (C_U - C_B)) + \max(0, l_{(u,x_2)} + F_x(x_2) - j \cdot (C_U - C_B)) \)
   - If \( b_{ij} \leq C_U \) then \( b_{ij} = b_{ij} + (i + j)C_U \), else \( b_{ij} = \infty \) \( // i + j \) buffers are not sufficient
// Choose the topmost buffer positions
2. For each \( \sigma \in \{+\, -\} \) do: \( F^\sigma(v) = \min\{b_{ij} | i, j = 0, 1, 2\} \), \( (F^\sigma, F^\sigma) = \arg\min\{b_{ij} | i, j = 0, 1, 2\} \)
// Find minimal label and normalize the opposite polarity label
3. \( F(v) = \min\{F^+(v), F^-(v)\} \)
   - If \( F^+(v) > C_U \), then \( (F^+, F^-) = (F^+, F^+) \), \( F^+(v) = F^+(v) + C_U \)
// Increment # of buffers for both stems and restore \( v \)'s labels
4. For each \( \sigma \in \{+\, -\} \) do: \( n^\sigma(x_1) = n^\sigma(x_1) + F^\sigma, n^\sigma(x_2) = n^\sigma(x_2) + F^\sigma, F^\sigma(v) = F^\sigma(v) - (F^\sigma + F^\sigma)C_U \)
// Reduce minimal label of \( v \) to 0, remove leaves \( x_1 \) and \( x_2 \), and reduce \( v \)'s stem
5. \( l_{(u,v)} = l_{(u,v)} + F^\sigma(v), I^-(v) = I^-(v) - F^\sigma(v), I^+(v) = 0 \)
6. \( T' = T' \setminus \{x_1, x_2\} \)
7. **Reduce_stem** \( (v) \)
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 $\sup_{I} \frac{A(I)}{OPT(I)}$, 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 non-inverting) buffer type. On the positive side, for any $\varepsilon > 0$, we give a factor $2(1 + \varepsilon)(1 + \frac{1}{C_U / C_b - 2})$ approximation algorithm for MBRP with single non-inverting buffer type and a factor $4(1 + \varepsilon)(1 + \frac{1}{C_U / C_b - 2})$ 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. Figure 3 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 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 8. 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 Non-Inverting Buffer Type

In this section we show that optimal buffering of an approximate rectilinear Steiner minimum tree over the terminals (Algorithm 3) comes within a constant factor of the MBRP optimum. Below, the output of a polynomial-time RSMT algorithm with approximation factor of \( \alpha \) will be referred to as an \( \alpha \)-approximate Steiner tree.

Algorithm 3: Steiner Tree Buffering (STB)

<table>
<thead>
<tr>
<th>Input:</th>
<th>Net ( N ) with source ( r ) and set of sinks ( S ), sink input capacitances ( c_s ), upper-bound ( C_U )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>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 )</td>
</tr>
</tbody>
</table>

1. Find an \( \alpha \)-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 (Algorithm 1)

**Theorem 4:** Algorithm 3 approximates the MBRP with single non-inverting buffer type within a factor of \( 2\alpha(1 + \frac{1}{C_U/C_b - 2}) \) for every net with total sink capacitance of at least \( C_b \).

**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.,

\[
CAP = \sum_{s \in S} c_s + C_U \cdot \left( \sum_{e \in T_{opt}} l_e \right)
\]

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.,

\[
OPT \geq \frac{CAP - C_b}{C_U - C_b}
\] (3)

Let \( CAP' \) be the capacitance before buffering of the \( \alpha \)-approximate Steiner tree constructed by Algorithm 3. 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.,

\[
CAP' - C_b \leq \alpha(CAP - C_b)
\] (4)

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 Algorithm 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.,

\[
A \leq \frac{CAP' - C_b}{C_U/2 - C_b} = \frac{2CAP' - C_b}{C_U - 2 \cdot C_b}
\] (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 2.1\( \alpha \) and 2.005\( \alpha \) in Theorem 4.
Finally, inequalities (3-5) give

\[
\frac{A}{OPT} \leq 2 \frac{CAP' - C_b}{CAP - C_b} \cdot \frac{C_U - C_b}{C_U - 2 \cdot C_b} \leq 2\alpha \cdot \left(1 + \frac{1}{C_U/C_b - 2}\right)
\]

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:

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

**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 non-inverting buffers to route the signal from the source. In the worst case this solution may require as many as \(|S|/2\) inverters, plus the non-inverter 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 non-inverting 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 non-inverting buffer with the same driving strength, an even better solution is provided by Algorithm 4. In this algorithm we construct a routing tree for all sinks, buffer it with non-inverting buffers, and then make it consistent with sink polarities by replacing each non-inverting buffer by two inverters.

**Algorithm 4: Steiner Tree Inverting Buffering (STIB)**

**Input:** Net \(N\) with source \(r\) and set of sinks \(S\), sink input capacitances \(c_s\) and polarities \(\sigma_s\), 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 \(l(p^+, p) = l(b, p)\)
   - For each \(\sigma \in \{+, -\}\) add to \(T\) a Steiner tree rooted at \(b^\sigma\) and spanning all sinks with polarity \(\sigma\) in \(D_b\)
   - \(T^* = T' \setminus D_b\)
3. Return \(T\)

**Theorem 5:** Algorithm 4 approximates the MBRP with single inverting buffer type within a factor of at most \(4\alpha(1 + \frac{1}{C_U/C_b - 2})\).

**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 \(T^*\).

\(^6\)For simplicity we assume that the buffer input capacitance \(C_b\) is less than any sink capacitance. Algorithm 4 can be modified such that this assumption is
The key observation is that the optimum number of inverting buffers, $OPT$, is no less than the optimum number of non-inverting buffers $OPT'$. Let $A'$ and $A$ be the number of buffers inserted by the algorithms STB and STIB, respectively. Then, by Theorem 4, $A \leq 2 \cdot A' \leq 4\alpha(1 + \frac{1}{C_U/C_b-2})OPT' \leq 4\alpha(1 + \frac{1}{C_U/C_b-2})OPT$. 

Using Arora’s PTAS [5], Theorem 5 gives:

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

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.

V. MBRP Heuristics with Improved Practical Performance

A. Non-Inverting 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 Figure 4), modifies the Steiner tree constructed by STB in a bottom-up fashion, starting from the sinks and working towards the root. When 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 $C_U$ (Figure 4(b)).

not necessary.
Algorithm 5: Cut&Connect

**Input:** Net $N$ with source $r$ and set of sinks $S$, sink input capacitances $c$, 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. $T' = \text{Steiner tree for } S \cup \{r\}$, 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 $T'_p$, where $p$ is either $b$ or the point closest to $\text{parent}(b)$ on the shortest path between $i$ and $T'_b$.
             - whichever of the two is closer to $\text{parent}(b)$
               - If $c(T'_p) < C_U$ then
                 - Place $b'(i)$ at distance $(C_U - c(T'_p))/C_w$ from $p$, towards $\text{parent}(b)$
                 - Set $\text{gain}(i) = c(T'_p)/C_w - \text{distance}(b, b'(i))$
               - End if
             - 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
   - $B = B \cup \{b\}$, $T = T \cup T'_b$, $T' = T' \setminus T'_b$
4. Return $T \cup T'$, with buffer set $B$

Similar to Cut&Connect, the Clustering heuristic 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 Figure 4(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 $O(|B|T_{\text{run}})$, where $T_{\text{run}}$ 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.
Algorithm 6: Clustering

**Input:** Net \( N \) with source \( r \) and set of sinks \( S \), sink input capacitances \( c_s \), upper-bound \( C_U \)

**Output:** Buffered routing tree \( T = (V,E,B) \) 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 \( c(T'_{\text{parent}(v)}) > C_U \)
   // Fill the load of the subtree by connecting neighboring sinks
   \[ \text{subtree load} = c(T'_v); S' = T'_v \cap S; T = T \cup T'_v; \]
   \[ q = \text{sink in } S \setminus S' \text{ closest to } S'; p = \text{sink of } S' \text{ closest to } q \]
   While \( \text{subtree load} + C_u l(p,q) + c_q < C_U \) do:
   \[ \text{subtree load} = \text{subtree load} + C_u l(p,q) + c_q \]
   \[ S' = S' \cup \{q\}; T = T + (p,q) \]
   \[ q = \text{sink in } S \setminus S' \text{ closest to } S'; p = \text{sink of } S' \text{ closest to } q \]
   End while
   Place buffer \( b \) at distance \( (C_U - \text{subtree load}) / C_u \) 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 \( B \)

**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 (Algorithm 7) 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.
Algorithm 7: Steiner Tree Inverting Buffering with Swapping (STIB-S)

**Input:** Net $N$ with source $r$ and set of sinks $S$, sink input capacitances $c_s$ and polarities $\sigma_s$, upper-bound $C_U$

**Output:** Buffered routing tree $T = (V, E, B)$ 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 $T$

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, Algorithm 8 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 its full capacity. Similar to the STIB and STIB-S algorithms, whenever an inverter is inserted by Algorithm 8 the driven sinks/buffers are connected to the inverter by duplicating paths of the routing tree (see Figure 5).

In Algorithm 8 we use some additional notation. For every node $v$ of a tree $T$, let $D^+_v$ ($D^-_v$) be the tree 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 $Br^+_v = D^+_v + (v, \text{parent}(v))$ if $S^+(v) > 0$ and $Br^+_v = \emptyset$ otherwise, and, similarly, $Br^-_v = D^-_v + (v, \text{parent}(v))$ if $S^-(v) > 0$ and $Br^-_v = \emptyset$ otherwise.
Algorithm 8: Steiner Tree Inverting Buffering with Load Filling (STIB-LF)

Input: Net $N$ with source $r$ and set of sinks $S$, sink input capacitances $c_s$ and polarities $\sigma_s$, upper-bound $C_U$

Output: Buffered routing tree $T = \langle r, V, E, B \rangle$ for $N$ consistent with sink polarities such that $c(D_b) \leq C_U$ for every $b \in \{r\} \cup B$

1. Find an $\alpha$-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 $B_{u_1}^+, B_{u_2}^-, B_{u_1}^-, B_{u_2}^+$, 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 $B_{u_1}^+, B_{u_2}^+$
   If $S^-(v) > C_U$ then insert inverter with “+” polarity in the highest feasible position on the maximum capacitance branch among $B_{u_1}^-, B_{u_2}^-$
   Else exit repeat loop
5. Return $T$

VI. EXPERIMENTAL RESULTS

We have implemented the Routed Net Buffering (RNB) and Routed Net Inverting Buffering (RNIB) algorithms for optimally buffering a given tree with a single non-inverting, respectively inverting, buffer type, the Cut&Connect and Clustering heuristics for MBRP with single non-inverting buffer type, as well as the Steiner Tree Inverting Buffering with Swapping (STIB-S) and the Steiner Tree Inverting Buffering with Load Filling (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, $C_w = 0.177fF/\mu m$, $C_b = 37.5fF$, while sink input capacitances are varying between $2.04fF$ and $200fF$.

Table I gives the results obtained by the three heuristics for non-inverting 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 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. 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. CONCLUSIONS AND FUTURE RESEARCH

In this paper we have addressed a minimum-buffered routing problem which asks for bounded input rise/fall time for all buffers and sinks. We have analyzed the approximation complexity of this problem and given provably-good algorithms for buffering with a single inverting or non-inverting 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.

7The 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.
A natural research direction is to extend the results in this paper to MBRP with multiple buffer/inverter 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 2-pin net is NP-hard. Our ongoing research addresses the case of libraries with small number of buffer types. We also investigate multi-source 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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>MST+RNB</th>
<th>MST+Cut&amp;Conn.</th>
<th>MST+Cluster</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>#term.</td>
<td>C&lt;sub&gt;t&lt;/sub&gt;</td>
<td>#b</td>
<td>WL</td>
<td>time</td>
</tr>
<tr>
<td>500</td>
<td>17</td>
<td>25</td>
<td>0.81</td>
<td>16</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>25</td>
<td>0.81</td>
<td>8</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
<td>25</td>
<td>0.81</td>
<td>4</td>
</tr>
<tr>
<td>4000</td>
<td>2</td>
<td>25</td>
<td>0.81</td>
<td>1</td>
</tr>
<tr>
<td>500</td>
<td>34</td>
<td>68</td>
<td>0.97</td>
<td>34</td>
</tr>
<tr>
<td>830</td>
<td>17</td>
<td>68</td>
<td>0.97</td>
<td>17</td>
</tr>
<tr>
<td>1900</td>
<td>56</td>
<td>45</td>
<td>1.02</td>
<td>54</td>
</tr>
<tr>
<td>2400</td>
<td>74</td>
<td>58</td>
<td>1.07</td>
<td>70</td>
</tr>
<tr>
<td>500</td>
<td>147</td>
<td>89</td>
<td>1.15</td>
<td>144</td>
</tr>
<tr>
<td>2600</td>
<td>70</td>
<td>89</td>
<td>1.14</td>
<td>67</td>
</tr>
<tr>
<td>1000</td>
<td>33</td>
<td>58</td>
<td>1.06</td>
<td>32</td>
</tr>
<tr>
<td>2000</td>
<td>17</td>
<td>58</td>
<td>1.06</td>
<td>17</td>
</tr>
<tr>
<td>4000</td>
<td>8</td>
<td>58</td>
<td>1.07</td>
<td>8</td>
</tr>
<tr>
<td>8000</td>
<td>4</td>
<td>58</td>
<td>1.07</td>
<td>3</td>
</tr>
<tr>
<td>500</td>
<td>44</td>
<td>266</td>
<td>2.63</td>
<td>236</td>
</tr>
<tr>
<td>12000</td>
<td>244</td>
<td>266</td>
<td>2.83</td>
<td>236</td>
</tr>
<tr>
<td>22000</td>
<td>56</td>
<td>266</td>
<td>2.63</td>
<td>55</td>
</tr>
<tr>
<td>34000</td>
<td>28</td>
<td>266</td>
<td>2.64</td>
<td>28</td>
</tr>
<tr>
<td>8000</td>
<td>13</td>
<td>266</td>
<td>2.63</td>
<td>13</td>
</tr>
</tbody>
</table>

TABLE I

| NUMBER OF BUFFERS, WIRELENGTH (MM), AND RUNTIME (CPU SEC.) FOR THE ROUTED NET BUFFERING (RNB), CUT&CONNECT, AND CLUSTERING HEURISTICS FOR NON-INVERTING BUFFER INSERTION. |


<table>
<thead>
<tr>
<th>Benchmark</th>
<th>MST+RNIB</th>
<th>MST+STIB-S</th>
<th>MST+STIB-LF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#term:</td>
<td>#term:</td>
<td>#term:</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>Ub</td>
<td>WL time</td>
</tr>
<tr>
<td>330</td>
<td>500</td>
<td>14 25 0.99</td>
<td>28 31 0.96</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>16 25 0.99</td>
<td>11 30 0.96</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>8 25 0.98</td>
<td>6 27 0.96</td>
</tr>
<tr>
<td></td>
<td>4000</td>
<td>4 25 0.99</td>
<td>3 27 0.96</td>
</tr>
<tr>
<td></td>
<td>8000</td>
<td>0 25 0.99</td>
<td>0 25 0.96</td>
</tr>
<tr>
<td>830</td>
<td>500</td>
<td>64 68 1.09</td>
<td>50 92 1.04</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>34 68 1.08</td>
<td>26 93 1.04</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>16 68 1.08</td>
<td>12 89 1.04</td>
</tr>
<tr>
<td></td>
<td>4000</td>
<td>6 68 1.08</td>
<td>5 90 1.04</td>
</tr>
<tr>
<td></td>
<td>8000</td>
<td>2 68 1.08</td>
<td>2 76 1.05</td>
</tr>
<tr>
<td>1900</td>
<td>500</td>
<td>112 45 1.10</td>
<td>80 51 1.07</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>56 45 1.09</td>
<td>40 50 1.07</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>26 45 1.09</td>
<td>19 49 1.07</td>
</tr>
<tr>
<td></td>
<td>4000</td>
<td>12 45 1.10</td>
<td>9 47 1.06</td>
</tr>
<tr>
<td></td>
<td>8000</td>
<td>4 45 1.09</td>
<td>3 48 1.06</td>
</tr>
<tr>
<td>2400</td>
<td>500</td>
<td>148 58 1.14</td>
<td>109 66 1.09</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>66 58 1.15</td>
<td>47 65 1.09</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>34 58 1.15</td>
<td>25 62 1.09</td>
</tr>
<tr>
<td></td>
<td>4000</td>
<td>16 58 1.14</td>
<td>11 61 1.09</td>
</tr>
<tr>
<td></td>
<td>8000</td>
<td>8 58 1.15</td>
<td>6 61 1.09</td>
</tr>
<tr>
<td>2600</td>
<td>500</td>
<td>292 89 1.17</td>
<td>215 107 1.12</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>140 89 1.17</td>
<td>102 102 1.12</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>66 89 1.16</td>
<td>49 101 1.12</td>
</tr>
<tr>
<td></td>
<td>4000</td>
<td>34 89 1.17</td>
<td>25 98 1.12</td>
</tr>
<tr>
<td></td>
<td>8000</td>
<td>16 89 1.17</td>
<td>12 97 1.13</td>
</tr>
<tr>
<td>12000</td>
<td>500</td>
<td>484 266 2.02</td>
<td>354 305 1.82</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>232 266 2.01</td>
<td>168 298 1.82</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>66 266 2.02</td>
<td>49 288 1.81</td>
</tr>
<tr>
<td></td>
<td>4000</td>
<td>34 266 2.02</td>
<td>25 288 1.81</td>
</tr>
<tr>
<td></td>
<td>8000</td>
<td>16 266 2.02</td>
<td>12 288 1.81</td>
</tr>
<tr>
<td>22000</td>
<td>500</td>
<td>282 1396 3.23</td>
<td>2069 1706 2.69</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>1344 1396 3.14</td>
<td>974 1646 2.67</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>660 1396 3.09</td>
<td>476 1583 2.68</td>
</tr>
<tr>
<td></td>
<td>4000</td>
<td>328 1396 3.07</td>
<td>236 1550 2.68</td>
</tr>
<tr>
<td></td>
<td>8000</td>
<td>160 1396 3.07</td>
<td>114 1509 2.68</td>
</tr>
<tr>
<td>34000</td>
<td>500</td>
<td>1606 990 4.31</td>
<td>1173 1133 3.68</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>774 990 4.26</td>
<td>563 1109 3.67</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>382 990 4.22</td>
<td>274 1087 3.67</td>
</tr>
<tr>
<td></td>
<td>4000</td>
<td>190 990 4.23</td>
<td>138 1067 3.66</td>
</tr>
<tr>
<td></td>
<td>8000</td>
<td>90 990 4.21</td>
<td>67 1068 3.67</td>
</tr>
</tbody>
</table>

TABLE II

NUMBER OF BUFFERS, WIRELENGTH (MM), AND RUNTIME (CPU SEC.) FOR ROUTED NET INVERTING BUFFERING (RNIB), STEINER TREE INVERTING BUFFERING WITH SWAPPING (STIB-S), AND STEINER TREE INVERTING BUFFERING WITH LOAD FILLING (STIB-LF) HEURISTICS ON TESTCASES WITH ALL SINKS OF THE SAME POLARITY.
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>MST+RNIB</th>
<th>SplitMST+RNIB</th>
<th>MST+STIB-S</th>
<th>SplitMST+STIB-S</th>
<th>MST+STIB-LF</th>
<th>SplitMST+STIB-LF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># #</td>
<td># #</td>
<td># #</td>
<td># #</td>
<td># #</td>
<td># #</td>
</tr>
<tr>
<td></td>
<td>cWL time</td>
<td>cWL time</td>
<td>cWL time</td>
<td>cWL time</td>
<td>cWL time</td>
<td>cWL time</td>
</tr>
<tr>
<td>300</td>
<td>177.25</td>
<td>196.21</td>
<td>144.21</td>
<td>196.21</td>
<td>122.27</td>
<td>196.21</td>
</tr>
<tr>
<td>400</td>
<td>187.25</td>
<td>196.21</td>
<td>144.21</td>
<td>196.21</td>
<td>122.27</td>
<td>196.21</td>
</tr>
<tr>
<td>800</td>
<td>177.25</td>
<td>196.21</td>
<td>144.21</td>
<td>196.21</td>
<td>122.27</td>
<td>196.21</td>
</tr>
</tbody>
</table>

| 830       | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 400       | 187.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 800       | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |

| 1900      | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 400       | 187.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 800       | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |

| 2100      | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 400       | 187.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 800       | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |

| 2400      | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 400       | 187.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 800       | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |

| 2600      | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 400       | 187.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 800       | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |

| 500       | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 400       | 187.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 800       | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |

| 12000     | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 400       | 187.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 800       | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |

| 22000     | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 400       | 187.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 800       | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |

| 34000     | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 400       | 187.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |
| 800       | 177.25   | 196.21      | 144.21   | 196.21        | 122.27    | 196.21        |

TABLE III
Number of buffers, wirelength (mm), and runtime (CPU sec.) for Routed Net Inverting Buffering (RNIB), Steiner Tree Inverting Buffering with Swapping (STIB-S), and Steiner Tree Inverting Buffering with Load Filling (STIB-LF) heuristics on testcases with random sink polarities. SplitMST variants correspond to independently buffering minimum spanning trees for the positive and negative sinks.