Minimum-Buffered Routing of Non-Critical Nets for Slew Rate and Reliability Control*

Charles Alpert,† Andrew B. Kahng, Bao Liu, Ion Măndoiu, and Alexander 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,bliu,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 include the following.

- 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 give a factor $2(1+\varepsilon)$ approximation algorithm and prove that no algorithm can guarantee a factor smaller than 2 unless P=NP. For the case of a single inverting buffer type, we give a factor $4(1+\varepsilon)$ approximation algorithm.
- 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.

1 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 (Tellez and Sarrafzadeh [24] formally prove one such equivalence). Such bounds also improve reliability with respect to hot-carrier oxide breakdown (hot electrons) [9, 11] 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 and NSF Grant CCR-9988331.

[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).

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 followings: (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 ECO 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.

1.1 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, 7, 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 mainly consist 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 [8, 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.*

1.2 Our Contributions

Our contributions as 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 give a factor $2(1+\varepsilon)$ approximation algorithm and prove that no algorithm can guarantee a factor smaller than 2 unless P=NP. For the case of a single inverting buffer type, we give a factor $4(1+\varepsilon)$ approximation algorithm.
- 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.

[2] A different algorithm for non-inverting buffers was previously given in [24].

1.3 Organization of the Paper

We formally define MBRP in Section 2. Then, in Section 3, 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 4 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 5, and present experimental results comparing the runtime/quality tradeoffs of the proposed heuristics in Section 6. We conclude in Section 7 with directions for future research.

2 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.

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

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

Load Model

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

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

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

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

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:

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

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

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

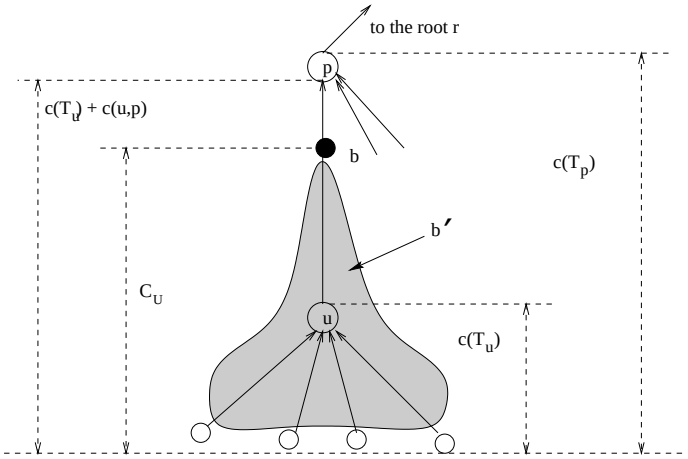

Figure 1: Since $c(T_b) = C_U$, the tree T_b (shaded area) must contain a buffer b' in any optimum buffering B_{opt}. $(B_{opt} \setminus \{b'\}) \cup \{b\}$ is then an optimum buffering of T containing b.

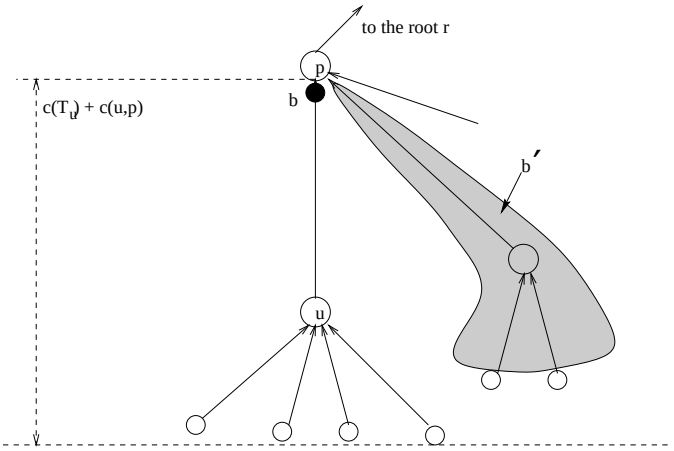

Figure 2: When b' is located on a different branch (shaded area) than that of the heaviest child u, $c(T_u) + c_{(u,p)} \geq c(D_{b'})$. Hence, $(B_{opt} \setminus \{b'\}) \cup \{b\}$ is an optimum buffering of T containing b.

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.,

$$cost(T) = |B| + 1 \tag{2}$$

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 parities), 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, 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) $cost(T) = |B| + 1$ is minimum among all buffered routing trees satisfying conditions (a) and (b).

3 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.

3.1 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]. Before 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. Due to space limitations we only give here a simple recursive description of the algorithm; the details of an $O(|S| + |B|)$ time implementation can be found in [3].

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 $c(D_b) \leq C_U$ 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 $B = B' \cup \{b\}$

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' of B_{opt}. We claim that $(B_{opt} \setminus \{b'\}) \cup \{b\}$ 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 *If the input to Algorithm 1 is a binary routing tree, then the lumped capacitive 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 4. 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.

3.2 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 σ_s, 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 $l^+(s) = 0$, 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:
   $l^\sigma_{ij} = \max\{0, l_{(v,x_1)} + l^\sigma(x_1) - i \cdot (C_U - C_b)\}$
   $+ \max\{0, l_{(v,x_2)} + l^\sigma(x_2) - j \cdot (C_U - C_b)\}$
   If $l^\sigma_{ij} \leq C_U$ then $l^\sigma_{ij} = l^\sigma_{ij} + (i + j)C_U$
   Else, $l^\sigma_{ij} = \infty$ // $i + j$ buffers are not sufficient

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

// 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)

4 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 $\sup \frac{A(I)}{OPT(I)}$, 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, we give a factor $(2 + \varepsilon)$ approximation algorithm for MBRP with single non-inverting buffer type, and a factor $(4 + \varepsilon)$ approximation algorithm for MBRP with single inverting buffer type.

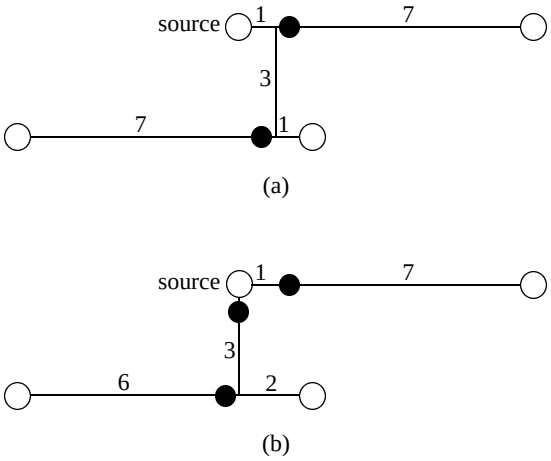

Figure 3: (a) Optimum buffered routing of a 4 terminal net with non-Hanan grid edge. (b) Best buffered routing on the Hanan grid.

4.1 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 [10]. 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.

4.2 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 α will be referred to as an α-*approximate Steiner tree*.

Algorithm 3: Steiner Tree Buffering (STB)

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 $c(D_b) \le C_U$ 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 (Algorithm 1)

Theorem 4 *Algorithm 3 approximates the MBRP with single non-inverting buffer type within a factor of $2\alpha(1+\varepsilon)$, where $\varepsilon = \frac{1}{C_U/C_b - 2}$.*[4]

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.,

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

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 $OPT \cdot C_U \ge CAP + (OPT-1)C_b$, i.e.,

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

Let CAP' be the capacitance before buffering of the α-approximate Steiner tree constructed by Algorithm 3. Then $CAP' - s \le \alpha(CAP - s)$, where $s = \sum_{s \in S} c_s$ is the total input capacitance of the sinks. Since $s \ge C_b$, this gives $CAP' \le \alpha CAP - (\alpha - 1)s \le \alpha(CAP - C_b) + C_b$, i.e.,

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

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 $C_U/2$. Thus, $CAP' + (A-1)C_b \ge A \cdot (C_U/2)$, i.e.,

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

Finally, inequalities (3-5) give

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

□

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 *The MBRP with single non-inverting buffer type can be approximated in polynomial time within a factor of $2(1+\varepsilon)$ for any $\varepsilon > \frac{1}{C_U/C_b - 2}$.*

4.3 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 a non-inverting buffer occupies close to twice the area of an inverter 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 iteratively replacing non-inverting buffers by inverters. In the worst case each non-inverting buffer is replaced by a pair of inverters, but if all sinks driven by a buffer have the same polarity then a single inverter replacement is sufficient.

[4] We require that $C_U/C_b > 2$ since otherwise buffering is impossible. In practice $C_U/C_b \gg 2$; in our benchmarks the ratio varies between 12 and 200, which corresponds to a value of ε between 0.1 and 0.005.

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 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 only sinks with the same polarity then
   Replace b by an inverter and add b's stage to T
   Else // b drives both positive and negative sinks
   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 $\sigma \in \{+,-\}$ do:
   Add to T a Steiner tree rooted at b^σ and spanning all sinks with polarity σ in D_b
   End for
   End if
   $T' = T' \setminus D_b$
   End for
3. Return T

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

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'.[5]

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, $A \leq 2 \cdot A' \leq 4\alpha(1+\varepsilon)OPT' \leq 4\alpha(1+\varepsilon)OPT$. □

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

Corollary 2 *The MBRP with single inverting buffer type can be approximated in polynomial time within a factor of* $4(1+\varepsilon)$ *for any* $\varepsilon > \frac{1}{C_U/C_b - 2}$.

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. Here we note that a practical, if not theoretical, improvement of Algorithm 4 is to compute the placement of inverters by a polarity-aware version of the RNB algorithm, instead of using the locations of the non-inverting buffers inserted by STB.

5 MBRP Heuristics with Improved Practical Performance

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, 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 re-connecting it to the closest point in T_b.

[5] 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 not necessary.

Algorithm 5: Cut&Connect

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
   Place $b'(i)$ 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
   $B \leftarrow B \cup \{b\}$; $T = T \cup T'_b$
   $T' = T' \setminus T'_b$
   End while
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 main differences between Clustering and Cut&Connect are in the way buffer loads are filled (Clustering adds one sink at a time, as opposed to a whole subtree in Cut&Connect) and in the fact that Clustering recomputes the Steiner tree after chopping off each buffer stage. 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 $T = (r,V,E,B)$ 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 $subtree\_load + C_w l_{(p,q)} < C_U$ do:
   $subtree\_load = subtree\_load + C_w l_{(p,q)}$
   $S' = S' \cup \{q\}$; $T = T + (p,q)$
   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
   $B \leftarrow B \cup \{b\}$; $S = (S \setminus S') \cup \{b\}$
   T' = Steiner tree for $S \cup \{r\}$, rooted at r
   End while
4. Return $T \cup T'$, with buffer set B

6 Experimental Results

We have implemented the RNB and RNIB algorithms for optimally buffering a given tree with a single non-inverting, respectively inverting, buffer type, as well as the Cut&Connect and Clustering heuristics for MBRP with single non-inverting buffer type. Table 1 gives the number of buffers inserted by the four algorithms on datasets extracted from recent industrial designs. In these experiments, all algorithms start with the minimum spanning tree over given terminals. For comparison, Table 1 includes the lower bound (3) on the optimum number of buffers.[6]

The results show that the Clustering heuristic finds consistently better solutions than the Cut&Connect heuristic, which in turn is consistently better than the STB algorithm. The Clustering heuristic comes closest to the computed lower bound, especially for large values of C_U, i.e, when few buffers are inserted. The seemingly larger room for improvement for the larger nets may be caused by the inaccuracy of the lower bound. The Cut&Connect and Clustering heuristics modify the tree in order to decrease the number of buffers, this results in a small wirelength increase (1-2%) compared to the length of the initial MST.

The RNIB results show that, for a fixed routing tree, the number of buffers that need to be inserted in order to enforce polarity constraints is 20–100% larger than the number of buffers needed without polarity constraints (the increase in buffer area depends on the relative size of inverting vs. non-inverting buffers with the same driving strength). We are currently exploring practical heuristics based on the STIB algorithm to reduce the number of inserted inverters by simultaneous routing and buffering.

7 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.

Our ongoing research addresses (i) multi-source formulations, in which the buffer solution should be legal for multiple rooted orientations of the tree, and (ii) multi-constraint formulations, in which, e.g., input capacitance and fanout must be upper-bounded simultaneously. We have already obtained encouraging preliminary results for these extensions.

References

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

[2] C. Alpert, A. Devgan, and S.T. Quay. Buffer insertion for noise and delay optimization. *IEEE Transactions on Computer-Aided Design*, 18:1633–1645, 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. Technical Report CS2001-0681, Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA, 2001.

[4] C.J. Alpert, R.G. Gandham, J.L. Neves, and S.T. Quay. Buffer library selection. In *IEEE International Conference on Computer Design*, pages 221–226, 2000.

[6] Formula (3) requires the length of the optimum rectilinear Steiner tree for the terminals. The values reported in Table 1 are based on the length of the Steiner trees computed by the batched iterated 1-Steiner heuristic [12] which is known to come to within 0.5% of the optimum on the average.

[5] S. Arora. Polynomial time approximation scheme for Euclidean TSP and other geometric problems. *Journal of the ACM*, 45:753–782, 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 Conference*, pages 69–99. MIT Press, 1989.

[7] W. Chen, C.-T. Hsieh, and M. Pedram. Simultaneous gate sizing and fanout optimization. In *Proceedings IEEE-ACM International Conference on Computer-Aided Design*, pages 374–378, 2000.

[8] M. Edahiro and R.J. Lipton. Clock buffer placement algorithm for wire-delay-dominated timing model. In *Proc. of the Great Lakes Symposium on VLSI*, pages 143–147, 1996.

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

[10] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32:826–834, 1977.

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

[12] A. B. Kahng and G. Robins. A new class of iterative Steiner tree heuristics with good performance. *IEEE Transactions on Computer-Aided Design*, 11:893–902, 1992.

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

[14] S. Kundu and J. Misra. A linear tree partitioning algorithm. *SIAM Journal on Computing*, 6:151–154, 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*, 31:437–447, 1996.

[16] J. Lillis, C.-K. Cheng, and T.-T. Lin. Similtaneous routing and buffer insertion for high performance interconnect. In *Proc. of the Great Lakes Symposium on VLSI*, pages 148–153, 1996.

[17] S. Lin and M. Marek-Sadowska. A fast and efficient algorithm for determining fanout trees in large networks. In *Proc. of the European Design Automation Conference*, pages 539–544, 1991.

[18] T. Okamoto and J. Cong. Buffered steiner tree construction with wire sizing for interconnect layout optimization. In *Proceedings IEEE-ACM International Conference on Computer-Aided Design*, pages 44–49, 1996.

[19] S. Pullela, N. Menezes, J. Omar, and L.T. Pillage. Skew and delay optimization for reliable buffered clock trees. In *IEEE-ACM International Conference on Computer-Aided Design*, pages 556–559, 1993.

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

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

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

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

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

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

Benchmark		MST+RNB		MST+Cut&Conn.		MST+Cluster		MST+RNIB		Lower
#terminals	C_U	#b	runtime	#b	runtime	#b	runtime	#b	runtime	Bound
330	500	17	0.81	16	0.83	16	0.94	21	0.83	15
330	1000	8	0.81	8	0.82	7	0.83	11	0.82	7
330	2000	4	0.81	4	0.83	3	0.81	7	0.82	2
330	4000	2	0.81	1	0.82	1	0.78	4	0.83	0
330	8000	0	0.81	0	0.82	0	0.78	0	0.82	0
830	500	34	0.97	34	0.97	33	2.05	42	0.96	32
830	1000	17	0.97	16	0.96	16	1.32	22	0.96	15
830	2000	8	0.97	8	0.96	8	1.06	10	0.96	6
830	4000	3	0.97	3	0.96	3	0.95	3	0.96	2
830	8000	1	0.97	1	0.96	1	0.88	1	0.97	0
1900	500	56	1.02	54	1.33	51	3.27	97	1.07	49
1900	1000	28	1.02	26	1.43	24	1.93	53	1.08	23
1900	2000	13	1.01	12	1.33	12	1.36	25	1.07	11
1900	4000	6	1.02	6	1.18	5	1.06	11	1.07	5
1900	8000	2	1.01	2	1.05	2	0.94	4	1.07	1
2400	500	74	1.07	70	1.61	64	4.93	133	1.17	62
2400	1000	33	1.06	32	1.43	31	2.65	56	1.18	29
2400	2000	17	1.06	17	1.87	15	1.73	32	1.18	14
2400	4000	8	1.07	8	1.45	7	1.29	15	1.18	7
2400	8000	4	1.07	3	1.35	3	1.07	8	1.18	2
2600	500	147	1.15	144	1.73	134	10.39	226	1.24	128
2600	1000	70	1.14	67	1.99	63	5.40	123	1.25	61
2600	2000	33	1.14	32	1.72	31	3.02	60	1.24	30
2600	4000	17	1.14	16	1.92	15	1.96	32	1.24	14
2600	8000	8	1.15	8	1.61	7	1.39	15	1.25	6
12000	500	244	2.63	236	9.27	222	106.83	420	3.05	184
12000	1000	116	2.63	113	11.54	106	46.90	207	3.05	88
12000	2000	56	2.63	55	12.42	52	21.25	103	3.06	42
12000	4000	28	2.64	28	13.32	25	10.59	55	3.06	20
12000	8000	13	2.63	13	8.1	12	5.82	26	3.07	9
22000	500	1418	4.39	1395	21.62	1305	1172.75	2094	5.10	1197
22000	1000	674	4.39	656	30.36	613	540.28	1126	5.10	575
22000	2000	330	4.39	319	49.58	298	257.99	583	5.11	282
22000	4000	164	4.39	159	95.40	146	121.24	297	5.11	139
22000	8000	80	4.39	78	106.98	72	60.33	145	5.14	68
34000	500	806	6.59	778	39.13	729	890.01	1387	7.74	591
34000	1000	388	6.58	374	58.55	350	424.81	696	7.76	283
34000	2000	191	6.58	153	89.04	171	208.79	354	7.75	138
34000	4000	95	6.57	92	147.62	84	103.59	179	7.74	68
34000	8000	45	6.57	44	113.80	42	49.25	85	7.76	33

Table 1: Number of buffers inserted and runtime of the four heuristics on eight industrial datasets. For all four 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. The lower bound has been calculated according to (3) with RSMT length estimated using the BI1S heuristic [12]. For all datasets, $C_w = 0.177\,fF/\mu$m and $C_b = 37.5\,fF$; sink input capacitances varies between $2.04\,fF$ and $200\,fF$.