

# An $O(bn^2)$ Time Algorithm for Optimal Buffer Insertion with $b$ Buffer Types

Zhuo Li, *Student Member, IEEE* and Weiping Shi, *Senior Member, IEEE*

**Abstract**—Buffer insertion is a popular technique to reduce the interconnect delay. The classic buffer insertion algorithm of van Ginneken has time complexity  $O(n^2)$ , where  $n$  is the number of buffer positions. Lillis, Cheng and Lin extended van Ginneken’s algorithm to allow  $b$  buffer types in time  $O(b^2n^2)$ . For modern design libraries that contain hundreds of buffers, it is a serious challenge to balance the speed and performance of the buffer insertion algorithm.

In this paper, we present a new algorithm that computes the optimal buffer insertion in  $O(bn^2)$  time. The reduction is achieved by the observation that the  $(Q, C)$  pairs of the candidates that generate the new candidates must form a convex hull. On industrial test cases, the new algorithm is faster than the previous best buffer insertion algorithms by orders of magnitude.

Since van Ginneken’s algorithm with multiple buffer types are used by most existing algorithms on buffer insertion and buffer sizing, our new algorithm improves the performance of all these algorithms.

**Index Terms**—Buffer insertion, interconnect, Elmore delay, routing, data structure

## I. INTRODUCTION

DELAY optimization techniques for interconnect are increasingly important for achieving timing closure of high performance designs. One popular technique for reducing interconnect delay is buffer insertion. A recent study by Saxena *et al* [1] projects that 35% of all cells will be intra-block repeaters for the 45 nm node. Consequently, algorithms that can efficiently insert buffers are essential for the design automation tools.

In 1990, van Ginneken [2] proposed an optimal buffer insertion algorithm for one buffer type. His algorithm has time complexity  $O(n^2)$ , where  $n$  is the number of candidate buffer positions. Lillis, Cheng and Lin [3] extended van Ginneken’s algorithm to allow  $b$  buffer types in time  $O(b^2n^2)$ . Recently, Shi and Li [4] presented a new algorithm with time complexity  $O(n \log n)$  for 2-pin nets, and  $O(n \log^2 n)$  for multi-pin nets, for one buffer type. Several works have built upon van Ginneken’s algorithm and its extension for multiple buffer types to include wire sizing [3], simultaneous tree construction [5–8], noise constraints [9] and resource minimization [3], [10].

Modern design libraries may contain hundreds of different buffers with different input capacitances, driving resistances, intrinsic delays, power levels, etc. If every buffer available for the given technology is allowed, it is stated in [11] that the current algorithms could possibly take days or even weeks

for large designs since all these algorithms are quadratic in terms of  $b$ . Alpert *et al* [11] studied how to reduce the size of the buffer library with a clustering algorithm. Though the buffer library size is reduced, the solution quality is degraded accordingly.

In this paper, we propose a new algorithm that performs optimal buffer insertion with  $b$  buffer types in  $O(bn^2)$  time. Our speedup is achieved by the observation that the candidates that generate new buffered candidates must lie on the convex hull of  $(Q, C)$ . Experimental results show that our algorithm is significantly faster than previous best algorithm.

Section II formulates the problem. Section III describes the new algorithm. Simulation results are given in Section IV. Extension to other problems is in Section V and conclusions are given in Section VI.

## II. PRELIMINARY

A net is given as a routing tree  $T = (V, E)$ , where  $V = \{s_0\} \cup V_s \cup V_n$ , and  $E \subseteq V \times V$ . Vertex  $s_0$  is the *source* vertex and also the root of  $T$ ,  $V_s$  is the set of *sink* vertices, and  $V_n$  is the set of *internal* vertices. Each sink vertex  $s \in V_s$  is associated with sink capacitance  $C(s)$  and required arrival time  $RAT(s)$ . A buffer library  $B$  contains different types of buffers and its size is represented by  $b$ . For each buffer type  $B_i \in B$ , the intrinsic delay is  $K(B_i)$ , driving resistance is  $R(B_i)$ , and input capacitance is  $C(B_i)$ . A function  $f : V_n \rightarrow 2^B$  specifies the types of buffers allowed at each internal vertex. Each edge  $e \in E$  is associated with lumped resistance  $R(e)$  and capacitance  $C(e)$ .

Following previous researchers [2], [3], [5], [6], [12], we use the Elmore delay for the interconnect and the linear delay for buffers. For each edge  $e = (v_i, v_j)$ , signals travel from  $v_i$  to  $v_j$ . The Elmore delay of  $e$  is

$$D(e) = R(e) \left( \frac{C(e)}{2} + C(v_j) \right),$$

where  $C(v_j)$  is the downstream capacitance at  $v_j$ . For any buffer type  $B_i$  at vertex  $v_j$ , the buffer delay is

$$D(v_j) = R(B_i) \cdot C(v_j) + K(B_i),$$

where  $C(v_j)$  is the downstream capacitance at  $v_j$ . When a buffer  $B_i$  is inserted, the capacitance viewed from the upper stream is  $C(B_i)$ .

For any vertex  $v \in V$ , let  $T(v)$  be the subtree downstream from  $v$ , and with  $v$  being the root. Once we decide where to insert buffers in  $T(v)$ , we have a *candidate*  $\alpha$  for  $T(v)$ . The delay from  $v$  to sink  $s \in T(v)$  under  $\alpha$  is

$$D(v, s, \alpha) = \sum_{e=(v_i, v_j)} (D(v_i) + D(e)),$$

This research was supported by the NSF grants CCR-0098329, CCR-0113668, EIA-0223785, ATP grant 512-0266-2001.

Z. Li is with Department of Electrical Engineering, Texas A&M University, College Station, Texas 77843, USA. Email: zhuoli@ee.tamu.edu .

W. Shi is with Department of Electrical Engineering, Texas A&M University, College Station, Texas 77843, USA. Email: wshi@ee.tamu.edu .

where the sum is over all edges  $e$  in the path from  $v$  to  $s$ . If  $v_i$  is a buffer in  $\alpha$ , then  $D(v_i)$  is the buffer delay. If  $v_i$  is not a buffer in  $\alpha$ , then  $D(v_i) = 0$ . The slack of  $v$  under  $\alpha$  is

$$Q(v, \alpha) = \min_{s \in T(v)} \{RAT(s) - D(v, s, \alpha)\}.$$

**Buffer Insertion Problem:** Given routing tree  $T = (V, E)$ , sink capacitance  $C(s)$  and  $RAT(s)$  for each sink  $s$ , capacitance  $C(e)$  and resistance  $R(e)$  for each edge  $e$ , possible buffer position  $f$ , and buffer library  $B$ , find a candidate  $\alpha$  for  $T$  that maximizes  $Q(s_0, \alpha)$ .

The effect of a candidate to the upstream is described by slack  $Q$  and downstream capacitance  $C$  [2]. Define  $C(v, \alpha)$  as the downstream capacitance at node  $v$  under candidate  $\alpha$ . For any two candidates  $\alpha_1$  and  $\alpha_2$  of  $T(v)$ , we say  $\alpha_1$  *dominates*  $\alpha_2$ , if  $Q(v, \alpha_1) \geq Q(v, \alpha_2)$  and  $C(v, \alpha_1) \leq C(v, \alpha_2)$ . The set of *nonredundant candidates* of  $T(v)$ , which we denote as  $N(v)$ , is the set of candidates such that no candidate in  $N(v)$  dominates any other candidate in  $N(v)$ , and every candidate of  $T(v)$  is dominated by some candidates in  $N(v)$ . Once we have  $N(s_0)$ , the candidate that gives the maximum  $Q(s_0, \alpha)$  can be found easily. The number of total nonredundant candidates is at most  $n + 1$  for one buffer type [2] and  $bn + 1$  for  $b$  buffer types [3], where  $n$  is the number of candidate buffer positions.

### III. NEW ALGORITHM

The previous best algorithm for multiple buffer types by Lillis, Cheng and Lin consists of three major operations: 1) adding buffers at a buffer position in  $O(b^2n)$  time, 2) adding a wire in  $O(bn)$  time, and 3) merging two branches in  $O(bn_1 + bn_2)$  time, where  $n_1$  and  $n_2$  are the numbers of buffer positions in the two branches. As a result, their algorithm has time complexity  $O(b^2n^2)$ . Note that the bottleneck of their algorithm is adding buffers. Their algorithm takes  $O(b^2n)$  time because it takes  $O(b^2n)$  time to generate all new candidates and  $O(b^2n)$  time to insert nonredundant ones into the original list of nonredundant candidates.

In this section, we show that the time complexity of the first operation, adding buffers at a buffer position, can be reduced to  $O(bn)$ , and thus our algorithm can achieve total time complexity  $O(bn^2)$ .

Assume we have computed the set of nonredundant candidates  $N(v_1)$  for  $T(v_1)$ , and now reach a buffer position  $v$ , see Fig. 1. Wire  $(v, v_1)$  has 0 resistance and capacitance. Define  $P_i(\alpha)$  as the slack if we add a buffer type  $B_i$  at  $v$  for any candidate  $\alpha$  in  $N(v_1)$ :

$$P_i(\alpha) = Q(v_1, \alpha) - R(B_i) \cdot C(v_1, \alpha) - K(B_i). \quad (1)$$

If we do not insert any buffer at  $v$ , then every candidate for  $T(v_1)$  is a candidate for  $T(v)$ . If we insert a buffer at  $v$ , then for every buffer type  $B_i$ ,  $i = 1, 2, \dots, b$ , there will be a new candidate  $\beta_i$ :

$$\begin{aligned} Q(v, \beta_i) &= \max_{\alpha \in N(v_1)} \{P_i(\alpha)\}, \\ C(v, \beta_i) &= C(B_i). \end{aligned}$$

Note that some of the new candidates  $\beta_i$ s could be redundant. Define the *best candidate* for  $B_i$  as the candidate  $\alpha_i \in N(v_1)$

such that  $\alpha_i$  maximizes  $P_i(\alpha)$  among all candidates of  $N(v_1)$ . If there are multiple  $\alpha$ 's that maximize  $P_i(\alpha)$ , the one with minimum  $C(\alpha)$  is chosen.

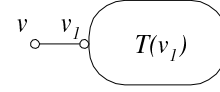


Fig. 1.  $T(v)$  consists of buffer position  $v$  and  $T(v_1)$ .

We show how to generate all  $\beta_i$ s in  $O(bn)$  time. Since all candidates discussed in this section are in  $N(v_1)$ , we will write  $Q(\alpha)$  for  $Q(v_1, \alpha)$ , and  $C(\alpha)$  for  $C(v_1, \alpha)$ . Suppose buffers in the buffer library are sorted according to its driving resistance  $R(B_i)$  in non-increasing order,  $R(B_1) \geq R(B_2) \geq \dots \geq R(B_b)$ . If some buffer types are not allowed at  $v$ , we simply omit them without affecting the rest of the algorithm.

*Lemma 1:* For any two buffer types  $B_i$  and  $B_j$ , where  $i > j$ , let their best candidates be  $\alpha_i$  and  $\alpha_j$ , respectively. Then we must have  $C(\alpha_i) \geq C(\alpha_j)$ .

*Proof:* From the definition of  $\alpha_i$ , we have  $P_i(\alpha_i) \geq P_i(\alpha_j)$  and  $P_j(\alpha_j) \geq P_j(\alpha_i)$ . Consequently,

$$\begin{aligned} Q(\alpha_i) - Q(\alpha_j) &\geq R(B_i) \cdot (C(\alpha_i) - C(\alpha_j)), \\ Q(\alpha_j) - Q(\alpha_i) &\geq R(B_j) \cdot (C(\alpha_j) - C(\alpha_i)). \end{aligned}$$

Therefore,  $(R(B_i) - R(B_j))(C(\alpha_i) - C(\alpha_j)) \leq 0$ .

Since  $i > j$ ,  $R(B_j) \geq R(B_i)$ . If  $R(B_j) > R(B_i)$ ,  $C(\alpha_i) \geq C(\alpha_j)$ . If  $R(B_j) = R(B_i)$ , then it is easy to get  $P_i(\alpha_i) = P_i(\alpha_j)$  and  $P_j(\alpha_j) = P_j(\alpha_i)$ . From the definition, when there are multiple  $\alpha$ 's that maximize  $P_i(\alpha)$ , the one with minimum  $C(\alpha)$  is chosen. Thus  $\alpha_i$  and  $\alpha_j$  should be the same candidate, which means  $C(\alpha_i) = C(\alpha_j)$ . ■

Lemma 1 implies that the best candidates  $\alpha_1, \dots, \alpha_b$  for buffer types  $B_1, \dots, B_b$  are in increasing order of  $C$ . However, this is not enough for an  $O(bn^2)$  time algorithm. In the following, we define the concept of *convex pruning*, which can be used to prune useless candidates that are not pruned by the traditional van Ginneken's algorithm.

**Convex pruning:** Let  $\alpha_1, \alpha_2$  and  $\alpha_3$  be three nonredundant candidates of  $T(v_1)$  such that  $C(\alpha_1) < C(\alpha_2) < C(\alpha_3)$ . If

$$\frac{Q(\alpha_2) - Q(\alpha_1)}{C(\alpha_2) - C(\alpha_1)} < \frac{Q(\alpha_3) - Q(\alpha_2)}{C(\alpha_3) - C(\alpha_2)}, \quad (2)$$

then we prune candidate  $\alpha_2$ .

Convex pruning can be explained by Figure 2. Consider  $Q$  as the  $Y$ -axis and  $C$  as the  $X$ -axis. Then the set of nonredundant candidate  $N(v_1)$  are a set of points in the two-dimensional plane. Candidate  $\alpha_2$  in the above definition is shown in Figure 2(a), and is pruned in Figure 2(b). Call the candidates after convex pruning  $M(v_1)$ . It can be seen that  $N(v_1)$  is a monotonically increasing sequence, while  $M(v_1)$  is a convex hull.

Function `ConvexPruning` performs convex pruning for any list of nonredundant candidates sorted in increasing  $Q$  and  $C$  order. The following C code defines the linked list data structure for the candidates:

```
typedef struct Candidate {
    double Q, C;
```

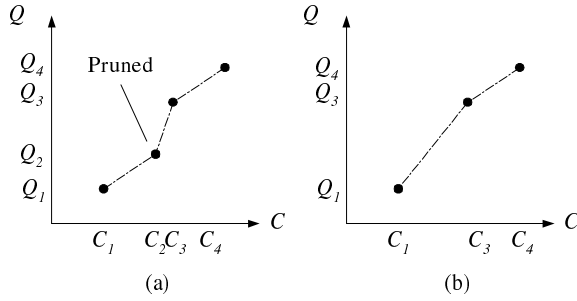


Fig. 2. (a) Nonredundant candidates  $N(v_1)$  on  $(Q, C)$  plane. (b) Nonredundant candidates  $M(v_1)$  after convex pruning.

```

struct Candidate *next, *prev;
    // double link list
} Candidate;

```

Let the candidate with minimum  $C$  be  $\alpha_1$ . We add a dummy candidate  $(-\infty, C(\alpha_1))$  at the beginning of the list to simplify the algorithm. The list is pointed by header. Function `LeftTurn` checks if  $a_1$ ,  $a_2$  and  $a_3$  form a left turn on the plane. It is the same as the condition in Eq. (2).

```

void ConvexPruning(Candidate *header)
{
    Candidate *a1, *a2, *a3;

    a1 = header;
    a2 = a1->next;
    a3 = a2->next;

    while (a3 != NULL) {
        if (LeftTurn(a1, a2, a3)) {
            // prune a2 and move backward
            free(a2);
            a1->next = a3;
            a3->prev = a1;
            a2 = a1;
            a1 = a1->prev;
        } else {
            // move forward
            a3 = a3->next;
            a2 = a2->next;
            a1 = a1->next;
        }
    }
}

```

**Lemma 2:** Given any set of  $k$  nonredundant candidates sorted in increasing  $Q$  and  $C$  order, function `ConvexPruning` performs convex pruning in  $O(k)$  time.

*Proof:* This procedure is known as Graham's scan in computational geometry [13]. It finds the convex hull of a set of points in sorted order in linear time. ■

**Lemma 3:** For any buffer type  $B_i \in \mathbf{B}$ , its best candidate  $\alpha_i$  that maximizes  $P_i(\alpha)$  is not pruned by `ConvexPruning`.

*Proof:* Consider any candidate  $\gamma \in N(v_1)$  with  $C(\gamma) > C(\alpha_i)$ . According to the definition of  $\alpha_i$ , we have  $P_i(\alpha_i) \geq P_i(\gamma)$ . Therefore,

$$\begin{aligned}
 Q(\gamma) - Q(\alpha_i) &\leq R(B_i) \cdot (C(\gamma) - C(\alpha_i)), \\
 \frac{Q(\gamma) - Q(\alpha_i)}{C(\gamma) - C(\alpha_i)} &\leq R(B_i).
 \end{aligned}$$

Similarly for any candidate  $\eta \in N(v_1)$  with  $C(\eta) < C(\alpha_i)$ , we have

$$\begin{aligned}
 Q(\alpha_i) - Q(\eta) &\geq R(B_i) \cdot (C(\alpha_i) - C(\eta)), \\
 \frac{Q(\alpha_i) - Q(\eta)}{C(\alpha_i) - C(\eta)} &\geq R(B_i).
 \end{aligned}$$

Therefore,

$$\frac{Q(\alpha_i) - Q(\eta)}{C(\alpha_i) - C(\eta)} \geq \frac{Q(\gamma) - Q(\alpha_i)}{C(\gamma) - C(\alpha_i)},$$

where  $\eta$  is any candidates with  $C(\eta) < C(\alpha_i)$ , and  $\gamma$  is any candidates with  $C(\gamma) > C(\alpha_i)$ . According to the definition of convex pruning,  $\alpha_i$  is not pruned. ■

**Lemma 4:** Let the set of nonredundant candidates after `ConvexPruning` be  $M(v_1)$  and assume  $M(v_1)$  are sorted in increasing  $Q$  and  $C$  order. Consider any three candidates  $\eta$ ,  $\alpha$ ,  $\gamma$  in  $M(v_1)$ , such that  $C(\eta) < C(\alpha) < C(\gamma)$ . For any buffer type  $B_i \in \mathbf{B}$ , if  $P_i(\eta) \geq P_i(\alpha)$ , then  $P_i(\eta) \geq P_i(\gamma)$ ; if  $P_i(\gamma) \geq P_i(\alpha)$ , then  $P_i(\gamma) \geq P_i(\eta)$ .

*Proof:* From the definition of convex pruning, we have

$$\frac{Q(\gamma) - Q(\alpha)}{C(\gamma) - C(\alpha)} \leq \frac{Q(\alpha) - Q(\eta)}{C(\alpha) - C(\eta)}.$$

If  $P_i(\eta) \geq P_i(\alpha)$ , then

$$\begin{aligned}
 \frac{Q(\alpha) - Q(\eta)}{C(\alpha) - C(\eta)} &\leq R(B_i), \\
 \frac{Q(\gamma) - Q(\alpha)}{C(\gamma) - C(\alpha)} &\leq R(B_i), \\
 Q(\alpha) - R(B_i) \cdot C(\alpha) &\geq Q(\gamma) - R(B_i) \cdot C(\gamma), \\
 P_i(\alpha) &\geq P_i(\gamma).
 \end{aligned}$$

Therefore,  $P_i(\eta) \geq P_i(\gamma)$ . Similarly, if  $P_i(\gamma) \geq P_i(\alpha)$ , then

$$\begin{aligned}
 \frac{Q(\gamma) - Q(\alpha)}{C(\gamma) - C(\alpha)} &\geq R(B_i), \\
 \frac{Q(\alpha) - Q(\eta)}{C(\alpha) - C(\eta)} &\geq R(B_i), \\
 Q(\alpha) - R(B_i) \cdot C(\alpha) &\geq Q(\eta) - R(B_i) \cdot C(\eta) \\
 P_i(\alpha) &\geq P_i(\eta).
 \end{aligned}$$

Therefore,  $P_i(\gamma) \geq P_i(\eta)$ . ■

Lemma 4 implies that for any buffer type  $B_i$ , if candidate  $\alpha$  maximizes  $P_i(\alpha)$  among its previous and next consecutive candidates in  $M(v_1)$ , then  $\alpha$  maximizes  $P_i(\alpha)$  among all candidates in  $M(v_1)$ .

Function `NewCandidate` identifies the best candidates  $\alpha_i$  from  $N(v_1)$  and generates new candidates  $\beta_i$ , for  $i = 1, \dots, b$ . Nonredundant candidates in  $N(v_1)$  are stored in increasing  $C$  order using a double link list pointed by header. Buffer types are sorted in non-increasing driver resistance order and stored in array  $\mathbf{B}$ . Function `P(i, a)` computes  $P_i(\alpha)$  as defined in

Eq. (1). Function `Sort(beta)` sorts  $\beta$ 's in nondecreasing  $C$  order.

```

void Candidate *NewCandidate(
    Candidate *header, Candidate *beta)
{
    Candidate *a1, *a2;
    int i;

    ConvexPruning(header);

    a1 = header;
    a2 = a1->next;
    for (i = 1; i <= b; i++) {
        while (a2 != NULL) {
            if (P(i, a1) < P(i, a2)) {
                a1 = a1->next;
                a2 = a1->next;
            } else
                break;
        }

        // generate new candidate
        beta[i]->Q = P(i, a1);
        beta[i]->C = B[i]->C;
    }

    Sort(beta);
}

```

*Theorem 1:* If  $v$  is a buffer position, wire  $(v, v_1)$  is a wire with zero resistance and capacitance, nonredundant candidates of  $N(v_1)$  are stored in increasing  $Q$  and  $C$  order, then function `NewCandidate` generates all new candidates for  $N(v)$  in  $O(bn)$  time.

*Proof:* Let the set of nonredundant candidates after `ConvexPruning` be  $M(v_1)$ . From Lemma 3, we know that all best candidates  $\alpha_i$ 's are in  $M(v_1)$ . From Lemma 1 and Lemma 4, starting from the first candidates in  $M(v_1)$ , function `NewCandidate` can find all  $\beta_i$ 's in the increasing order of  $i$ .

Now consider the time complexity. According to Lemma 2, function `ConvexPruning` takes  $O(bn)$  time. The `for` loop takes  $O(bn+b) = O(bn)$  time. To reduce the time complexity for function `Sort`, we sort the entire buffer library according to input capacitance  $C(B_i)$  in  $O(b \log b)$  time during preprocessing, and establish an order from buffer index  $i$  to the order in  $C(B_i)$ . Then each time function `Sort` is called, the new candidates  $\beta_i$ 's can be sorted in nondecreasing  $C$  order by using the index in  $O(b)$  time. ■

Once we have all new candidates generated and sorted in increasing  $Q$  and  $C$  order, it is easy to merge with nonredundant candidates in  $N(v_1)$  to produce  $N(v)$ . The time it takes is linear in terms of the two lists:  $O(bn) + O(b) = O(bn)$ . Since the other two operations, adding a wire and merging, can both be done in time  $O(bn)$ , we have:

*Theorem 2:* The optimal buffer insertion problem for  $b$  buffer types and  $n$  possible buffer positions can be computed in time  $O(bn^2)$ .

#### IV. SIMULATION

Both the algorithm of Lillis *et al* [3] and the new algorithm are implemented in C and run on a Sun SPARC workstations with 400 MHz and 2 GB memory. The device and interconnect parameters are based on TSMC 180 nm technology. We have 4 different buffer libraries, of size 8, 16, 32 and 64 respectively. The value of  $R(B_i)$  is from 180  $\Omega$  to 7000  $\Omega$ ,  $C(B_i)$  is from 0.7 fF to 23 fF, and  $K(B_i)$  is from 29 ps to 36.4 ps. The sink capacitances range from 2 fF to 41 fF. The wire resistance is 0.076  $\Omega/\mu m$  and the wire capacitance is 0.118 fF/ $\mu m$ . Table I shows for large industrial circuits, the new algorithm is up to 11 times faster than Lillis' algorithm. The memory usage is only 2% more due to the double linked list used by the new algorithm.

Fig. 3 compares the time complexity of two algorithms for the net with 1944 sinks and 33133 buffer positions with respect to the size of buffer library  $b$ . In the figure, the  $y$  axis is normalized to the running time of the case when the buffer library size is 8. Though the worst case time complexity of Lillis' algorithm is quadratic in terms of  $b$ , it behaves more like a linear function of  $b$ , as observed in [11]. The time complexity of our algorithm is also linear, but has a much smaller slope.

Fig. 4 compares the time complexity of the two algorithms for the net with 1944 sinks, with respect to the number of buffer positions  $n$ . The buffer library size is 32. In the figure, the  $y$  axis is normalized to the running time of the case with 1943 buffer positions. We can see that while Lillis' and our algorithms both behave quadratically, our algorithm shows much slower growing trend since the operation of adding buffers becomes more dominant among three major operations when  $n$  increases.

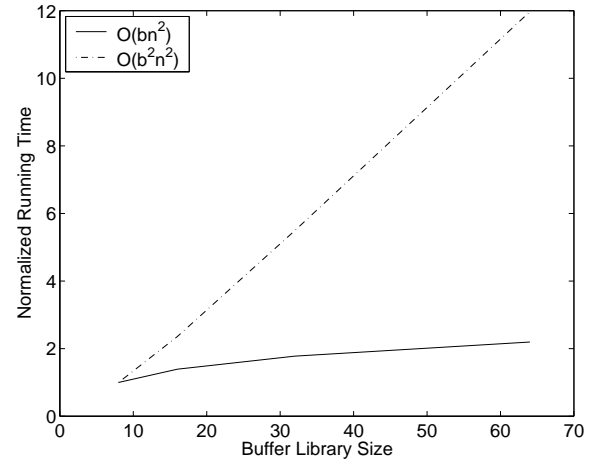


Fig. 3. Comparison of normalized running time of our new  $O(bn^2)$  time algorithm and the  $O(b^2n^2)$  time algorithm by Lillis *et al*. Number of sink is 1944 and number of buffer positions is 33133.

#### V. EXTENSION

The algorithm described in Section III can be extended to improve the buffer cost minimization algorithm by Lillis, Cheng and Lin [3]. They represent each candidate as a tuple  $(Q, C, W)$ , where  $W$  is the total buffer cost, and perform three operations during dynamic programming: 1) adding buffers

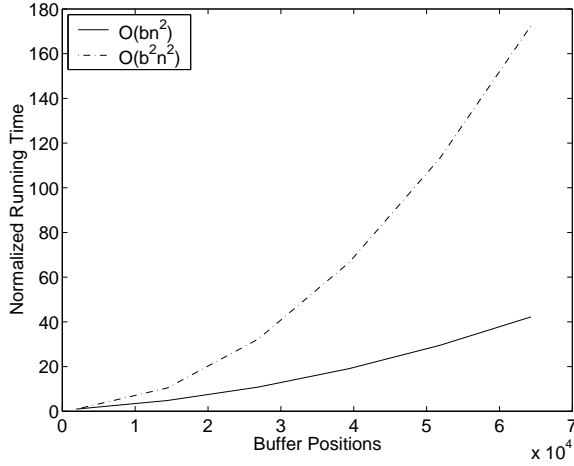


Fig. 4. Comparison of normalized running time of our new  $O(bn^2)$  time algorithm and the  $O(b^2n^2)$  time algorithm by Lillis *et al.* Number of sink is 1944 and number of buffer types is 32.

at a buffer position, 2) adding a wire, and 3) merging two branches. In their algorithm, candidates are first grouped according to  $W$ , and then for each value of  $W$ , stored in increasing order of  $(Q, C)$ . According the analysis in [3], the operation of adding buffers takes  $O(bN)$  time to generate new candidates, where  $N$  is the number of nonredundant candidates.

We extend our algorithm to  $(Q, C, W)$  framework as follows. For each  $W$ , we apply function `NewCandidate` on its list of  $(Q, C)$  candidates. With a similar analysis as in Section III, it is clear that the time to generate new candidates is reduced to  $O(N)$ . The time for other two operations is the same.

Our new algorithm can also be easily integrated with predictive pruning [10], [14], and inverting buffer types [3].

## VI. CONCLUSION

We presented a new algorithm for optimal buffer insertion with  $b$  buffer types of worst case time  $O(bn^2)$ . This is an improvement of the previous best  $O(b^2n^2)$  algorithm [3]. Simulation results show our new algorithm is significantly faster than  $O(b^2n^2)$  algorithms for large industrial circuits with large buffer libraries. Our algorithm can also be applied to resource minimization and inverting buffer types. Since multiple buffer types are used in most existing algorithms on buffer insertion, the new algorithm can significantly improve the performance of all these algorithms.

## REFERENCES

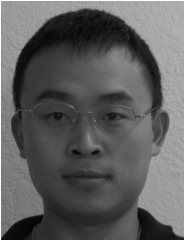
- [1] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "Repeater scaling and its impact on cad," *IEEE Trans. Computer-Aided Design*, vol. 23, no. 4, pp. 451–463, 2004.
- [2] L. P. P. van Ginneken, "Buffer placement in distributed RC-tree network for minimal Elmore delay," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp. 865–868.
- [3] J. Lillis, C. K. Cheng, and T.-T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," *IEEE J. Solid-State Circuits*, vol. 31, no. 3, pp. 437–447, 1996.
- [4] W. Shi and Z. Li, "A fast algorithm for optimal buffer insertion," *IEEE Trans. Computer-Aided Design*, 2005, to be published.

TABLE I

SIMULATION RESULTS FOR INDUSTRIAL TEST CASES, WHERE  $m$  IS THE NUMBER OF SINKS,  $n$  IS THE NUMBER OF BUFFER POSITIONS, AND  $b$  IS THE LIBRARY SIZE.

$m$	$n$	$b$	CPU Time (sec)		Speed-up
			$O(bn^2)$	$O(b^2n^2)$ [3]	
337	336	8	0.08	0.09	1.11
		16	0.14	0.16	1.14
		32	0.23	0.36	1.57
		64	0.42	0.91	2.17
	5647	8	1.54	2.15	1.40
		16	2.11	4.55	2.16
		32	2.81	9.99	3.56
		64	4.05	22.52	5.56
	10957	8	4.56	7.15	1.57
		16	6.02	15.74	2.61
		32	7.62	34.02	4.46
		64	9.98	74.55	7.47
1944	1943	8	0.93	0.90	0.97
		16	1.62	1.86	1.15
		32	2.78	4.38	1.58
		64	4.54	10.71	2.36
	33133	8	22.96	38.19	1.66
		16	31.97	90.08	2.82
		32	40.83	209.82	5.14
		64	50.42	457.22	9.07
	64323	8	70.23	141.07	2.01
		16	95.78	337.97	3.53
		32	117.38	755.46	6.44
		64	136.85	1596.61	11.67
2676	2675	8	1.16	1.13	0.97
		16	2.07	2.38	1.15
		32	3.83	5.78	1.51
		64	6.18	14.15	2.30
	45075	8	27.31	44.29	1.62
		16	36.75	98.31	2.68
		32	47.8	226.25	4.73
		64	64.02	543.45	8.49
	87475	8	82.67	163.87	1.98
		16	108.16	372.22	3.44
		32	134.83	835.04	6.19
		64	164.08	1864.08	11.36

- [5] T. Okamoto and J. Cong, "Buffered steiner tree construction with wire sizing for interconnect layout optimization," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1996, pp. 44–49.
- [6] H. Zhou, D. F. Wong, I. M. Liu, and A. Aziz, "Simultaneous routing and buffer insertion with restrictions on buffer locations," *IEEE Trans. Computer-Aided Design*, vol. 19, no. 7, pp. 819–824, 2000.
- [7] M. Hrkic and J. Lillis, "S-tree: a technique for buffered routing tree synthesis," in *Proc. ACM/IEEE Design Automation Conf.*, 2002, pp. 578–583.
- [8] —, "Buffer tree synthesis with consideration of temporal locality, sink polarity requirements, solution cost and blockages," in *Int. Symp. Physical design*, 2002, pp. 98–103.
- [9] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion for noise and delay optimization," in *Proc. ACM/IEEE Design Automation Conf.*, 1998, pp. 362–367.
- [10] W. Shi, Z. Li, and C. J. Alpert, "Complexity analysis and speedup techniques for optimal buffer insertion with minimum cost," in *Proc. Conf. Asia South Pacific Design Automation*, 2004, pp. 609–614.
- [11] C. J. Alpert, R. G. Gandham, J. L. Neves, and S. T. Quay, "Buffer library selection," in *Proc. IEEE Int. Conf. Computer Design*, 2000, pp. 221–226.
- [12] C. Alpert and A. Devgan, "Wire segmenting for improved buffer insertion," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 588–593.
- [13] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," *Information Processing Letters*, vol. 1, pp. 132–133, 1972.
- [14] W. Shi and Z. Li, "An  $O(n \log n)$  time algorithm for optimal buffer insertion," in *Proc. ACM/IEEE Design Automation Conf.*, 2003, pp. 580–585.



**Zhuo Li** received the B.S. and M.S. degrees in Electrical Engineering from Xi'an Jiaotong University, China, in 1998 and 2001 respectively. He is currently pursuing the Ph.D. degree in Electrical Engineering at Texas A&M University, College Station.

In summer 2004, he worked at IBM Austin Research Laboratory, Austin, TX. His research interests include interconnect optimization, clock network synthesis, timing analysis, delay testing, and design and analysis of VLSI design automation algorithms.



**Weiping Shi** received the B.S. and M.S. degrees in Computer Science from Xi'an Jiaotong University, China, in 1982 and 1984 respectively, and Ph.D. degree in computer science from University of Illinois at Urbana-Champaign in 1992. Currently he is an Associate Professor in the Department of Electrical Engineering, Texas A&M University, College Station.

His research interests include layout synthesis, parasitic extraction, testing, and design and analysis of algorithms.