O(mn) Time Algorithm for Optimal Buffer Insertion of Nets with *m* Sinks

Zhuo Li, *Senior Member, IEEE,* Ying (Nancy) Zhou, *Member, IEEE,* and Weiping Shi, *Senior Member, IEEE*

***Abstract*—Buffer insertion is an effective technique to reduce interconnect delay. In this paper, we give a simple $O(mn)$ time algorithm for optimal buffer insertion, where m is the number of sinks and n is the number of buffer positions. When m is small, our algorithm is a significant improvement over the recent $O(n \log^2 n)$ time algorithm by Shi and Li, and the $O(n^2)$ time algorithm of van Ginneken. For b buffer types, our algorithms runs in $O(b^2n + bmn)$ time, an improvement of the recent $O(bn^2)$ algorithm by Li and Shi. The improvement is made possible by an innovative linked list that can perform addition of a wire, addition of a buffer in amortized $O(1)$ time, and smart design of pointers. We then present the extension of our algorithm for the buffer cost minimization problem, which improves the previous best algorithm. On industrial test cases, the new algorithms is faster than previous best algorithms by an order of magnitude.**

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

I. Introduction

Delay optimization techniques for interconnect are increasingly important for achieving timing closure of high performance designs. One popular technique to reduce interconnect delay is buffer insertion. Saxena *et al.* [1] projected that 35% of all cells would be intrablock repeaters for the 45 nm node. Consequently, algorithms that can efficiently insert buffers are essential for the design automation tools.

This paper studies buffer insertion in interconnect with a set of possible buffer positions and a discrete buffer library. In 1990, van Ginneken [2] proposed an $O(n^2)$ time dynamic programming algorithm for buffer insertion with one buffer type, where n is the number of possible buffer positions. His algorithm finds a buffer insertion solution that maximizes the slack at the source. In 1996, Lillis *et al.* [3] extended van Ginneken's algorithm to allow b buffer types in time $O(b^2n^2)$. In 2003, Shi and Li [4] used a number of techniques to improve the time complexity to $O(b^2n \log n)$ for two-pin nets, and $O(b^2n \log^2 n)$ for multipin nets. To improve the efficiency of the merging operation, Chen and Zhou [5] proposed a data structure based on skip list, while keeping the same complexity. To reduce the quadratic effect of b, Li and Shi [6] proposed an $O(bn^2)$ algorithm. However, all these algorithms do not utilize the fact that most nets have small numbers of pins and large number of buffer positions in real applications.

In this paper, we first propose a new algorithm that performs optimal buffer insertion for two-pin nets in time $O(b^2n)$. The speedup is achieved by an observation that the best candidate to be associated with any buffer must lie on the convex hull of the (Q, C) plane, a clever bookkeeping method and an innovative linked list that allow $O(1)$ time update for adding a wire or a candidate. The new data structure, which is a simple implicit linked list, is much simpler than the candidate tree used in [4] and the skip list used in [5]. We then extend the algorithm to m-pin nets in time $O(b^2n + bmn)$. Experimental results show that our algorithm is faster than previous best algorithms by an order of magnitude. Note that all previous research assumed m and n are of the same order. But in fact, m is often much less than n. Even if $m > n$, we can merge sinks in a branch that contains no buffer position, without changing the problem. Therefore in this paper we assume $m \leq n$.

The van Ginneken [2] algorithm does not control buffer costs. Lillis *et al.* [3] presented an pseudopolynomial time algorithm to control buffer cost. Later, the min-cost buffer insertion problem was proven to be NP-hard in [8] and a polynomial time approximation scheme algorithm was presented in [9]. In this paper, we present how to extend our max-slack algorithm for buffer cost minimization problem on two-pin and multipin nets. Our experimental results show that we can get significant speedup compared to the state-of-the-art optimal algorithm and implementation [8].

Note that the buffer insertion algorithm for the max-slack problem still has its practical value in physical synthesis ecosystem for timing driven placement [10], [11], multiobjective optimizations with buffer-interconnect delay model [13], [14], where the virtual buffering model is used.

In this paper, Elmore delay is used as interconnect delay model due to its high fidelity for the buffer location decision (though the final solution acceptance/rejection can be determined with real timer) and speed [10], [16]. Some adjustments can be used to improve the accuracy [15] without changing the optimality of our algorithms. We also assume a Steiner tree is given, which could be a buffer-aware Steiner tree [12], or directly from global or detail wires. One can insert buffers along the routed wires to minimize the congestion or fixed postrouting timing problems from detour wires.

The preliminary version of this paper is shown in [17], and the main differences are: 1) two new Lemmas added to complete the proof of the algorithm for the multipin nets; 2) the extension to the buffer cost minimization problems; and 3) new implementation of all previous algorithms with state-of-the-art speedup techniques. The algorithms presented in this paper can also be applied to layer assignment algorithm in [7].

II. Preliminary

A net is a tree $\boldsymbol{T} = (V, E)$, where $V = \{v_0\} \cup V_s \cup V_n$, and $E \subset V \times V$. Vertex v_0 is the *source* vertex and also the root of $\boldsymbol{T}$, V_s is the set of *sink* vertexes, 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 $\boldsymbol{B}$ contains b types of buffers. For each buffer type $B_i \in \boldsymbol{B}$, the intrinsic delay is $K(B_i)$, driving resistance is $R(B_i)$, and input capacitance is $C(B_i)$. A function $f : V_n \rightarrow \boldsymbol{2^B}$ specifies the types of buffers allowed at each buffer position.

Manuscript received May 24, 2011; revised August 28, 2011; accepted October 3, 2011. Date of current version February 17, 2012. This work was supported by the National Science Foundation, under Grants CCR-0098329, CCR-0113668, EIA-0223785, and ATP 512-0266-2001. This paper was recommended by Associate Editor P. Saxena.

Z. Li and Y. Zhou are with IBM Austin Research Laboratory, Austin, TX 78758 USA (e-mail: lizhuo@us.ibm.com; nancyz@us.ibm.com).

W. Shi is with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: wshi@ece.tamu.edu).

Digital Object Identifier 10.1109/TCAD.2011.2174639

Each buffer type B_i also has a buffer cost weight $W : \boldsymbol{B} \to [0, +\infty)$. Each edge $e \in E$ is associated with lumped resistance $R(e)$ and capacitance $C(e)$.

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. If v_i is inserted a buffer of type B_k, then the buffer delay is $D(v_i) = R(B_k) \cdot C(v_i) + K(B_k)$, where $C(v_i)$ is the downstream capacitance at v_i, and the capacitance viewed from the upper stream is $C(B_k)$. For any vertex 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* α for $T(v)$. The delay from v to sink $s_i \in T(v)$ under α is $D(v, s_i, \alpha) = \sum_{e=(v_j,v_k)}(D(v_j)+D(e))$, where the sum is over all edges in the path from v to s_i. If v_j is inserted a buffer in α, then $D(v_j)$ is the buffer delay, otherwise, it is zero. The slack of v under α is $Q(v, \alpha) = \min_{s_i \in T(v)}\{RAT(s_i) - D(v, s_i, \alpha)\}$.

Max-Slack Buffer Insertion Problem: Given a net $\boldsymbol{T} = (V, E)$, possible buffer position function f, and buffer library $\boldsymbol{B}$, find a candidate α for $\boldsymbol{T}$ that maximizes $Q(v_0, \alpha)$.

The effect of a candidate α for tree $T(v)$ at v to the upstream is traditionally described by slack $Q(v, \alpha)$ and downstream capacitance $C(v, \alpha)$ [2]. For any two candidates α_1 and α_2 of $T(v)$, we say α_1 *dominates* α_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 $N(v)$.

Min-Cost Buffer Insertion Problem: Given a net $\boldsymbol{T} = (V, E)$, possible buffer position function f, and buffer library $\boldsymbol{B}$, buffer cost function $W : \boldsymbol{B} \to (0, +\infty)$, find a candidate α for $\boldsymbol{T}$ that satisfies $Q(v_0, \alpha) \geq \text{RAT}(v_0)$ and the total buffer cost is minimum.

A pseudopolynomial algorithm was presented in [3] that uses (Q, C, W) to describe each candidate. We will first discuss the max-slack problem.

III. Two-Pin Nets

In this section, we show how to compute optimal buffer insertion for two-pin nets in $O(b^2 n)$ time.

A. Convex Pruning

The concept of convex pruning was first proposed in [6].

Definition 1: Let α_1, α_2 and α_3 be three nonredundant candidates of $T(v)$ such that $C(\alpha_1) < C(\alpha_2) < C(\alpha_3)$ and $Q(\alpha_1) < Q(\alpha_2) < Q(\alpha_3)$. If $\frac{Q(\alpha_2)-Q(\alpha_1)}{C(\alpha_2)-C(\alpha_1)} < \frac{Q(\alpha_3)-Q(\alpha_2)}{C(\alpha_3)-C(\alpha_2)}$, then we call α_2 nonconvex, and prune it.

Lemma 1: For two-pin nets, convex pruning preserves optimality.

This lemma (see [17] for the complete proof) only applies to two-pin nets. For multipin nets nonredundant candidates that are pruned by convex pruning could still be useful for the upstream.

Convex pruning of a list of nonredundant candidates sorted in increasing (Q, C) order can be performed in linear time, and the insertion of a new candidate is $O(1)$ [6].

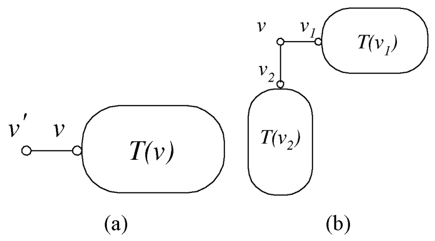

Fig. 1. (a) $T(v')$ consists of buffer position v' and $T(v)$. (b) $T(v)$ consists of $T(v_1)$ and $T(v_2)$.

B. Best Candidates

Assume we have computed the set of nonredundant candidates $N(v)$ for $T(v)$, and now reach a buffer position v', see Fig. 1. Wire (v', v) has 0 resistance and capacitance. Define $P_i(\alpha)$ as the slack if we add a buffer of type B_i at v' for any candidate α: $P_i(\alpha) = Q(v, \alpha) - R(B_i) \cdot C(v, \alpha) - K(B_i)$.

If we do not insert any buffer at v', then every candidate for $T(v)$ is a candidate for $T(v')$. If we insert a buffer at v', then for every buffer type B_i, $i = 1, 2, \ldots, b$, there will be a new candidate β_i: $Q(v', \beta_i) = \max_{\alpha \in N(v)}\{P_i(\alpha)\}$, $C(v', \beta_i) = C(B_i)$. Define the best candidate for B_i as the candidate $\alpha \in N(v)$ such that α maximizes $P_i(\alpha)$ among all candidates in $N(v)$. If there are multiple αs that maximize $P_i(\alpha)$, choose the one with minimum C. From Lemma 1, it is easy to see that all best candidates are on the convex hull. The following lemma says that if we sort candidates in increasing Q and C order from left to right, then as we add wires to the candidates, we always move to the left to find the best candidates.

Lemma 2: For any $T(v)$, let nonredundant candidates after convex pruning be $\alpha_1, \alpha_2, \ldots, \alpha_k$, in increasing Q and C order. Now add wire e to each candidate α_j and denote it as $\alpha_j + e$. For any buffer type B_i, if α_j gives the maximum $P_i(\alpha_j)$ and α_k gives the maximum $P_i(\alpha_k + e)$, then $k \leq j$.

The following lemma says the best candidate can be found by local search, if all candidates are convex.

Lemma 3: For any $T(v)$, let nonredundant candidates after convex pruning be $\alpha_1, \alpha_2, \ldots, \alpha_k$, in increasing Q and C order. If $P_i(\alpha_{j-1}) \leq P_i(\alpha_j)$, $P_i(\alpha_j) \geq P_i(\alpha_{j+1})$, then α_j is the best candidate for buffer type B_i and $P_i(\alpha_1) \leq \cdots \leq P_i(\alpha_{j-1}) \leq P_i(\alpha_j)$, $P_i(\alpha_j) \geq P_i(\alpha_{j+1}) \geq \cdots \geq P_i(\alpha_k)$.
Please see [17] for the complete proof of Lemmas 2 and 3.

C. Data Structure

We store all nonredundant candidates of $T(v)$ in a linked list $L(v)$ of the following data structure:

```
typedef struct Candidate {
    double q, c;
    Candidate *next, *prev;
} Candidate;
```

We also have three global variables: `double Qa`, `Ca`, and `Ra`. $L(v)$ is organized in increasing C and Q order, and pruned by convex pruning. The value of Q and C of each candidate α, pointed by `a`, are given by fields `a->q` and `a->c`, as well as global variables `Qa`, `Ca`, and `Ra`: $Q(\alpha) =$ `(a->q)` $-$ `Qa` $-$ `Ra` $\cdot$ `(a->c)`, $C(\alpha) =$ `(a->c)` $+$ `Ca`.

To facilitate the search for best candidates and the insertion of new candidates, we have two arrays of pointers: `Candidate *best[b], *new[b]`, where `best[i]` points

Algorithm 1 Algorithm Two-Pin

```
Input Routing tree $T(v_1)$ consists of path $v_1, \dots, v_{n+1}$,
    where $v_{n+1}$ is the sink.
Output Nonredundant candidates of $T(v_1)$ stored in
    linked list $L$.
Begin
1: Let Qa=0, Ca=0, Ra=0;
2: Let $L$ contain one candidate $(Q, C)$, where
       $Q = \mathrm{RAT}(v_{n+1})$ and $C = C(v_{n+1})$;
3: Let all best and new pointers point to the only
       candidate in $L$;
4: For $i = n$ to 1 do
5:     AddWire(e), where $e = (v_i, v_{i+1})$;
6:     For each buffer type $B_j$ allowed at $v_i$ do
7:         AddBuffer(j);
8: Return $L$;
End.
```

Algorithm 2 Algorithm M-Pin

```
Input Routing tree $T(v)$ with root $v$.
Output List $A(v)$ that contains all nonredundant
    candidates of $T(v)$.
Begin
1: If $T(v)$ consists of path $v$ to $v_1$, where $v_1$ is a branch
       vertex then
2:     Recursively compute $A(v_1)$ for $T(v_1)$;
3:     $A(v)$ = 2PinSubroutine($A(v_1)$);
4: Else $T(v)$ consists of subtrees $T(v_1)$ and $T(v_2)$,
       where wires $(v, v_1)$ and $(v, v_2)$ have zero resistance
       and capacitance;
5:     Recursively compute $A(v_1)$ and $A(v_2)$;
6:     Merge $A(v_1)$ and $A(v_2)$ to form $A(v)$;
7: Return $A(v)$;
End.
```

to the most recent best candidate for B_i, and `new[i]` points to the most recent new candidate for B_i.

D. Algorithm

When we reach an edge `e` with resistance `e->R` and capacitance `e->C`, we update `Qa`, `Ca`, and `Qa` to reflect the new values of Q and C of all candidate in L in $O(1)$ time, without actually touching any candidate: `Qa = Qa + e->R*e->C/2 + e->R*Ca`, `Ca = Ca + e->C`, `Ra = Ra + e->R`. This is similar to Shi and Li's algorithm [4], but much simpler.

When we reach a buffer position, we may generate a new candidate for each buffer type B_i. But first, we have to find the best candidate for B_i, which is done by pointer `best[i]`. After the best candidate is found, form the new candidate `a`, where `a->c = B[i]->C − Ca`, `a->q = P(i,\,best[i]) + Qa + Ra*a->c`. It is easy to verify that the above transformation of `q` and `c` fields will make the new candidate consistent with every other candidate in $L(v)$. Now insert the new candidate into L. The location to insert new candidates also moves to the left in L, because the capacitances of all candidates increase when wires are added. Finally, we perform convex pruning around the new candidate. The entire algorithm is illustrated in Algorithm 1, where the details of `AddBuffer`, `AddWire`, and the proof of Theorem 1 are in [17].

Theorem 1: Algorithm 1 finds the optimal buffer insertion of any two-pin nets in worst-case time $O(b^2 n)$.

IV. Multipin Nets

We now extend the two-Pin algorithm to multipin nets. In a multipin net, a candidate for a two-pin segment may be merged with a candidate of a different branch, before associated with a buffer. In this case, optimal solution could come from a nonconvex candidate. Therefore, we need all nonredundant candidates of every two-pin segment, not only the convex ones.

This is done by a subroutine `2PinSubroutine(...)` for two-pin segments. The subroutine is similar to Algorithm 1, but in addition to list $L(v)$, maintains a second list $A(v)$. $A(v)$ contains *All* nonredundant candidates of $T(v)$, including nonconvex ones. So $A(v)$ is a superset of $L(v)$. Best candidates are still found through L, yet new candidates are inserted to both L and A. For any two-pin segment $u_1, u_2, \dots, u_k$, the subroutine takes as input $A(u_k)$, prunes nonconvex ones to get $L(u_k)$, find the `best` and `new` pointers from $L(u_k)$, and computes each $L(u_i)$ and $A(u_i)$ as it moves all the way to u_1. Note that the merging process is still performed on $A(v)$ lists for both branches.

To get the best complexity, for multipin nets, we require that buffers are sorted in nonincreasing driving resistance $R(B_1) \geq R(B_2) \geq \cdots \geq R(B_b)$.

The following lemmas show important properties for the locations of new pointers generated after merging two branches, which are the key to the complexity proof of M-Pin algorithm later. Some notations are defined here that will be used for both lemmas. For any $T(v)$ with two subtrees $T(v_1)$ and $T(v_2)$, as shown in Fig. 1, where edges (v, v_1) and (v, v_2) have zero resistance and capacitance, let the list of nonredundant candidates (including nonconvex ones) $\alpha_1, \alpha_2, \dots, \alpha_k$ of $T(v_1)$ be $A(v_1)$, the list of nonredundant candidates $\beta_1, \beta_2, \dots, \beta_l$ of $T(v_2)$ be $A(v_2)$, and the list of nonredundant candidates $\gamma_1, \gamma_2, \dots, \gamma_t$ of $T(v)$ after merging be $A(v)$. All candidates are in increasing Q and C order.

Lemma 4: For buffer type B_i, if the `best[i]` pointer of $A(v_1)$ points to α_o, `best[i]` pointer of $A(v_2)$ points to β_p, and `best[i]` pointer of $A(v)$ points to γ_q, then $q \leq p + o - 1$.

Proof: We first show how the merging process is performed. When two subtrees are merged, for each candidate α_a in $A(v_1)$, we find a candidate β_b in $A(v_2)$ such that $Q(v_1, \beta_b) \geq Q(v_2, \alpha_a)$, and $C(v_1, \beta_b)$ is the minimum among all such β_bs. We then form a new candidate γ in $A(v)$ with $Q(v, \gamma) = \min\{Q(v_1, \alpha_a), Q(v_2, \beta_b)\}$, $C(v, \gamma) = C(v_1, \alpha_a) + C(v_2, \beta_b)$. Do the same for each candidate in $A(v_2)$. Therefore, Q of each new candidate is either decided by $A(v_1)$ or $A(v_2)$.

If there is a candidate γ_h in $A(v)$ that is formed by the best candidate α_o in $A(v_1)$ and another candidate β_g in $A(v_2)$, where $Q(v_1, \alpha_o) \leq Q(v_2, \beta_g)$, $Q(v, \gamma_h) = Q(v_1, \alpha_o)$, $C(v, \gamma_h) = C(v_1, \alpha_o) + C(v_2, \beta_g)$, then for any new candidate γ in $A(v)$ that is formed by a candidate α_a in $A(v_1)$ and a candidate β_b in $A(v_2)$, where $a > o$, $b \geq g$, we show that $P_i(\gamma_h) \geq P_i(\gamma)$ as follows.

Since α_o is the best candidate for buffer type B_i, we have $R(B_i) \geq \frac{Q(v_1,\alpha_a)-Q(v_1,\alpha_o)}{C(v_1,\alpha_a)-C(v_1,\alpha_o)}$. If the Q of γ is decided by α_a, we have

$$
\begin{aligned}
&\frac{Q(v,\gamma) - Q(v,\gamma_h)}{C(v,\gamma) - C(v,\gamma_h)} \\
&= \frac{Q(v_1,\alpha_a) - Q(v_1,\alpha_o)}{C(v_1,\alpha_a) - C(v_1,\alpha_o) + C(v_2,\beta_b) - C(v_2,\beta_g)} \\
&\leq \frac{Q(v_1,\alpha_a) - Q(v_1,\alpha_o)}{C(v_1,\alpha_a) - C(v_1,\alpha_o)} \leq R(B_i).
\end{aligned}
$$

If the Q of γ is decided by β_b, we know that $Q(v_2, \beta_b) \leq Q(v_1, \alpha_a)$, and

$$\frac{Q(v,\gamma) - Q(v,\gamma_h)}{C(v,\gamma) - C(v,\gamma_h)} = \frac{Q(v_2,\beta_b) - Q(v_1,\alpha_o)}{C(v_1,\alpha_a) - C(v_1,\alpha_o) + C(v_2,\beta_b) - C(v_2,\beta_g)}$$
$$\leq \frac{Q(v_1,\alpha_a) - Q(v_1,\alpha_o)}{C(v_1,\alpha_a) - C(v_1,\alpha_o)} \leq R(B_i).$$

So no matter how the Q of γ is decided, we have $\frac{Q(v,\gamma)-Q(v,\gamma_h)}{C(v,\gamma)-C(v,\gamma_h)} \leq R(B_i)$, which means $P_i(\gamma_h) \geq P_i(\gamma)$.

Similarly, it is easy to show that if there is a candidate γ_s in $A(v)$ that is formed by the best candidate β_p in $A(v_2)$ and another candidate α_f in $A(v_1)$, where $Q(v_1, \alpha_f) \geq Q(v_2, \beta_p)$, $Q(v, \gamma_s) = Q(v_2, \beta_p)$, $C(v, \gamma_s) = C(v_1, \alpha_f) + C(v_2, \beta_p)$, then for any new candidate γ in $A(v)$ that is formed by a candidate α_a in $A(v_1)$ and a candidate β_b in $A(v_2)$, where $a \geq f$, $b > p$, we have $P_i(\gamma_s) \geq P_i(\gamma)$.

Therefore, for any candidate γ in $A(v)$ with $Q(\gamma) \geq Q(\gamma_{\min(s,h)})$, we have $P_i(\gamma_{\min(s,h)}) \geq P_i(\gamma)$. So $q \leq \min(s, h) \leq p + o - 1$. ■

From the above lemma, the total number of times that `best[i]` moved in $A(v_1)$, $A(v_2)$, and may move in $A(v)$ in the future is bounded by $q + k - o + l - p \leq k + l - 1$, which is the total number of candidates so far.

Lemma 5: For buffer type B_i, if the `new[i]` pointer of $A(v_1)$ points to α_o, `new[i]` pointer of $A(v_2)$ points to β_p, and `new[i]` pointer of $A(v)$ points to γ_q, then $q \leq p + o - 1$.

Proof: During the merging process, the capacitance of new merged candidate is always bigger than the capacitance of individual candidate from either branches. Therefore, `new[i]` pointer of $A(v)$ will always point to the candidate in $A(v)$ whose Q is decided by α_o or β_p or the candidate at its left side. So we have $q \leq p + o - 1$. ■

From the above lemma, the total number of times that `new[i]` moved in $A(v_1)$, $A(v_2)$, and may move in $A(v)$ in the future is bounded by $q + k - o + l - p \leq k + l - 1$, which is the total number of candidates so far.

Theorem 2: Algorithm 2 computes the optimal buffer insertion of an m-pin net in time $O(b^2n + bmn)$.

Proof: We compute the same set of all nonredundant candidates as previous algorithms, so the algorithm is correct.

At each branch vertex, merging $A(v_1)$ and $A(v_2)$ takes $O(bn)$. Generating L list from A list takes $O(bn)$ by convex pruning. Since buffers are sorted in nonincreasing driving resistance, it takes $O(bn)$ to find all `best` pointers [6]. Since both A and L are sorted in increasing C order, finding all `new` pointers takes $O(bn)$ time. In addition, from Lemmas 4 and 5, for any buffer type B_i, the number of times that `best[i]` pointers and `new[i]` pointers moved in all branches is bounded by the total number of candidates, $O(bn)$. Since there are b buffer types, the total time is bounded by $O(b^2n)$ for all two-pin segments.

Therefore, the time complexity of M-Pin algorithm is $O(b^2n + bmn)$. ■

Our new algorithm can be easily integrated with predictive pruning [4], [8], and inverting buffer types [3].

V. Buffer Cost Minimization

Now we consider the min-cost buffer insertion problem. We assume all costs are integers. For two-pin nets, let integer ω be the maximum possible cost of any candidate, while the minimum nonzero cost is scaled to 1. The algorithm in [3] performs the following operations for two-pin nets: at each buffer position, insert $b \cdot \omega$ new candidates. Since there are n buffer positions, the total number of nonredundant candidates is $O(bn\omega)$. Therefore, the time complexity of their algorithm is $O(b^2n^2\omega)$. In [6], the complexity can be reduced to $O(bn^2\omega)$. In this section, we reduce the time complexity to $O(b^2n\omega)$.

We use the same (Q, C, W) paradigm, where W is the total buffer cost. For each $T(v)$, candidates are stored in ω lists $L_1, L_2, \ldots, L_\omega$. List L_i contains candidates with cost i. In each list, candidates are stored as (Q, C) pairs using implicit representation described above, and pruned through convex pruning. The same global variables are used: `Qa`, `Ca`, and `Ra`. When we reach each wire, we perform the same operation as before in $O(1)$ time. When we reach a buffer position, we perform the same operation for each list L_i; form b new candidates with each buffer B_j and insert the new candidates

TABLE I
Simulation Results for a 2-mm Two-Pin Net

n	b	CPU Time (s)			
		New $O(b^2n)$	Li–Shi [6] $O(bn^2)$	Shi–Li [4] $O(b^2n\log n)$	LCL [3] $O(b^2n^2)$
404	1	0.01	0.02	0.01	0.01
	4	0.02	0.03	0.10	0.03
	8	0.03	0.04	0.35	0.05
	16	0.07	0.05	1.37	0.10
2044	1	0.01	0.59	0.06	0.34
	4	0.06	0.66	0.62	0.66
	8	0.16	0.71	2.14	1.08
	16	0.35	0.81	7.67	2.00
10 404	1	0.09	16.75	0.51	9.48
	4	0.38	18.00	3.68	18.54
	8	0.80	18.41	13.92	29.88
	16	1.79	18.89	50.83	52.47

TABLE II
Simulation Results for Multipin Nets

m	n	b	CPU Time (s)			
			New $O(b^2n + bmn)$	Li–Shi [6] $O(bn^2)$	Shi–Li [4] $O(b^2n\log^2 n)$	LCL [3] $O(b^2n^2)$
25	1337	1	0.02	0.06	0.04	0.04
		4	0.07	0.18	0.44	0.17
		8	0.14	0.30	1.75	0.43
		16	0.30	0.46	7.38	1.06
	2567	1	0.04	0.22	0.07	0.13
		4	0.14	0.60	0.91	0.57
		8	0.29	0.99	3.66	1.54
		16	0.58	1.45	15.31	3.76
	12 407	1	0.19	4.77	0.44	2.58
		4	0.71	12.86	5.22	13.41
		8	1.41	21.19	21.82	38.39
		16	2.92	29.99	88.33	97.83
337	5647	1	0.12	0.17	0.15	0.11
		4	0.37	0.47	1.77	0.42
		8	0.68	0.76	7.15	0.94
		16	1.29	1.15	27.67	2.06
	10 957	1	0.20	0.50	0.30	0.31
		4	0.70	1.30	3.57	1.25
		8	1.29	2.09	14.4	2.82
		16	2.47	3.04	56.58	6.36
	53 437	1	1.03	9.17	1.74	5.08
		4	3.56	23.82	20.5	24.08
		8	6.87	36.99	83.09	64.94
		16	13.10	50.07	332.26	162.85

TABLE III
Simulation Results for Min-Cost Buffer Insertion, Where n Is the Multiplying Factor of the Number of Candidate Locations

n	Library	CPU Time (s)		Speedup
		LCL with Predictive Pruning [3], [8]	New	
1X	Small(8)	19.46	21.93	0.89
	Large(14)	44.86	39.52	1.14
	All(24)	86.76	72.54	1.20
5X	Small(8)	346.81	331.88	1.04
	Large(14)	614.67	446.61	1.38
	All(24)	1109.36	671.61	1.65
8X	Small(8)	736.54	678.38	1.09
	Large(14)	1202.18	851.67	1.41
	All(24)	2304.04	1206.82	1.91

TABLE IV
SIMULATION RESULTS FOR MAX-SLACK BUFFER INSERTION

n	Library	CPU Time (s)			
		New $O(b^2n + bmn)$	Li–Shi [6] $O(bn^2)$	Shi–Li [4] $O(b^2n \log^2 n)$	LCL [3] $O(b^2n^2)$
8X	Small(8)	1.32	1.81	17.38	2.12
	Large(14)	2.38	2.24	40.33	3.55
	All(24)	4.77	2.93	110.45	5.98
50X	Small(8)	7.37	71.45	136.36	121.65
	Large(14)	13.45	75.29	321.32	196.67
	All(24)	25.86	83.82	904.49	341.09
80X	Small(8)	11.89	186.27	238.79	327.11
	Large(14)	21.76	196.11	564.97	535.12
	All(24)	41.71	212.77	1585.86	918.61

into list $L_{i+W(B_j)}$. We do not perform pruning across different lists. This gives the total time as claimed.

For multisink nets, the problem is NP-hard [8]. Our algorithm can improve the practical runtime when we reach each wire or a buffer position, though the time complexity is still dominated by the merging part.

VI. SIMULATION

The first set of experiments are for max-slack buffer insertion problem. All algorithms ($O(b^2n^2)$ [3], $O(b^2nlog^2n)$ [4], $O(bn^2)$ [6], and the new algorithm) are implemented in C and run on a Sun SPARC workstations with 400 MHz clock and 2 GB memory. For all algorithms, the memory management is done in the way similar to [4], which means for the two-pin net, all algorithms take $O(bn)$ memory, and for the net with m sinks, except for $O(b^2nlog^2n)$ algorithm, all other algorithms take $O(bmn)$ instead of $O(bn^2)$ memory. Therefore, the implementation of $O(b^2n^2)$ in this paper produce much better running time than the one in [4]. Also, predictive pruning [4], [8] has been implemented in all algorithms to get the fastest running time for each algorithm. This is the first experiment to compare all optimal buffering algorithms with the state-of-the-art of the implementation.

Table I shows for a 2-mm long two-pin net with different possible buffer insertion locations, the new algorithm is up to 20 times faster than previous best algorithms. Table II shows for large industrial multipin nets where m is as high as 337, the new algorithm is still faster than previous best algorithms, and the speedup could be 16 times even for 16 types of buffers. All algorithms generate same slacks. The device and interconnect parameters for these two experiments are based on TSMC 180 nm technology and can be found in [17].

The second set of experiments are for min-cost buffer insertion problem. We test our new algorithm on nets extracted from an industrial ASIC chip with 300k+ gates [8]. The gates have been placed and buffers are required to optimize timing. This group consists 429 nets with two to five pins among 1000 most time consuming nets. Each net has tens to hundreds of buffer positions with different metal layers and vias. The buffer library consists of 24 buffers, in which 8 are noninverting buffers and 16 are inverting buffers. The range of driving resistance is from 120 Ω to 945 Ω, and the input capacitance is from 6.27 fF to 121.56 fF. In this case, our new algorithm could get to 2X faster than previous best optimal algorithm [8]. The result is shown in Table III. Note that when the number of buffer positions is small and the buffer library size is small, the new algorithm could be a little slower due to its data structure overhead.

For the same 1000 nets, another experiment is done to compare algorithms $O(b^2n^2)$ [3], $O(b^2nlog^2n)$ [4], $O(bn^2)$ [6], and the new algorithm for the max-slack buffer insertion problem. The result is shown in Table IV. This set of experiment is run on a Red Hat Linux machine with 2.93 GHz clock and 128 GB memory, because the original machine for the other experiments is not available to the authors when this experiment is performed. All algorithms for the max-slack buffer insertion problem run much faster than those for min-cost buffer insertion problem after CPU scaling. We also choose 8x, 50x, and 80x as multiplying factors of the number of candidate locations to show the meaningful runtime comparison of different algorithms. The new algorithm is up to 17 times faster than previous best algorithms.

The source codes of the $O(mn)$ algorithm and all previous fast buffer insertion algorithms that are presented in this paper have been released to the public at [18].

REFERENCES

[1] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "Repeater scaling and its impact on CAD," *IEEE Trans. Comput.-Aided Des.*, vol. 23, no. 4, pp. 451–463, Apr. 2004.

[2] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree network for minimal Elmore delay," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 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, Mar. 1996.

[4] W. Shi and Z. Li, "A fast algorithm for optimal buffer insertion," *IEEE Trans. Comput.-Aided Des.*, vol. 24, no. 6, pp. 879–891, Jun. 2005.

[5] R. Chen and H. Zhou, "A flexible data structure for efficient buffer insertion," in *Proc. IEEE Int. Conf. Comput. Des.*, Oct. 2004, pp. 216–221.

[6] Z. Li and W. Shi, "An $O(bn^2)$ time algorithm for optimal buffer insertion with b buffer types," *IEEE Trans. Comput.-Aided Des.*, vol. 25, no. 3, pp. 484–489, Mar. 2006.

[7] Z. Li, C. J. Alpert, S. Hu, T. Muhmud, S. T. Quay, and P. Villarrubia, "Fast interconnect synthesis with layer assignment," in *Proc. Int. Symp. Phys. Des.*, 2008, pp. 71–77.

[8] 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 Des. Automat.*, 2004, pp. 609–614.

[9] S. Hu, Z. Li, and C. J. Alpert, "A fully polynomial time approximation scheme for timing driven minimum cost buffer insertion," in *Proc. ACM/IEEE Des. Automat. Conf.*, Jul. 2009, pp. 424–429.

[10] C. J. Alpert, S. Karandikar, Z. Li, G.-J. Nam, S. T. Quay, H. Ren, C. N. Sze, P. G. Villarrubia, and M. Yildiz, "Techniques for fast physical synthesis," *Proc. IEEE*, vol. 95, no. 3, pp. 573–599, Mar. 2007.

[11] N. Viswanathan, G.-J. Nam, J. A. Roy, Z. Li, C. J. Alpert, S. Ramji, and C. C. N. Chu, "ITOP: Integrating timing optimization within placement," in *Proc. ISPD*, 2010, pp. 83–90.

[12] C. J. Alpert, M. Hrkic, J. Hu, and S. T. Quay, "Fast and flexible buffer trees that navigate the physical layout environment," in *Proc. DAC*, 2004, pp. 24–29.

[13] D. A. Papa, T. Luo, M. D. Moffitt, C. N. Sze, Z. Li, G.-J. Nam, C. J. Alpert, and I. L. Markov, "RUMBLE: An incremental, timing driven, physical-synthesis optimization algorithm," in *Proc. ISPD*, Apr. 2008, pp. 2–9.

[14] Z. Li, D. A. Papa, C. J. Alpert, S. Hu, W. Shi, C. C. N. Sze, Y. Zhou, "Ultra-fast interconnect driven cell cloning for minimizing critical path delay," in *Proc. ISPD*, 2010, pp. 75–82.

[15] A. I. Abou-Seido, B. Nowak, and C. Chu, "Fitted Elmore delay: A simple and accurate interconnect delay model," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, no. 7, pp. 691–696, Jul. 2004.

[16] Y. Zhou, C. J. Alpert, Z. Li, C. N. Sze, and L. Trevillyan, "Shedding physical synthesis area bloat," *Very Large Scale Integr. Des.*, vol. 2011, no. 503025, pp. 1–10, 2011 [Online]. Available: http://www.hindawi.com/journals/vlsi/2011/503025/cta

[17] Z. Li and W. Shi, "An $O(mn)$ time algorithm for optimal buffer insertion of nets with m sinks," in *Proc. Conf. Asia South Pacific Des. Automat.*, 2006, pp. 320–325.

[18] *FBI: Fast Buffer Insertion for Interconnect Optimization* [Online]. Available: http://dropzone.tamu.edu/~zhuoli/GSRC/fast_buffer_insertion.html