

Complexity Analysis and Speedup Techniques for Optimal Buffer Insertion with Minimum Cost*

Weiping Shi

Dept. of Electrical Engineering
Texas A&M University
College Station, TX 77843
wshi@ee.tamu.edu

Zhuo Li

Dept. of Electrical Engineering
Texas A&M University
College Station, TX 75203
zhuoli@ee.tamu.edu

Charles J. Alpert

IBM Austin Research Lab.
11501 Burnet Road
Austin, TX 78758
alpert@us.ibm.com

Abstract- As gate delays decrease faster than wire delays for each technology generation, buffer insertion becomes a popular method to reduce the interconnect delay. Several modern buffer insertion algorithms (e.g., [7, 6, 15]) are based on van Ginneken's dynamic programming paradigm [14]. However, van Ginneken's original algorithm does not control buffering resources and tends to over-buffering, thereby wasting area and power. It has been a major open problem whether it is possible to optimize slack and at the same time minimize the buffer usage.

This paper settles this open problem by showing that for arbitrary integer cost functions, the problem is NP-complete. We also extend the pre-buffer slack technique [12] to minimize the buffer cost. This technique can significantly reduce the running time and memory in buffer cost minimization problem. The experimental results show that our algorithm can speed up the running time up to 17 times and reduces the memory to 1/30 of traditional best know algorithm. Finally, we show how to efficiently deal with multiway merge in buffer insertion.

I. INTRODUCTION

As feature sizes continue to shrink, buffering nets become increasingly critical and problematic. A recent study by Saxena *et al.* [11] projects that 35% of all cells will be intra-block repeaters for the 45 nm node. The sheer explosion in the number of repeaters makes it paramount to conserve buffering resources as much as possible during physical synthesis.

In 1990, van Ginneken [14] proposed a buffer insertion algorithm that is now considered a classic in the field. Several works have built upon this algorithm to include wire sizing [8], higher-order delay models [3, 4], simultaneous tree construction [9, 10], multiple buffer types [8], and noise constraints [2]. Given a fixed Steiner topology, the algorithm inserts buffers in a bottom-up manner to optimize the worst slack to any sink under the Elmore delay model. The algorithm has time complexity $O(n^2)$, where n is the number of potential insertion points. The algorithm does not control buffering resources and will insert as many buffers as needed to obtain the optimal slack. In practice, this results in a significant over buffering whereby a few picoseconds of performance may be squeezed out for several additional buffers. Also, one frequently wants to find the cheapest solution that meets the timing target, not

necessarily the optimal solution in terms of minimal delay. In fact, van Ginneken [14] recognized this, writing, "In addition to the optimization of the timing, the number of buffers used can be optimized. This is done by using triples of numbers rather than pairs for the options. ... Unfortunately, this makes the algorithm no longer polynomial." The pairs he referred to are solutions storing capacitance and slack. Lillis *et al.* [8] presented an implementation that adds a third element to control resource utilization, but were unable to claim a polynomial algorithm.

Recently, Shi and Li [12] showed how the dynamic programming approach used by van Ginneken's can be made to run in $O(n \log n)$ time by four speedup techniques. Among those techniques, we find that the pre-buffer slack pruning technique alone can be incorporated in many buffer insertion algorithms and both timing and space performance can be improved.

In this work, we prove that for arbitrary buffer cost functions, the problem of minimizing buffering resources subject to timing constraints is NP-complete. In addition, we show how to apply the pre-buffer slack pruning technique to minimize the buffer cost. Experiment results show that this technique can significantly speed up the running time and reduce the memory usage.

Finally, we address the problem of merging branches when the number of child nodes is more than two. For these cases, previous work suggests converting the tree into a binary tree by using zero length wires. However, the mechanism for conversion can potentially yield different results. We show that one can explore the entire solution space and still maintain a polynomial algorithm as long as the maximum degree of a node is bounded by a constant

The rest of the paper is organized as follows. Section II presents notation and defines the buffer insertion problem. Section III is the NP-completeness proof. Section IV overviews the extension of van Ginneken's algorithm to control resource usage. Section V presents the pre-buffer slack pruning technique and experimental results showing its effectiveness. Section VI shows how we deal with merging problems.

II. PRELIMINARIES

A net is given as a routing tree $\mathbf{T} = (V, E)$, where $V = \{s_0\} \cup V_s \cup V_n$, and $E \subseteq V \times V$. Vertex s_0 is the *source* vertex

*Research of W. Shi and Z. Li was supported in part by NSF grants CCR-0098329, CCR-0113668, EIA-0223785, ATP grant 512-0266-2001 and a fellowship from Applied Materials.

and also the root of \mathbf{T} , V_s is the set of *sink* vertices, and V_n is the set of *internal* vertices. Each sink vertex $s_i \in V_s$ is associated with sink capacitance $C(s_i)$ and required arrival time $RAT(s_i)$. A target required arrival time for source $RAT(s_0)$ is also given. A buffer library B contains different types of buffers. For each buffer type b_i , 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 buffer type b_i also has a buffer cost weight $W : B \rightarrow [0, \infty)$. Each edge $e \in E$ is associated with lumped resistance $R(e)$ and capacitance $C(e)$.

Following previous researchers [1, 14, 15], 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 $D(e) = R(e)(C(v_j) + C(e)/2)$, where $C(v_j)$ is the downstream capacitance at v_j . For any buffer b at vertex v_j , the buffer delay is $D(v_j) = K(b) + R(b) \cdot C(v_j)$, where $C(v_j)$ is the downstream capacitance at v_j . When a buffer b is inserted, the capacitance viewed from the upper stream is $C(b)$.

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* α for $T(v)$. The delay from v to sink $s_i \in T(v)$ under α is

$$D(v, s_i, \alpha) = \sum_{e=(v_j, v_k)} (D(v_j) + D(e)), \quad (1)$$

where the sum is over all edges in the path from v to s_i . If v_j is a buffer in α , then $D(v_j)$ is the buffer delay. If v_j is not a buffer in α , then $D(v_j) = 0$. The slack of v under α is

$$Q(v, \alpha) = \min_{s_i \in T(v)} \{RAT(s_i) - D(v, s_i, \alpha)\}. \quad (2)$$

The buffer cost of α is the total cost of buffers used in α :

$$W(v, \alpha) = \sum_{b_i \in \alpha} W(b_i). \quad (3)$$

Buffer Insertion Problem: Given routing tree $\mathbf{T} = (V, E)$, sink capacitance $C(s_i)$ and $RAT(s_i)$ for each sink s_i , capacitance $C(e)$ and resistance $R(e)$ for each edge e , buffer library B , possible buffer position f , and buffer cost function W , find a candidate α for \mathbf{T} that satisfies $Q(s_0, \alpha) \geq RAT(s_0)$ and the total buffer cost $W(s_0, \alpha)$ is minimum.

III. COMPLEXITY ANALYSIS

Theorem 1 *If the cost of each buffer is an arbitrary integer, then to minimize the total buffer cost is NP-complete.*

Proof: The problem is clearly in NP. We now show a reduction from 2-1 partition, a known NP-complete problem [5]:

Instance: Positive integers x_1, x_2, \dots, x_{2n} . Let $\sum_{i=1}^{2n} x_i = 2N$

Question: Is there an index set I that contains exactly one of $2i-1$ and $2i$ for $1 \leq i \leq n$, such that $\sum_{i \in I} x_i = N$?

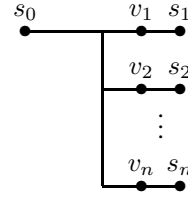


Fig. 1. Construction used for reduction, where v_1, \dots, v_n are buffer positions and s_1, \dots, s_n are sinks.

TABLE I
CONSTRUCTION OF SINKS.

| Sink s_i | $C(s_i)$ | $Q(s_i)$ |
|------------|-----------|---------------------|
| s_1 | N^{n+2} | $N^{n+1} + N^{n+2}$ |
| s_2 | N^{n+1} | $N^{n+1} + N^{n+2}$ |
| \vdots | \vdots | \vdots |
| s_n | N^3 | $N^{n+1} + N^{n+2}$ |

Given an instance of the 2-1 partition problem, we construct an instance of the buffer insertion problem as shown in Fig. 1. There are n sinks and $2n$ buffer types as shown in Tables I and II. For source s_0 , the buffer driver resistance $R(s_0) = N^n$. All wires have zero resistance and capacitance. Clearly, every number in the construction can be expressed in $O(n \log N)$ bits.

Now we claim there is a solution for the buffer insertion instance with $Q(s_0) \geq 0$ and total buffer cost at most

$$M = N + \sum_{i=1}^n N^i$$

if and only if there is a solution for the 2-1 partition instance.

First assume there is a solution for the buffer insertion problem. It is easy to see that there must be a buffer at every v_i , since otherwise $R(s_0) \cdot C(s_i) \geq N^{n+3} > Q(s_i)$ for any i . Furthermore, v_1 must use either buffer type b_1 or b_2 , since otherwise $R(b_i) \cdot C(s_1) \geq N^{n+3} > Q(s_0)$. Since v_1 must use either b_1 or b_2 , v_2 can not use b_1 or b_2 anymore, since otherwise the total buffer cost will be at least $2N^n > M$. Repeat the argument for every i , we know the buffer types for v_i can only be b_{2i-1} or b_{2i} . This way, the delay caused by buffers at v_i is N^{n+2} , for all $i = 1, 2, \dots, n$.

TABLE II
CONSTRUCTION OF BUFFERS.

| Buffer b_i | $R(b_i)$ | $C(b_i)$ | $W(b_i)$ |
|--------------|-----------|------------|-----------------|
| b_1 | 1 | x_1 | $x_2 + N^n$ |
| b_2 | 1 | x_2 | $x_1 + N^n$ |
| b_3 | N | x_3 | $x_4 + N^{n-1}$ |
| b_4 | N | x_4 | $x_3 + N^{n-1}$ |
| \vdots | \vdots | \vdots | \vdots |
| b_{2n-1} | N^{n-1} | x_{2n-1} | $x_{2n} + N$ |
| b_{2n} | N^{n-1} | x_{2n} | $x_{2n-1} + N$ |

Let I be the set of buffer indices that are inserted at v_1, \dots, v_n . Then

$$\begin{aligned} Q(s_0) &= \min_{1 \leq i \leq n} \{Q(s_i) - N^{n+2}\} - R(b_0) \cdot \sum_{i \in I} C(b_i) \\ &= N^{n+1} - N^n \sum_{i \in I} x_i, \\ W(s_0) &= \sum_{i \notin I} x_i + \sum_{i=1}^n N^i. \end{aligned}$$

Since we have both $Q(s_0) \geq 0$ and $W(s_0) \leq M$, we must have $Q(s_0) = 0$ and $W(s_0) = M$, which is a solution to the 2-1 partition instance.

On the other hand, any solution to the 2-1 partition instance, we can assign buffers according to the partition, and prove the solution satisfy the requirements. \square

Finally, we say a few words about the difference between NP-complete and NP-hard [5]. Some literatures use NP-complete to describe a decision problem, and NP-hard to describe an optimization problem. This difference is rather technical. However, there is a fundamental difference that should be emphasized: The NP-hard class includes problems that are NP-complete, PSPACE-complete, EXTIME-complete, etc, all the way to undecidable [5]. Therefore, by saying a problem is NP-complete, it also puts an upper bound on the complexity.

IV. (Q, C, W) FRAMEWORK

In van Ginneken's original algorithm, the effect of a candidate to the upstream is described by the (Q, C) pair, where Q is the slack at the current tree node and C is the downstream capacitance. To constrain total resource usage, van Ginneken [14] suggested to add a total cost W to the tuple i.e., (Q, C, W) , which is implemented by Lillis *et al.* [8].

For any two candidates α_1 and α_2 of $T(v)$, we say α_1 dominates α_2 , if $Q(v, \alpha_1) \geq Q(v, \alpha_2)$, $C(v, \alpha_1) \leq C(v, \alpha_2)$ and $W(v, \alpha_1) \leq W(v, \alpha_2)$. 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 any candidate of $T(v)$ is dominated by some candidates in $N(v)$.

Lillis' algorithm generates candidates in a bottom up manner starting from the sinks. Initially, each sink s_i has a single candidate α , whereby $Q(s_i, \alpha) = RAT(s_i)$, $C(s_i, \alpha) = C(s_i)$, and $W(s_i, \alpha) = 0$. There are three basic operations during the bottom-up traversal as shown in Fig. 2.

1. **Add a wire.** As one propagates candidates from node v_1 up to its parent node v , one must incorporate the delay of wire (v, v_1) into each candidate of v_1 . The number of candidates does not increase (and may even decrease due to pruning).
2. **Add a buffer.** At a node v , one may potentially consider adding buffers to some subset of candidates at v_1 . The number of candidates will increase, but is bounded by the number of different values of W that can possibly be generated.

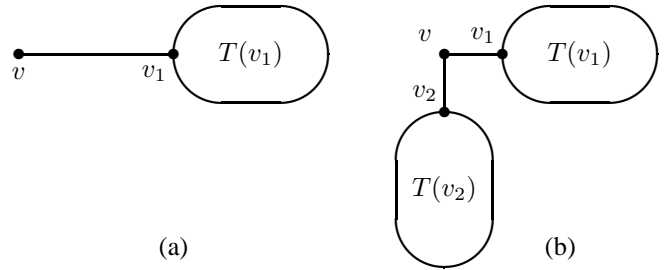


Fig. 2. (a) $T(v)$ consists of edge (v, v_1) and $T(v_1)$. (b) $T(v)$ consists of $T(v_1)$ and $T(v_2)$.

3. **Merge two sub-trees.** For now, assume that every Steiner tree can be transformed into an equivalent binary tree by adding zero length wires. The merging of sub-trees $T(v_1)$ and $T(v_2)$ when controlling resources is the most expensive of the three operations. We discuss this process in further detail below.

After performing these operations as required, eventually the set of candidates $N(s_0)$ at the source is identified. One can then explore the candidates in this set to find the solution corresponding to the desired cost/slack tradeoff.

The third step of merging two branches can be problematic because the number of candidate solutions can potentially explode. Consider the example in Fig. 3. Here we assume that the cost function is the simple (yet practical) number of buffers, i.e., $W(b_i) = 1$, for all $b_i \in B$. This type of function is useful if one is using just one buffer type or if one is at a stage in the design where area is not as significant as the designer effort to make an ECO (Engineering Change Order) change. Also, it is certainly a reasonable choice if all the buffers in the given library are fairly close in size.

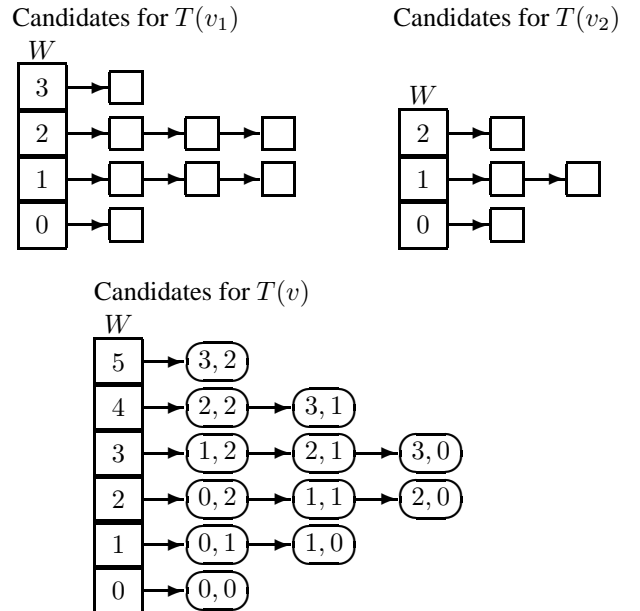


Fig. 3. Example of the algorithm for merging left and right candidates to obtain a single set of candidates for the branching point. Here, the cost function is the number of buffers inserted.

The set of candidates is stored as an array, indexed by the cost W . New candidates are generated by exploring potential merges so that the new candidates are generated in nondecreasing order of cost. For example, first the zero-buffer left candidates are merged with the zero-buffer right candidates. Then the zero-buffer left candidates are merged with the one-buffer right candidates followed by the one-buffer left candidates and the zero-buffer right candidates.

When each new candidate is generated, its capacitance and slack can be inserted into a range-query tree [8] to allow for pruning based on just Q and C . The trick is that by visiting all new candidates in nondecreasing order of cost, it is guaranteed that each new candidate added to the range-query tree will be dominated in terms of cost by the other candidates already in the tree. Then one only needs to determine additional dominance in Q and C to see whether the candidate should be rejected. This test can be done in time logarithmic in the size of the tree.

As one can see from Fig. 3, the number of candidates can potentially explode. Let n_1 and n_2 be the number of candidates in the sub-trees $T(v_1)$ and $T(v_2)$, then there can be $n_1 \cdot n_2$ possible candidates, this leads to a possibly exponential algorithm.

V. PRE-BUFFER SLACK PRUNING

A. General Idea

The concept of pre-buffer slack is first proposed by Shi and Li [12]. The main idea is explained as follows.

For now assume there is only one buffer type b . When we compare candidates at v , it is insufficient to only compare Q and C values at v . Instead, we want a candidate α that maximizes pre-buffer slack

$$P(v, \alpha) = Q(v, \alpha) - K(b) - R(b) \cdot C(v, \alpha), \quad (4)$$

among all candidates. However, such a candidate is not necessarily the candidate that maximizes Q . It is because when a buffer is attached, some nonredundant candidates might become redundant.

For any candidates α_1 and α_2 of $T(v)$, we say α_1 *b-dominates* α_2 if $P(v, \alpha_1) \geq P(v, \alpha_2)$ and $C(v, \alpha_1) \leq C(v, \alpha_2)$. In [12], there are two important lemmas.

Lemma 1 *If α_1 b-dominates α_2 , then α_2 is redundant.*

Lemma 2 *If α_1 and α_2 do not b-dominate one another, then $P(v, \alpha_1) > P(v, \alpha_2)$ if and only if $Q(v, \alpha_1) > Q(v, \alpha_2)$.*

We can expand the above concept and lemmas considering the buffer cost. For any candidates α_1 and α_2 of $T(v)$, we say α_1 *b-dominates* α_2 if $P(v, \alpha_1) \geq P(v, \alpha_2)$, $C(v, \alpha_1) \leq C(v, \alpha_2)$ and $W(v, \alpha_1) \leq W(v, \alpha_2)$.

Lemma 3 *If α_1 b-dominates α_2 , then α_2 is redundant.*

Proof: From Lemma 1, we know that α_2 is redundant in terms of (Q, C) . Since $W(v, \alpha_1) \leq W(v, \alpha_2)$, and for any cases of adding a buffer, adding wire delay or merging, same cost will be added to both α_1 and α_2 , then α_2 is redundant. \square

It is easy to see if α_1 dominates α_2 , then α_1 *b-dominates* α_2 , and α_2 will be pruned. Since the pre-buffer slack is used to determine the dominance, we call this pruning technique as pre-buffer slack pruning. It not only gives a better pruning criteria, but also allows us to find the candidate that gives the maximum P in $O(1)$ time.

Most traditional buffer insertion algorithms based on (Q, C) can be improved using the new pruning technique based on (P, C) . Also, this method does not change the data structure or the frame of previous algorithms. In the application, the value P does not need to be stored. It can be computed from Q and will only be used when the redundancy is checked.

For multiple buffer types, the pre-buffer slack is defined for each type of buffer b_i : $P_i(v, \alpha) = Q(v, \alpha) - K(b_i) - R(b_i) \cdot C(v, \alpha)$. In other words, $P_i(v, \alpha)$ is the slack before an imaginary buffer of type b_i at v . For any two candidates α_1 and α_2 of $T(v)$, we say α_1 *b_i-dominates* α_2 if $P_i(v, \alpha_1) \geq P_i(v, \alpha_2)$, $C(v, \alpha_1) \leq C(v, \alpha_2)$, and $W(v, \alpha_1) \leq W(v, \alpha_2)$.

B. Application to Buffer Cost Minimization

For buffer insertion with minimum cost, we use the framework of Lillis *et al.* [8] and apply the pre-buffer slack pruning technique to both solution set indexed by cost and the range-query tree (if we perform on only one of them, the optimal solution can still be achieved except that few redundant solutions will be pruned). Since all new candidates are visited in nondecreasing order of cost, the pre-buffer slack pruning technique guarantees the optimality.

Multiple buffer types are also considered here. However, this will increase the space complexity by a factor of $|B|$ without. We can avoid the extra space increase by only pruning those candidates b_k -dominated by other candidates, where b_k is the buffer with the smallest $R(b_i)$ among all buffers. There will be less redundant solutions pruned compared with using $|B|$ trees, but our experiments show that there are still many redundant solutions been pruned, compared with previously Q pruning technique.

C. Experimental Results

To show the advantage of new pruning technique, we tested our new algorithm for buffer insertion with cost constraints. Six different buffer types based on TSMC 180nm technology are used. For the smallest buffer(1X), $R(b) = 1440\Omega$, $C(b) = 2.9$ fF, and $K(b) = 36.4$ ps. The largest buffer is 8X. The sink capacitances range from 2 fF to 41 fF. The wire resistance is $0.076 \Omega/\mu\text{m}$ and the wire capacitance is 0.118 fF/ μm . Intrinsic gate delay is identical for all buffers. All algorithms are implemented in C and run on a Sun SPARC workstations with 400 MHz and 2 GB memory. The implemented algorithms also output buffer positions.

The first experiment is to compare our new technique with Lillis' original algorithm [8] for buffer cost minimization. The buffer cost is the number of buffers. In Table III, the time and memory results are shown for six buffer types and the number of buffer positions are the number of sinks. Our algorithm is 2 to 17 times faster than Lillis' algorithm and uses 1/1.5 to 1/30 of memory. The performance of our algorithm is better when the size of buffer library is large. Since generally the

TABLE III
SIMULATION RESULTS FOR SIX BUFFER TYPES WITH BUFFER COST CONSTRAINTS.

| Sinks m | Buffer positions n | Buffer cost W | CPU Time (sec) | | Speedup | Memory (MB) | | Reduction |
|--------------|----------------------------|-----------------------|-----------------------|--------------------|---------|-----------------------|--------------------|-----------|
| | | | Lillis (Q, C, W) | New (P, C, W) | | Lillis (Q, C, W) | New (P, C, W) | |
| 337 | 337 | 30 | 2.51 | 0.79 | 3.2 | 0.91 | 0.22 | 4.1 |
| | | 50 | 3.11 | 1.09 | 2.9 | 1.39 | 0.25 | 5.6 |
| | | 100 | 4.58 | 2.48 | 1.8 | 1.52 | 0.26 | 6.1 |
| 1944 | 1944 | 30 | 142.56 | 9.80 | 14.5 | 13.14 | 0.70 | 18.8 |
| | | 50 | 244.90 | 14.14 | 17.3 | 24.15 | 0.96 | 25.2 |
| | | 100 | 470.87 | 26.84 | 17.5 | 53.06 | 1.76 | 30.1 |
| 2676 | 2676 | 30 | 196.23 | 22.26 | 8.8 | 12.50 | 1.60 | 7.8 |
| | | 50 | 347.86 | 33.63 | 13.0 | 22.86 | 2.26 | 10.1 |
| | | 100 | 559.54 | 57.81 | 9.7 | 42.82 | 3.81 | 11.2 |

buffer library is large in industry designs, so our algorithm can achieve more significant improvement in practice.

We also compare our algorithm with the $O(|B|^2 n \log n)$ time algorithm of Shi and Li [12], without considering the buffer cost. The $O(|B|^2 n \log n)$ time algorithm is asymptotically the fastest algorithm reported. Simulation results are shown in Table IV for six buffer types on several industrial test cases. We can see that for multiple buffer types without considering buffer cost, our new algorithm is better than the $O(|B|^2 n \log n)$ time algorithm when n is small. The reason is that $O(|B|^2 n \log n)$ algorithm needs to use $|B|$ trees to store b_i -dominant solutions, which results in high overhead. Instead, our new algorithm only needs one tree to store b_k -dominant solutions, for b_k that the smallest driving resistance.

VI. HIGH DEGREE VERTICES

During buffer insertion, one problem is to deal with routing tree vertices of out-degree greater than 2. Although any such vertex can be replaced by a number of out-degree 2 vertices, the order is not unique. For example, there are three ways to replace an out-degree 3 vertex in Fig. 4 with out-degree 2 vertices. Each new vertex, solid or hollow, is a possible buffer position. Solid vertices involves merging and buffer insertion. Hollow points involve buffer insertion only.

Theorem 2 *Let v be an out-degree d vertex, then there are at most $n + 2^{O(d)}$ nonredundant candidates for $T(v)$, where n is the total number of candidates in the branches.*

Proof: For a vertex with out-degree d , the number of ways to merge the d branches by a sequence of 2-merges is $1 \cdot 3 \cdot 5 \cdots (2d - 3) = 2^{O(d)}$. This can be shown by solving the recurrence relation using generation function [13]. Fig. 4 shows the three ways for $d = 3$. We will first consider whether to insert a buffer at each hollow point. This way, the total number of candidates in the branches is at most $n + d$.

Now consider the merging points, shown as the solid points in Fig. 4. In each case, say Fig. 4(b), since there can be at most $d - 1$ new buffers, the number of candidates with new buffers is at most d . The candidates without new buffers will be the same as the candidates without new buffers in any other case,

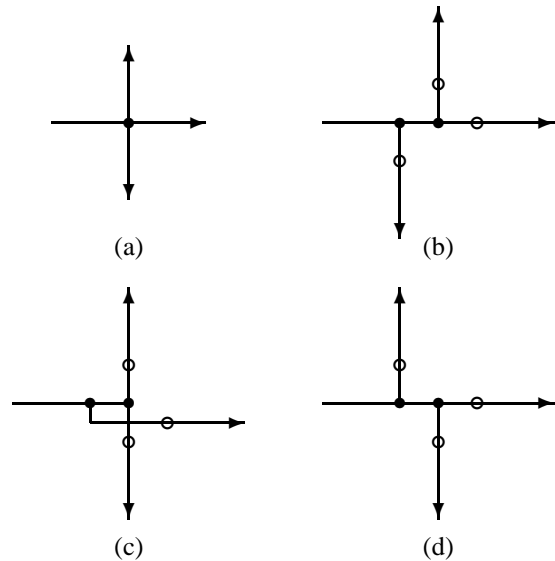


Fig. 4. Three ways to replace an out-degree 3 vertex (top left) by out-degree 2 vertices. For an out-degree d vertex, there are $1 \cdot 3 \cdot 5 \cdots (2d - 3)$ ways.

say Fig. 4(c). Therefore if we combine all the cases, there will be only $2^{O(d)}$ additional candidates. \square

VII. CONCLUSION

We prove the buffer cost minimization problem is NP-complete in general. On the other hand, we propose algorithms using the pre-buffer slack pruning technique for the buffer insertion problem, with buffer cost constraints. Experimental results show that pruning using (P, C, W) is more efficient than using (Q, C, W) . We also show an efficient way to merge vertices with degree greater than 2 in buffer insertion.

The authors thank Jiang Hu for discussion and Douglas B. West for reference.

REFERENCES

- [1] C. J. Alpert and A. Devgan, "Wire segmenting for improved buffer insertion," *Proc. 1997 DAC*, 588–593.

TABLE IV
SIMULATION RESULTS FOR SIX BUFFER TYPES WITHOUT BUFFER COST CONSTRAINTS.

| Sinks m | Buffer positions n | CPU Time (sec) | | Memory (MB) | |
|--------------|-------------------------|---------------------|--------|---------------------|-------|
| | | $O(B ^2 n \log n)$ | New | $O(B ^2 n \log n)$ | New |
| 337 | 337 | 0.42 | 0.06 | 0.13 | 0.05 |
| | 11384 | 10.92 | 8.17 | 0.77 | 2.20 |
| | 16780 | 16.78 | 17.58 | 1.09 | 4.54 |
| 1944 | 1944 | 2.50 | 0.43 | 0.26 | 0.25 |
| | 67408 | 73.4 | 86.52 | 1.67 | 19.06 |
| | 98632 | 113.15 | 187.96 | 2.37 | 39.05 |
| 2676 | 2676 | 3.41 | 0.65 | 0.27 | 0.34 |
| | 91600 | 99.06 | 137.91 | 1.69 | 25.94 |
| | 134101 | 152.79 | 293.71 | 2.39 | 52.06 |

- [2] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion for noise and delay optimization," *Proc. 1998 DAC*, 362–367.
- [3] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion with accurate gate and interconnect delay computation," *Proc. 1999 DAC*, 479–484.
- [4] C.-P. Chen and N. Menezes, "Noise-aware repeater insertion and wire sizing for on-chip interconnect using hierarchical moment matching," *Proc. 1999 DAC*, 502–506.
- [5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, 1979.
- [6] M. Hrkic and J. Lillis, "S-tree: a technique for buffered routing tree synthesis," *Proc. 2002 DAC*, 578–583.
- [7] J. Hu, C. J. Alpert, S. T. Quay, G. Gandham, "Buffer insertion with adaptive blockage avoidance," *IEEE Trans. Computer-Aided Design*, 22(4), 2003, 492–498.
- [8] 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 Trans. Solid-State Circuits*, 31(3), 1996, 437–447.
- [9] J. Lillis, C. K. Cheng and T.-T. Y. Lin, "Simultaneous routing and buffer insertion for high performance interconnect," *Proc. 1996 Great Lakes Symposium on VLSI*, 148–153.
- [10] T. Okamoto and J. Cong, "Buffered Steiner tree construction with wire sizing for interconnect layout optimization," *Proc. 1996 ICCAD*, 44–49.
- [11] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "The scaling challenge: Can correct-by-construction design help?" *Proc. 2003 ISPD*, 51–58.
- [12] W. Shi and Z. Li, "An $O(n \log n)$ time algorithm for optimal buffer insertion," *Proc. 2003 DAC*, 580–585.
- [13] S. P. Stanley, *Enumerative Combinatorics*, Cambridge University Press, Cambridge, 1999.
- [14] L. P. P. van Ginneken, "Buffer placement in distributed RC-tree network for minimal Elmore delay," *Proc. 1990 ISCAS*, 865–868.
- [15] 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*, 19(7), 2000, 819–824.