Optimal and Efficient Buffer Insertion and Wire Sizing

John Lillis, Chung-Kuan Cheng
Dept. of Computer Science and Engineering

Ting-Ting Y. Lin
Dept. of Electrical and Computer Engineering

University of California, San Diego
La Jolla, CA 92093-0114

ABSTRACT

We present optimal solutions to the following problems: (1) post-layout buffer insertion, (2) wire-sizing and (3) simultaneous buffer insertion and wire-sizing. We optimize a practical objective function: *required arrival time*. To the best of our knowledge, this work represents the first sub-exponential algorithms for these problems. In experiments, we observe substantial improvements over the results of [5] for buffer insertion, and up to 25% improvement in delay by wire-sizing.

1. INTRODUCTION

Two effective methods for timing optimization of VLSI and MCM systems are *buffer insertion* (e.g. [7, 5]) and *wire-sizing* [1, 2, 6]. In this paper we give algorithms solving these problems both simultaneously and separately.

1.1 Buffer Insertion

Two basic methods of buffer insertion have been studied: *pre-layout* insertion (e.g. [7]) and *post-layout* insertion (e.g. [5]).

In this paper, we attack the post-layout problem for the following reasons: (1) topological information can be utilized in timing analysis, (2) it does not impose an apriori topology on the net in the form of a buffer tree, and (3) optimal solutions are tractable in many practical situations (while the pre-layout problem is NP-hard [7]).

Previously, [5] established that substantial reductions in delay can be achieved by post-layout buffer insertion. They proposed a Dynamic-Programming algorithm using the fact that maximum buffer fanout is often limited in practice to reduce the search space of the problem. While practical for small libraries and small fanout, the algorithm could not efficiently handle large fanout and libraries.

1.2 Wire Sizing

Technology trends have resulted in wire resistance contributing significantly to delay; hence the smallest width is not always optimal. Cong, Leung and Zhou [1, 2] proposed an $O(n^r)$ wire sizing algorithm for an n segment tree with r possible wire widths; their objective function is a linear combination of the sink delays. Later, in [6], Sapetnekar studied the more practical objective of *minimizing maximum delay* and showed that Cong's $O(n^r)$ algorithm did not apply, leaving the enumerative $O(r^n)$ algorithm as the best upper bound.

1.3 Our Contributions

This paper presents two main contributions:

- The first sub-exponential algorithm for optimal post-layout buffer insertion; we later incorporate simultaneous, optimal wire-sizing.
- From our work on simultaneous wire-sizing, we derive, to the best of our knowledge, the first sub-exponential optimal wire-sizing algorithm.

Our algorithms run in polynomial time when all capacitive values in the problem instance are, or can be mapped to, *polynomially-bounded integers*.

As such, our algorithms are *pseudo-polynomial* [4], but are of significant practical value since capacitive values tend to be approximate in nature and small variances in their values are not expected to affect the solution significantly; hence, any error associated with mapping to an integer domain should be tolerable or non-existent for a reasonably large integer domain. We note that, for wire-sizing, since routing is typically done on a grid, the possible capacitive values are discrete in nature since wire segments are multiples of a basic grid length; thus, our algorithm can be implemented such that no discretization is necessary and no error is possible.

We demonstrate the feasibility of our algorithms experimentally.

1.4 Notational Conventions

In this paper we use the following notation:

T_v	routing tree rooted at node v.
$l(v)$, $r(v)$	left and right children of node v.
$p(v)$	parent of node v.
e_v	edge from node v to its parent.
c_e	capacitance of edge e.
r_e	resistance of edge e.
c_b	input capacitance of buffer b.
r_b, r_g	output resistance of buffer b or gate g.
d_b, d_g	intrinsic delay of buffer b or gate g.
q_v	required arrival time of sink node v.
leaves(T)	set of leaves of tree T.

2. DELAY MODEL

Delay on a root-sink path is made of (1) wire delay and (2) delay through buffers and the driving gate.

We use the Elmore delay model [3] for wire delay. Letting $c(T_v)$ be the capacitance at node v, the Elmore delay of edge e_v is

0-7803-2584-2/95 $3.00

$$r_{e_v}(\frac{c_{e_v}}{2} + c(T_v)).$$

The delay through a buffer or gate b at node v is determined by the parameters $c(T_v)$, b's *intrinsic delay* d_b and *output resistance* r_b. The delay through the buffer is

$$\text{buf\_delay}(v, b) = d_b + r_b \cdot c(T_v).$$

Note the key to buffer insertion is the *decoupling* effect of buffers on $c(T_v)$. I.e., a buffer *decouples* the capacitance of its descendants from its ancestors.

Subsequently, we use *delay*(u, v) to indicate the total wire and buffer delay from u to v in a buffer tree.

3. PROBLEM FORMULATIONS

Given the above delay model, we define the *Optimal Buffer Insertion Problem* in the following.

We are given a binary routing tree T with driving gate g and buffer library B with the following parameters: resistance r_g and intrinsic delay d_b of driving gate g, capacitance c_e and resistance r_e of each edge e, input capacitance c_u and required time q_u of each sink u, and, for each buffer $b \in B$, intrinsic delay d_b, output resistance r_b and input-capacitance c_b.

Our objective function is a common generalization of minimization of maximum delay: maximization of *required arrival time*. The required arrival time at node v, $q(T_v)$ is the latest time at which the input(s) of v must be available for the required arrival times of all sinks to be met. Formally, $q(T_v)$ is defined as:

$$q(T_v) = \min_{u \in \text{leaves}(T_v)} (q_u - \text{delay}(v, u)).$$

Given this framework, our goal is to solve the **Optimal Buffer Insertion Problem (OBI):** Given routing tree T and buffer library B, assign each internal node of T a member of the set $B \cup \{\emptyset\}$ where $\emptyset$ indicates "no buffer inserted" such that $q(T)$, is maximized.

The **Optimal Wire Sizing Problem** is defined similarly: Given T and a set of wire widths W, each width parameterized by a capacitance per unit length and resistance per unit length, select widths for each wire in T such that $q(T)$, is maximized.

These methods can be combined for simultaneous optimization.

4 ALGORITHM CapDP

We now present a pseudo-polynomial time algorithm solving OBI. In the following we assume that all capacitive values are integers in the range $1..c_{\max}$ where $c_{\max}$ is the largest capacitive value and is polynomially bounded in the size of the net.

We have devised an algorithm, *Capacitance-based Dynamic-Programming* or *CapDP*, described in pseudo-code in Figure 1. The algorithm proceeds in bottom-up order. Key to the algorithm is the set C_v: the set of all possible capacitive values that the sub-tree T_v can present to its parent over all possible buffer configurations (including c_{e_v}). Formally,

$$C_v = \begin{cases} \{c_v + c_{e_v}\} & v \text{ a sink} \\ \{c_l + c_r + c_{e_v} | c_l \in C_{l(v)}, c_r \in C_{r(v)}\} \cup & \\ \quad \{c_b + c_{e_v} | b \in B\} & \text{otherwise.} \end{cases}$$

The two sets of the union correspond to the case where no buffer is placed at v and the case where some buffer b is placed at v respectively.

For arbitrary capacitive values, $|C_v|$ may grow exponentially in n However, since $c_{\max}$ is polynomially bounded, $|C_v| = O(n \cdot c_{\max})$ and also is polynomial in size. This is clear since, regardless of buffer configuration, any wire, buffer etc. is affected by $O(n)$ other objects and their capacitive values.

The algorithm *CapDP* constructs a matrix *OPT_Q* where OPT_Q$(v, c) = q$, the optimal required arrival time at v when v presents load capacitance $c \in C_v$ to its parent (including c_{e_v}). We compute this matrix inductively and obtain the optimal solution for the root by selecting the capacitive value maximizing the required arrival time.

We now outline the pseudo-code in Figure 1.

Lines 4-12 represent initial conditions and base cases for the algorithm: lines 4-9 indicate that the algorithm has not yet found feasible solutions for internal nodes, lines 9-12 define C_v for sinks v.

Lines 13-32 represent the Dynamic-Programming portion of the algorithm. We examine all pairs of capacitive values c_l and c_r from v's left and right children (line 16), to construct C_v (lines 23, 24) and compute OPT_Q. When node v is not the root (lines 22-36), we construct a candidate solution from the optimal solutions of its children (based on c_l and c_r) for the no-buffer (lines 21-24) and buffer (lines 25-29) case and account for the delay of e_v and buffer delay if applicable. The candidate solutions are compared with the best solutions so far.

When v is the root (lines 17-21), we take into account the delay of the driving gate to compute the resulting required arrival time.

Not indicated in the pseudo-code is how to retrieve the optimal buffer configuration. This is done by maintaining, in addition to the required arrival time of the sub-problems, the following: (1) which

Algorithm: CapDP

```
1. Input: Routing Tree T = (V, E), driving gate g,
2.        Buffer Library B
3. Output: Best achievable required time, q_max(T)
4.  q_max(T) = −∞
5.  Foreach internal node v and possible cap. c {
6.    C_v = ∅
7.    OPT_Q(v, c) = −∞
8.  }
9.  Foreach sink node v ∈ V {
10.   C_v = {c_v + c_{e_v}}
11.   OPT_Q(v, c_v + c_{e_v}) = q_v − delay(e_v)
12. }
13. Foreach internal node v in bottom-up order{
14.   Foreach pair c_l ∈ C_{l(v)} and c_r ∈ C_{r(v)} {
15.     If (v = g) {          /* Root Case */
16.       q_l = OPT_Q(l(v), c_l) − buf_delay(v, g)
17.       q_r = OPT_Q(r(v), c_r) − buf_delay(v, g)
18.       q_max(T) = max(q_max(T), min(q_l, q_r))
19.     }
20.     Else {                /* Internal Node Case */
21.       c_t = c_l + c_r + c_{e_v}    /* Total Cap */
22.       C_v = C_v ∪ {c_t}
23.       q_new = min(OPT_Q(l(v), c_{l(v)}),
                      OPT_Q(r(v), c_{r(v)})) − delay(e_v)
24.       OPT_Q(v, c_t) = max(OPT_Q(v, c_t), q_new)
25.       Foreach buffer b ∈ B {
26.         c_t = c_b + c_{e_v}
27.         q_new = min(OPT_Q(l(v), c_l),
                        OPT_Q(r(v), c_r))
                    −delay(e_v) − buf_delay(v, b)
28.         OPT_Q(v, c_t) = max(OPT_Q(v, c_t), q_new)
29.       }
30.     }
31.   }
32. }
33. return q_max(T)
```

Figure 1: Pseudo-code for Algorithm CapDP

buffer was used (or none), and (2) the capacitive values presented by the left and right children resulting in the optimal solution. Given this, we can recursively construct the optimal tree.

We can show by induction the following:

Theorem 1: *Algorithm CapDP correctly computes* OPT_Q(v, c) *for* T *and thus solves OBI.*

4.1 Run-Time Analysis

We bound the run-time of CapDP in terms of the tree size n, the size of the buffer library $|B|$ and the largest capacitive value $c_{\max}$. We know that, for all nodes v, $|C_v| = O(n \cdot c_{\max})$. Thus, for each node, we perform $O(|B|(n \cdot c_{\max})^2)$ work since we examine all pairings from $C_{l(v)}$ and $C_{r(v)}$ (note that this is a gross over-estimate of the typical case). This gives an overall run-time of

$$O(n \cdot |B|(n \cdot c_{\max})^2) = O(n^3|B| \cdot c_{\max}^2).$$

4.2 Implementation Details

One issue is how to handle capacitive values expressed in floating point. We deal with this by mapping capacitive values to an integer domain. In this process, we are essentially rounding the values. In our implementation, we can select the size of the integer domain upon which to map the values. Experiments have shown that relatively small domain sizes are sufficient for high quality solutions. No improvement in the solution was possible when the domain-size increased beyond a threshold - typically less than one hundred. Also, since the given values are, by nature, approximate, we do not expect much impact from the mapping in practice.

Another issue is dealing with the sparseness of matrix OPT_Q with respect to capacitance c. This is addressed by storing the *OPT_Q* in a hash table. Hence, we use storage proportional to the valid entries in the matrix, maintain fast access time, and have run-time typically much less than the upper-bound.

4.3 Algorithm Extension: Wire Sizing

We have generalized *CapDP* to simultaneously perform optimal wire sizing. For ease of presentation, we have omitted code for wire sizing in Figure 1. The extension, however, is relatively straightforward. We assume that there is a fixed set of discrete wire widths from which we may choose. We simply generalize the definition of C_v to include the capacitive values associated with all possible widths of edge e_v and compute the delay associated with that edge accordingly.

Remark: recall that the typical grid structure eliminates possible error as previously discussed.

4.4 Dealing With Inverters

Presumably the signal that reaches the sinks must be the signal that left the source, not its inverse. This is an issue since inverters are commonly used buffers. We deal with this by adding a third dimension to *OPT_Q*. I.e. *OPT_Q*(v, c, p) is the optimal required time at v when capacitance c is presented to v's parent *and* the polarity of the incoming signal is p. This maintains optimality rather than simplifying the problem by, for example, assuming each buffer is a pair of inverters. This only adds a constant factor to the run-time.

4. EXPERIMENTAL RESULTS

We implemented *CapDP* on a Sun SPARC 20 workstation under the C/UNIX environment. Routing trees were generated in the same manner as in [5] based on modern technology parameters.

Input capacitance of sink nodes were set to 0.01pF, wires at 1μm width were assumed to have 0.1fF per μm and .05Ω per μm, and the driving gate had an output resistance of 300Ω.

Buf #	Intrinsic Delay (ps)	Load Cap (pF)	Output Res (Ω)	Inv
1	300.0	0.01	3170.0	0
2	300.0	0.02	1585.0	1
3	300.0	0.04	792.0	1
4	300.0	0.08	396.0	0
5	300.0	0.08	396.0	1
6	300.0	0.16	198.0	1
7	300.0	0.16	198.0	0

Table 1: *Buffer Library used in experiments*

	DPABI		CapDP	
n	delay[k]	cpu	delay	cpu
25	1621[5]	0.1	1621	0.3
50	1829[12]	80.0	1829	3.3
100	2007[6]	0.5	2007	16.2
200	2424[8]	24.1	2424	50.4
300	2760[11]	5062	2760	102.3

Table 2: *DPABI vs. CapDP.* $|B| = 2$, *k=fanout*

We first compared the performance of DPABI and CapDP on trees ranging in size from 25 to 300 sinks. Table 2 shows the results when only buffers 1 and 2 were used; Table 3 shows the results when all 7 buffers are used. Delay is measured in nano-seconds, cpu-time in seconds. The entries for DPABI are the best solution achievable in approximately 2 hours or less. The value k is the fanout limit used to achieve the solution (the smallest k giving that solution). For simplicity, required times were identical, making maximum delay the measure of interest.

The main conclusion is that, while DPABI may be competitive for a small buffer library as in Table 2, it cannot compete with CapDP for a larger buffer library as in Table 3. In fact, we observe a 10% delay improvement on average in Table 3 and run-time improvements often of orders of magnitude. Note also that the "natural" value of k is technology dependent and hence CapDP is more flexible since it has no inherent fanout limit.

Wire-sizing results appear in Table 4. For a fixed topology of 15 sinks, we ran CapDP allowing various widths. Wire lengths were longer to approximate MCM geometric distances where wire-sizing helps. We expect similar results for IC's where geometries smaller than 1μm are used. For this case, no improvement was observed by allowing widths beyond 4μm in which case we observe an improvement of 25% in delay in modest run-time.

	DPABI		CapDP	
n	delay[k]	cpu	delay	cpu
25	1135[7]	97	1068	0.7
50	1422[7]	1734	1230	1.8
100	1669[6]	1527	1494	8.3
200	1947[6]	4021	1718	40.4
300	2133[5]	8624	1864	97.3

Table 3: *DPABI vs. CapDP.* $|B| = 7$, *k=fanout*

widths(μm)	delay	cpu
1	127	-
1-2	101	1.7
1-4	94.5	2.3
1-6	94.5	3.2

Table 4: *Effect of Wire Sizing (n=15, Delay in ns)*

5. CONCLUSIONS

We have described an algorithm, *CapDP*, which derives optimal solutions to the *post-layout buffer insertion problem*, the *wire-sizing problem* and to these problems simultaneously. CapDP represents, to the best of our knowledge, the first subexponential algorithm for these problems. The algorithm runs in pseudo-polynomial time and is efficient in practice. We have shown the efficiency and effectiveness of CapDP expermentaly.

REFERENCES

[1] J. Cong, K.S. Leung, and D. Zhou, "Performance-driven interconnect design based on distributed RC model," *Proc. ACM/IEEE Design Automation Conf.*, pp. 606-611, 1993.

[2] J. Cong, K.S. Leung, "Optimal wiresizing under the distributed Elmore delay model," *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pp. 634-639, 1993.

[3] W.C. Elmore, "The Transient Response of Damped Linear Network with particular Regard to Wideband Amplifiers," *J. Applied Physics* 19 (1948), pp 55-63.

[4] M. Garey and D. Johnson, *Computers and Intractability* Freeman, 1979.

[5] J. Lillis, C.K. Cheng and T.T. Y. Lin, "Optimal Buffer Insertion into a Steiner Tree," *Unpublished (Submitted to ACM/IEEE Design Automation Conf. 1995.)*

[6] S. S. Sapatnekar, "RC Interconnect Optimization under the Elmore Delay Model," *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp. 387-391.

[7] H.J. Touati, "Performance-Oriented Technology Mapping," Ph.D dissertation, Memorandum No. UCM/ERL M90/109, Dept. of Electrical Engineering and computer Science, UC Berkeley, 28 November 1990.