

# Making Fast Buffer Insertion Even Faster via Approximation Techniques

Zhuo Li, C. N. Sze, Jiang Hu and Weiping Shi

Department of Electrical Engineering

Texas A&M University

Charles J. Alpert

IBM Austin Research Lab

# Outline

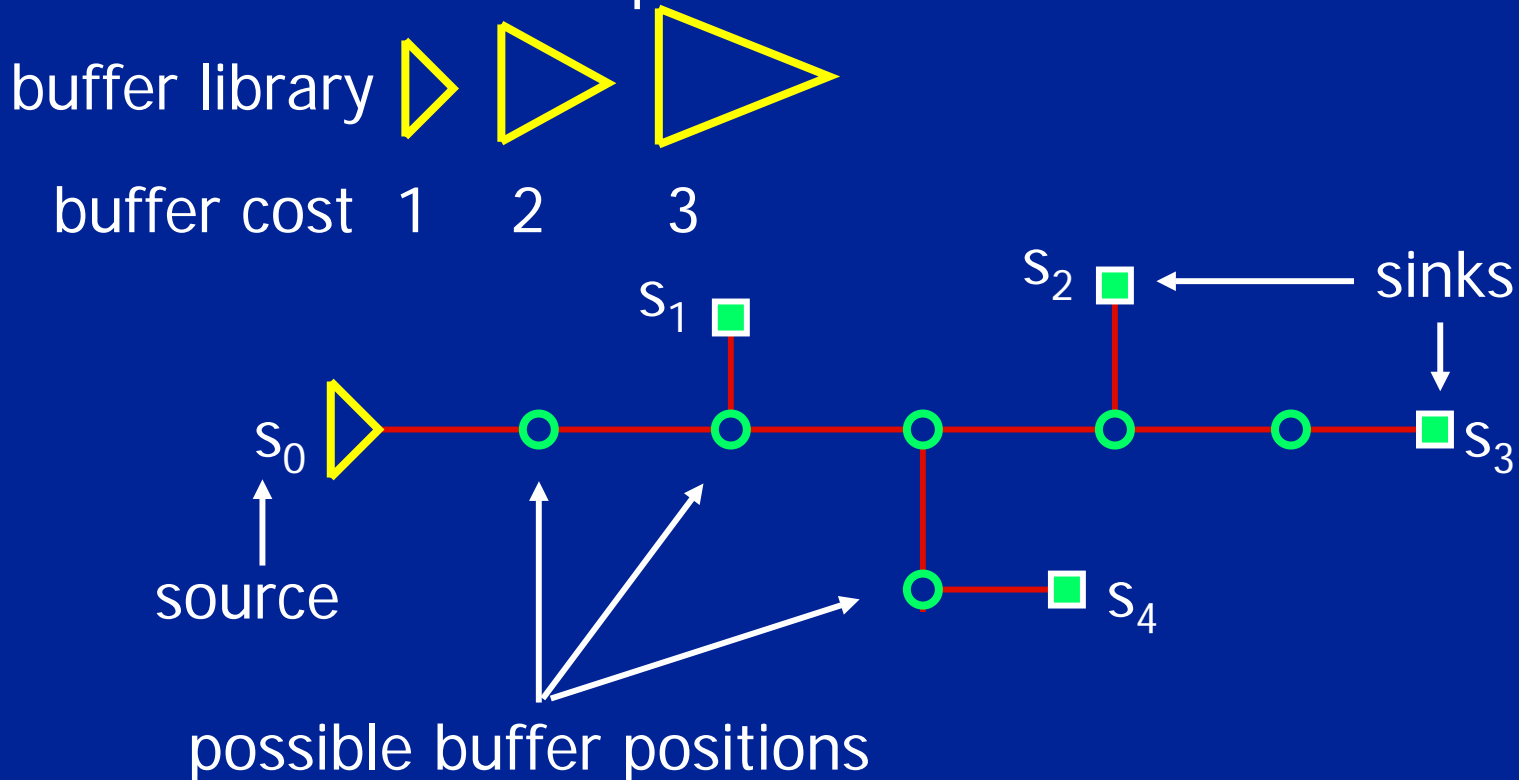
- Introduction
- Approximation techniques
  - ⊙ Aggressive pruning
  - ⊙ Convex pruning
  - ⊙ Clever buffer Library
- Experimental Results
- Conclusion

# Introduction

- Buffer insertion and sizing is one of the most effective method for reducing interconnect delay.
- Saxena, et al. [TCAD 04] predicted that the number of buffers for inter-block or intra-block communications will explode
  - ⦿ For 65nm technology, 15% of cells are buffers
  - ⦿ For 45nm technology, 35% of cells are buffers
  - ⦿ For 32nm technology, 70% of cells are buffers
- With such large number of buffers, fast algorithms for buffer insertion is crucial for timing closure.

# Problem Formulation

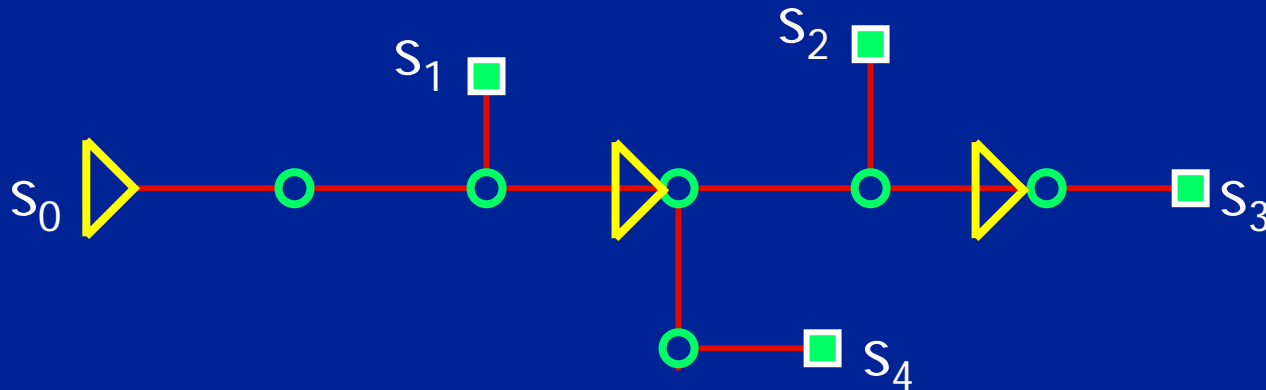
- Given: A routing tree,  $n$  possible buffer positions, sink capacitances and required arrival times (RAT), a buffer library and a cost function  $W$  (for power, area, etc), unit wire resistance and capacitance.



# Maximum Slack Problem

- Find: Where to insert buffers so that the slack at the source  $Q(s_0)$  is maximized.

$$Q(s_0) = \min_{i>0} \{ RAT(s_i) - delay(s_0, s_i) \}$$



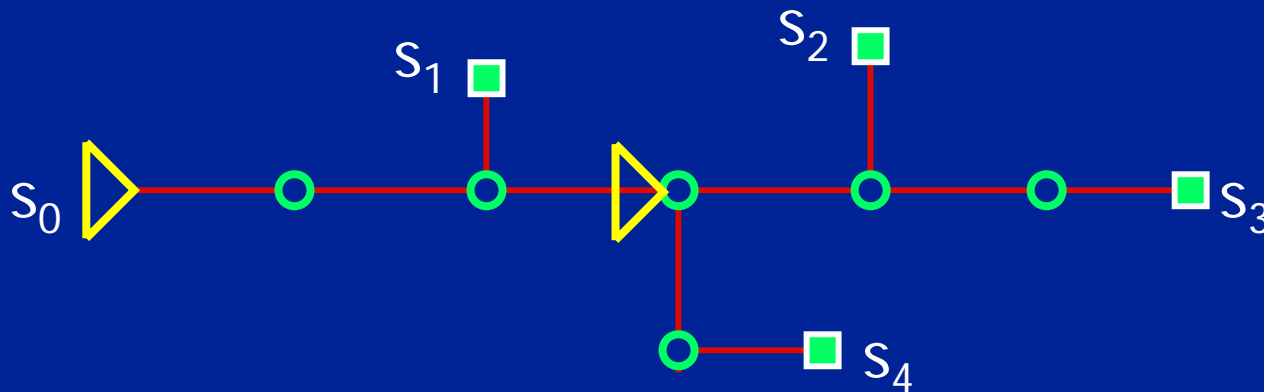
without buffer,  $Q(s_0) = -50$

with 2 buffers,  $Q(s_0) = 100$

# Minimum Cost Problem

- Find: Where to insert buffers so that the slack at the source  $Q(s_0) > 0$  and the buffer cost  $W(s_0)$  is minimized.

$$W(s_0) = \sum W(b_i)$$



with 1 buffer,  $Q(s_0) = 10$

# Previous Research

## ● Maximum Slack

- ⊙ van Ginneken [ISCAS 90]:  $O(n^2)$  time and space, where  $n$  is the number of buffer positions.
- ⊙ Lillis, Cheng and Lin [TCAS 96]:  $O(b^2n^2)$  time and space for  $b$  buffer types.
- ⊙ Shi and Li [DAC 04]:  $O(n \log n)$  time for 2-pin nets,  $O(n \log^2 n)$  time for multi-pin nets.  $O(n \log n)$  space.
- ⊙ Li and Shi [DATE 05]:  $O(bn^2)$  time and space for  $b$  buffer types.

## ● Minimum Cost

- ⊙ Lillis, Cheng and Lin [TCAS 96]: pseudo-polynomial time algorithm.
- ⊙ Shi, Li and Alpert [ASPDAC 04]: buffer cost minimization is NP-hard if  $b$  is a variable.

# Dynamic Programming

- $(Q, C, W)$  representation
  - ⊙ Each candidate solution is represented by a  $(Q, C, W)$  triple, where  $Q$  is slack,  $C$  is capacitance, and  $W$  is buffer cost.
  - ⊙ For two candidates  $A_1$  and  $A_2$  of the same branch, if  $Q(A_1) \leq Q(A_2)$ ,  $C(A_1) \geq C(A_2)$  and  $W(A_1) \geq W(A_2)$ , then  $A_1$  is redundant.
- Add a wire
  - ⊙ Each  $(Q, C, W)$  becomes  $(Q - (\text{wire delay}), C + (\text{wire cap}), W)$ .
- Add a buffer
  - ⊙ Each  $(Q, C, W)$  produces  $b$  new candidate  $(Q - (\text{buf delay}), (\text{buf cap}), W + (\text{buf cost}))$ .
- Merge two branches
  - ⊙ Each pair of candidates  $(Q_l, C_l, W_l)$  and  $(Q_r, C_r, W_r)$  becomes  $(\min\{Q_l, Q_r\}, C_l + C_r, W_l + W_r)$ .

# Outline

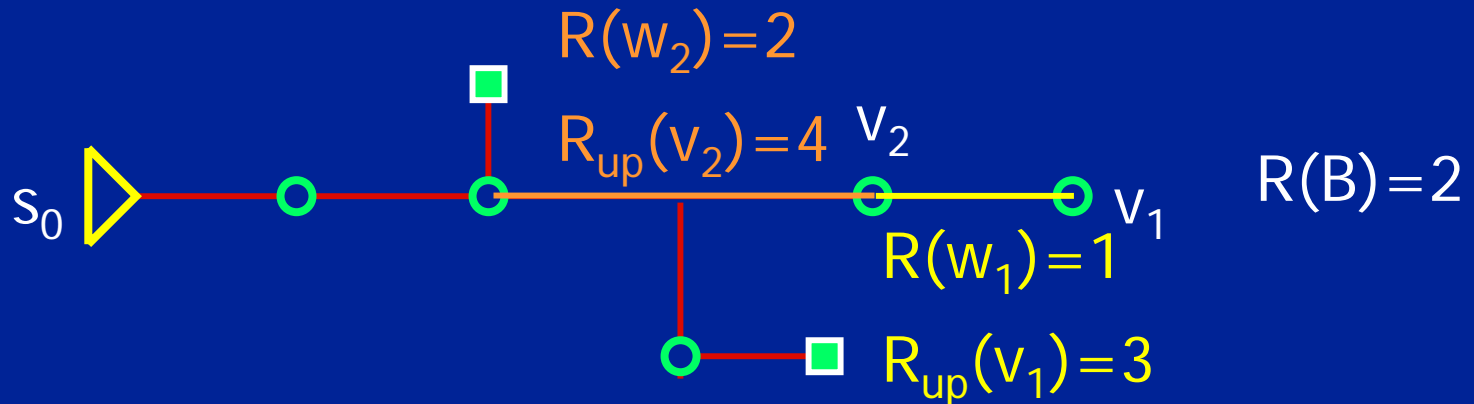
- Introduction
- Approximation techniques
  - ⊙ Aggressive Pruning
  - ⊙ Convex Pruning
  - ⊙ Clever Buffer Library
- Experimental Results
- Conclusion

# Approximation Techniques

- An approximation algorithm finds a solution close to optimal, but runs much faster or is much better in other measures.
  - ⦿ Sacrificing 1% slack may speed up the algorithm 10X
  - ⦿ Sacrificing 1% slack may save 20% buffers
- Most industrial nets are small and medium size, and buffer library size is large.

# Predictive Pruning

- Traditional  $(Q, C)$  or  $(Q, C, W)$  pruning is too conservative
- Predictive pruning uses **upstream resistance  $R_{up}$**  to prune candidates
  - ⊙  $R_{up}(v)$  of node  $v$  is the resistance from  $v$  to the nearest upstream buffer position, including the upstream buffer/driver resistance



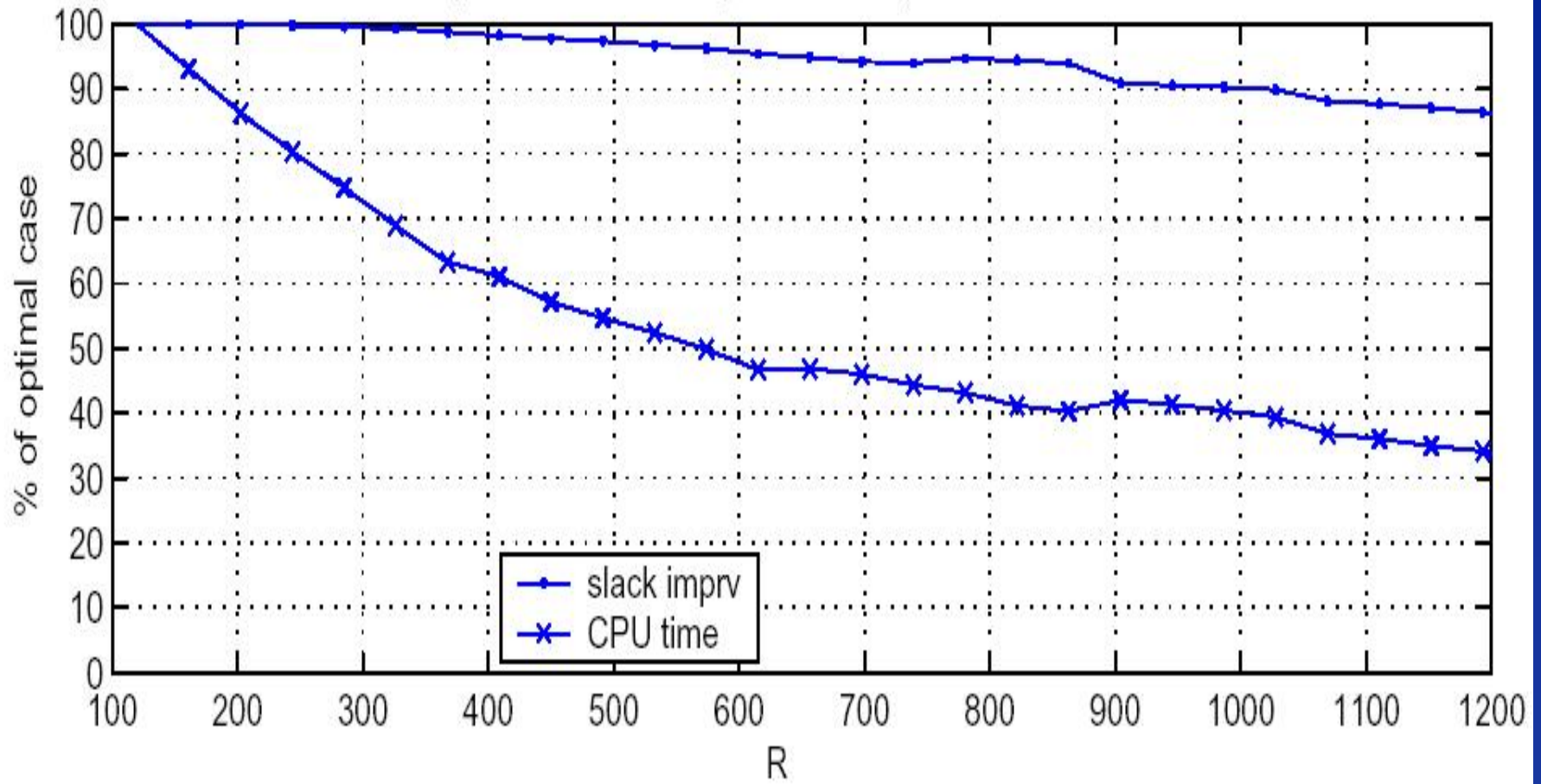
# Predictive Pruning

- For two candidates  $(Q_1, C_1)$  and  $(Q_2, C_2)$  at node  $v$ , if  $C_1 \geq C_2$  and  $Q_1 - C_1 * R_{up} \leq Q_2 - C_2 * R_{up}$ . Then candidate  $(Q_1, C_1)$  can be pruned.
- Example
  - ⊙ Consider two candidates with  $(Q, C)$  being  $(90, 30)$  and  $(50, 10)$  and  $R_{up} \geq 2$ .
  - ⊙ Add  $R_{up}$  to both candidates, their slacks will be  $90 - 30 * R_{up}$  and  $50 - 10 * R_{up}$ .
  - ⊙ Since  $R_{up} \geq 2$ , then  $90 - 30 * R \leq 50 - 10 * R$ .  
Therefore candidate  $(90, 30)$  can be pruned.
- Predictive pruning for  $(Q, C, W)$  is similar.



# Speed, Slack & $R_{up}$ Tradeoff

Comparison of the change in slack imprv and CPU-time

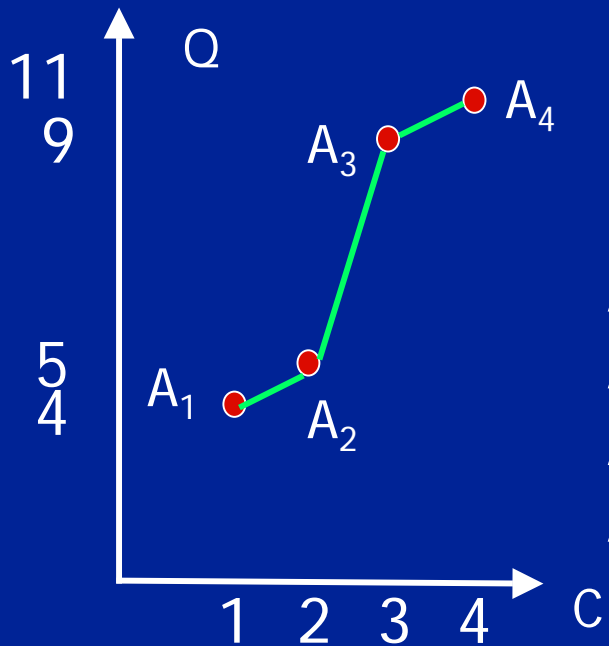


# Outline

- Buffer Insertion
- Approximation techniques
  - ⊙ Aggressive Pruning
  - ⊙ Convex Pruning
  - ⊙ Clever Buffer Library
- Experimental Results
- Conclusion

# Convex Pruning

- Consider 4 candidates with  $(Q, C)$  values as follows



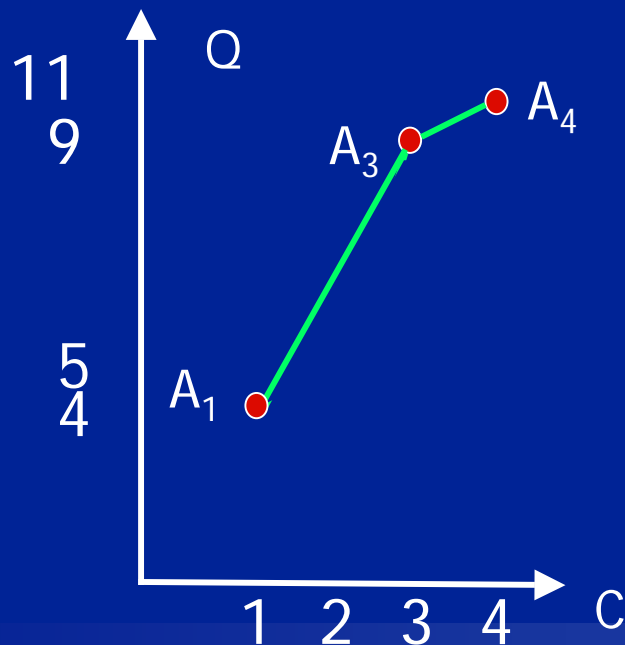
As resistance is added, Q becomes worse

$A_1: (4, 1)$	$(3, 1)$	$(2, 1)$	$(1, 1)$
$A_2: (5, 2)$	<del><math>(3, 2)</math></del>		
$A_3: (9, 3)$	$(6, 3)$	$(3, 3)$	<del><math>(0, 3)</math></del>
$A_4: (11, 4)$	$(7, 4)$	<del><math>(3, 4)</math></del>	

Red arrows indicate the progression from left to right in each row.

# Convex Hull

- If we only add resistance, the candidates that can ever give the best  $Q$  must be on the convex hull in the  $(Q, C)$  plane.
- However if merge is involved, candidates that are not on the convex hull may still be useful.



Using Graham's scan,  
the convex pruning can  
be done in linear time

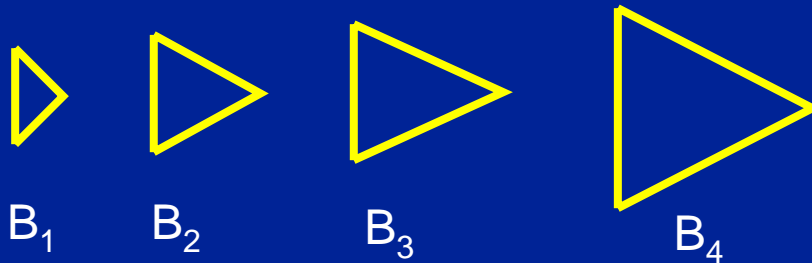
# Clever Buffer Library

- Modern libraries contain hundreds of different buffers with different characteristics.
  - ⦿ Polarity, input capacitance, driving resistance, intrinsic delay, noise margin, power, area, etc
- Buffer library size has square effect on running time.
- Alpert, et al. [ICCD 00] used clustering to select a subset of the buffer library. Reduced library size with slack degradation.

# Van Ginneken Style Algorithm

- For  $N$  candidates and  $b$  buffer types,  $O(bN)$  time to generate  $O(bN)$  new candidates.
- Some candidates are not worth considering, such as small buffers driving large capacitance.

Buffer library



$(Q_1, C_1, W_1)$  Try  $B_1 \sim B_4$

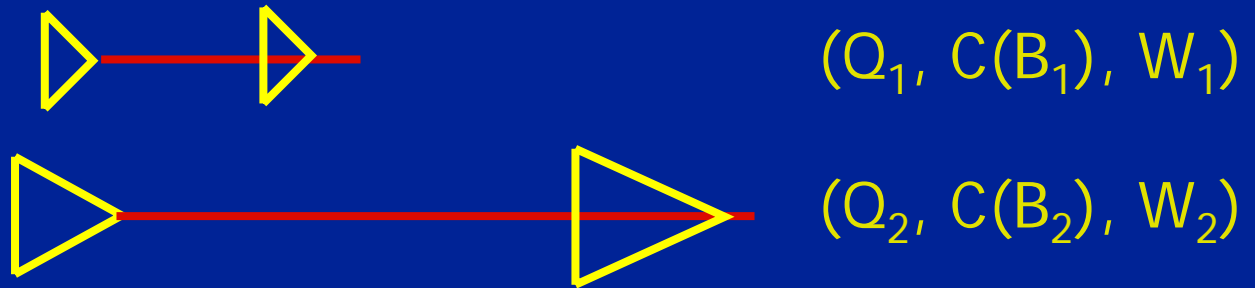
$(Q_2, C_2, W_2)$  ...

...

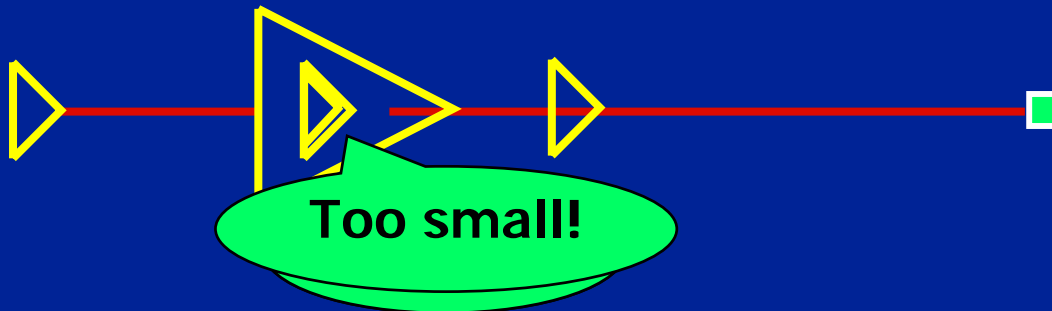
$(Q_n, C_n, W_n)$  ...

# Another Way?

- VG Style Algorithm: For every buffer, choose the best load to give the best slack.



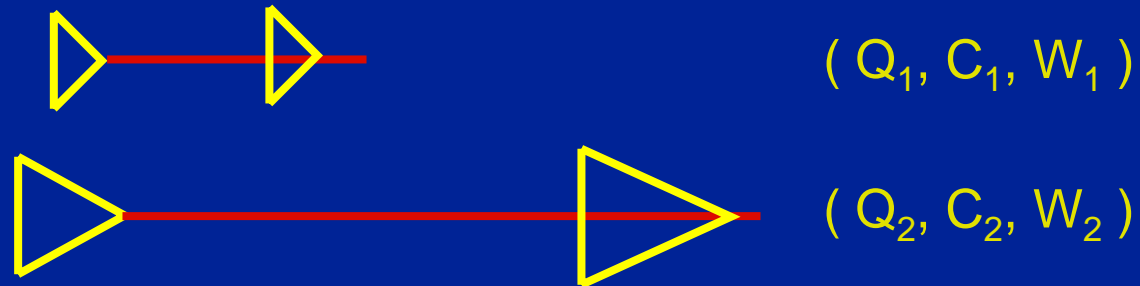
- New method: For every load, choose the best buffer to give the best predicted delay.



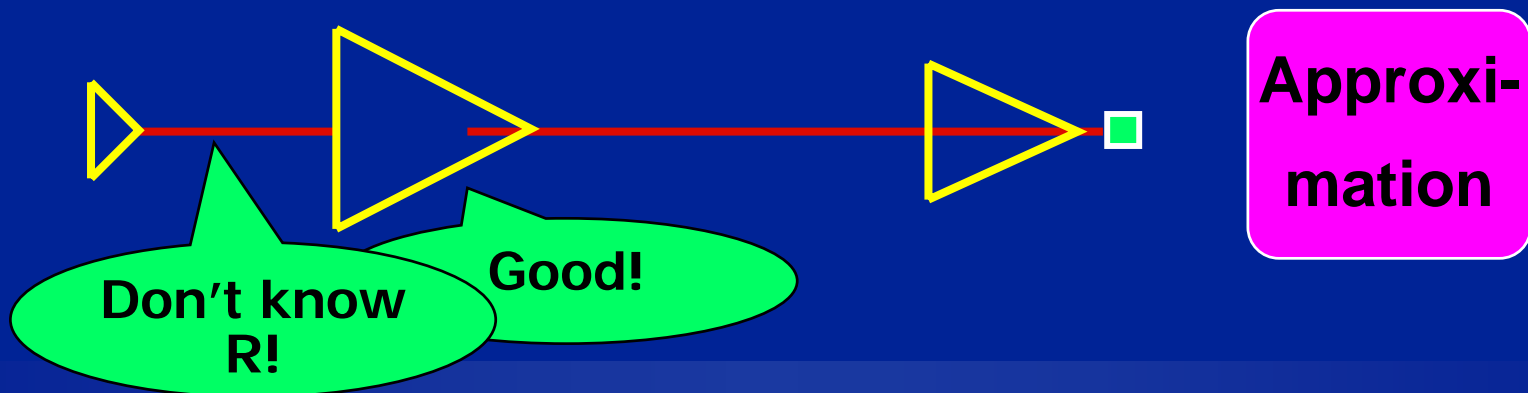
# Another Way?

- VG Style Algorithm: For every buffer, choose best load to give best slack. Optimal but expensive.

Optimal

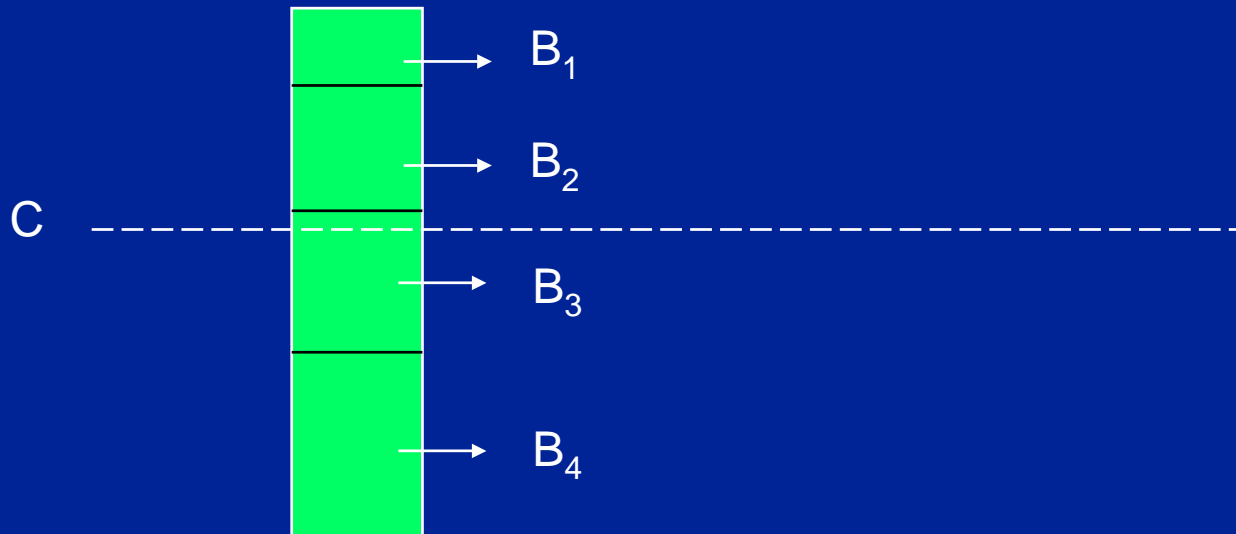


- New method: For every load, choose one buffer to give best predicted delay. Approximation but efficient.



# Table Based Buffer Insertion

- Build a table to select buffers according to load capacitance  $C$



$( Q_1, C_1, W_1 )$   $B_1$

$( Q_2, C_2, W_2 )$   $B_2$

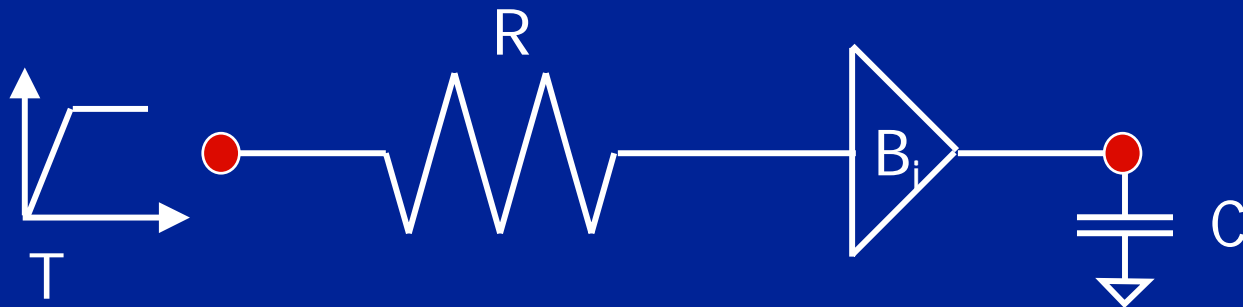
...

$( Q_n, C_n, W_n )$   $B_4$



# Build Lookup Table

- For  $N$  candidates and  $b$  buffer types,  $O(N)$  time to generate  $k$  new candidates,  $k \leq b$ .
- Table lookup
  - ⦿ For each possible values of load capacitance  $C$ , typical length wire resistance  $R$  and slew rate  $T$ , find the delay of each buffer  $B_i$ .
  - ⦿ Select the buffers that give the best delay for each  $C$ .
  - ⦿ Flexible to add other tables: polarity, cost, slew, etc.



# Outline

- Introduction
- Approximation techniques
  - ⊙ Aggressive Pruning
  - ⊙ Convex Pruning
  - ⊙ Clever Buffer Library
- **Experimental Results**
- Conclusion

# Experimental Results

- On three groups of nets from industrial 300k ASICs, where  $m$  is the number of sinks.

Test Cases	$m \leq 5$	$5 < m \leq 20$	$20 < m \leq 50$	$50 < m \leq 100$	$100 < m$	Total
Nets in Chip-A	944	56	0	0	0	1000
Nets in Chip-B	0	29	581	345	45	1000
Nets in Chip-C	2	1478	2956	513	51	5000

- Buffer library of 24 buffers, inverting and non-inverting

# Compare CPU Time

Test Case	Algorithms	Slack Improve	CPU time
Chip-A	Lillis-Cheng-Lin	5954.9	315.1
	Shi-Li-Alpert	5954.9	280.6
	new	5945.5	33.5
Chip-B	Lillis-Cheng-Lin	1310.6	1861.3
	Shi-Li-Alpert	1310.6	860.3
	new	1290.0	75.0
Chip-C	Lillis-Cheng-Lin	2868.9	3577.4
	Shi-Li-Alpert	2868.9	1881.8
	new	2785.4	20.4

# Compare with Small Library

- Compared with Library selection technique of Alpert et al., combined with the fastest optimal algorithm of Shi-Li-Alpert

Test Case	Library	Slack Imp	# Buffers	CPU
Chip-A	4 (LibSel)	5905	9864	59.1
	24 (new)	5945	9724	33.5
Chip-B	4 (LibSel)	1287	10235	220.0
	24 (new)	1290	9746	75.0
Chip-B	4 (LibSel)	2755	32361	460.4
	24 (new)	2785	29776	175.6

# Conclusion

- We present three approximation techniques
  - ⊙ Significantly faster than previous best methods, even for nets of medium or small sizes.
    - 9~25 times faster than dynamic programming algorithms of Lillis-Cheng-Lin [TCAS 96]
    - 8~11 times faster than predictive pruning of Shi-Li-Alpert [ASPDAC 04]
    - Only sacrificing 3% slack optimality.
  - ⊙ Easy to be embedded in existing buffer insertion algorithms, and for considering other objectives such as polarity, slew, and blockage.
- Trade-off between slack, buffer cost and CPU time can be adjusted according to user's needs.