A Simple and Efficient Algorithm for Finding Longest Simple Paths in Cyclic Combinational Circuits

Ali Dasdan and Santanu Kolay

February 28, 2004
PrimeTime
Synopsys, Inc.
Mountain View, CA
Outline

• Static timing analysis (STA)
• Combinational loops
• Problem & complexity
• Definitions & notation
• Algorithms
  ■ see the paper for complexity analyses
• Test suite
• Experimental results
• Conclusions & future work
Simple STA Flow

- RTL
  - Synthesis
    - Gate level
  - Place & Route
    - Layout
  - Design Signoff

- STA tool
  - Timing constraints (.sdc)
  - Design (.v, .vhdl)
  - Path constraints (.sdf)
  - Delays/parasitics (.sdf, .SPEF)

- Tech lib
  - Models
  - Commands / Variables (.tcl, .sdc)

- Delay, slew, cap, etc.
Simple STA Tool

STA tool:
- circuit to timing graph
- compute delays & slews
- compute arrival times
  (compute longest paths)
- compute required times
- compute slacks
- report violating paths

input circuit

tech lib

timing constraints

pass/fail
Circuit to Timing Graph

Circuit (hypergraph) → Timing (directed) graph

node → Arc
**Types of Loops**

**Sequential loops**  
(see the SMO model [IEEE-TCAD92] and its extensions)

**Combinational loops**  
(functional: see Edwards [DAC03], Jiang et al. [ICCAD04], Reidel [PhD04])  
(topological: see Hsu et al. [ICCD98] and this paper [TAU05])

Out of scope:  
loops due to xtalk
Need for Combinational Loops

\[
\begin{align*}
  f_1 &= x_1(x_2 + x_3) \\
  f_2 &= x_2 + x_1x_3 \\
  f_3 &= x_3(x_1 + x_2) \\
  f_4 &= x_1 + x_2x_3 \\
  f_5 &= x_2(x_1 + x_3) \\
  f_6 &= x_3 + x_1x_2
\end{align*}
\]

Rivest [IEEE-TC77]:
Given odd \( n > 1 \), \( n \) AND2 and \( n \) OR2:
- this circuit produces \( 2n \) distinct outputs \( f_i \).
- each output depends on all inputs \( x_j \).
- \#gates in cyclic = \( 2n \) (optimal).
- \#gates in acyclic = \( 3n-2 \).
Dealing with Combinational Loops

- **Strategy:** BREAK!
- **Means of breaking (Jouppi[TCAD87]):**
  - **static loop breaking:**
    - **method:** user or tool breaks loops literally.
    - minimizing #arcs or #nodes broken is NP-hard.
    - **pros:** no more loops in timing graph.
    - **cons:** can break critical paths.
  - **dynamic loop breaking:**
    - **method:** tool traces simple paths thru loops.
    - **pros:** no missing critical path.
    - **cons:** NP-hard
      - i.e., exponential time unless P=NP
- Both are supported by commercial STA tools and commonly used.
- **Our focus:** topological dynamic loop breaking
Static Loop Breaking

Breaking any arc on the loop breaks a valid path!
• combinational loop arcs: 1, 2, 3, and 4.
• if 2 or 3 is broken, red path is broken.
• if 1 or 4 is broken, blue path is broken.
Problem

Given:
• a digraph $G=(V,E)$
• arcs with weights (delays)
• source & sink nodes

Required:
• longest simple paths (& lengths) between every source and sink pair

Problem Complexity

• Three versions of the problem:
  
  - **G is acyclic: P**
    - $O(n+m)$ using LONGEST-PATHS-ACYCLIC (CLRS[1991])
  
  - **G is cyclic with non-positive loops: P**
    - $O(nm)$ using BELLMAN-FORD (CLRS[1991])
  
  - **G is cyclic with positive loops: NP-hard**
    - Hsu et al. [ICCD98]:
      - time = $O(c^2n^5+kn)$ & space = $O(cn^3+m+kn)$
      - (our paper corrects the original analyses.)
    
    - this paper [TAU05]:
      - time = $O(k(n+m))$ & space = $O(m+kn)$
    
    - Mehlhorn et al. [IPL02]:
      - time = $O(mn+n^2\log n)$ & space = $O(m+n)$
      - sets a path length to infinity if it hits a positive cycle!
    
  • Notation: $n=\#\text{nodes}$, $m=\#\text{arcs}$, $k=\#\text{paths}$, $c=\#\text{cycles}$

• Deciding among these versions, if necessary, also takes $O(nm)$ time using BELLMAN-FORD.
Simple Paths & Loops

Paths:
- simple
- not

Loops:
- simple
- not
Strong Connectivity

- **Strongly connected**
- **Not strongly connected**

Component graph is necessarily acyclic!

Entry and exit strongly connected component (SCC)
Algorithm by Hsu et al. [ICCD98]

**HSU-SUN-DU(G)**

1. find all cycles in G
2. construct a bipartite graph for each cycle by inserting an arc for every path prefix
3. combine bipartite graphs if they share nodes or arcs
4. replace each cycle by its bipartite graph
5. run LONGEST-PATHS-ACYCLIC(new, acyclic G)

- runs for all versions of the problem
- see fig.6 & next slide for bipartite graph handling
- called YSD in the paper
Example

input graph

cycle bipartite graphs

final graph after bipartite graph merging
Our Algorithm

**LONGEST-PATHS-CYCLIC(G)**

1. compute SCCs of G
2. mark entry / exit nodes of SCCs
3. for each SCC do
   1. enumerate all paths from entry to exit nodes
   2. replace SCC with its paths in G
4. run LONGEST-PATHS-ACYCLIC(new, acyclic G)

- runs for all versions of the problem
- See fig.3 for the pseudocode.
Example

input graph

component graph

entry = 2

exit = 4

final graph
A Real Example SCC
## Test Suite: Industrial Designs

<table>
<thead>
<tr>
<th>Name</th>
<th>#pins</th>
<th>#cells</th>
<th>#SCCs</th>
<th>Selected SCCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>451,780</td>
<td>153,835</td>
<td>32</td>
<td>C1</td>
</tr>
<tr>
<td>D2</td>
<td>8,920,245</td>
<td>1,890,597</td>
<td>354</td>
<td>C2</td>
</tr>
<tr>
<td>D3</td>
<td>1,653,608</td>
<td>242,535</td>
<td>45</td>
<td>C3,C5</td>
</tr>
<tr>
<td>D4</td>
<td>1,292</td>
<td>336</td>
<td>10</td>
<td>C4</td>
</tr>
<tr>
<td>D5</td>
<td>6,205</td>
<td>1,354</td>
<td>1</td>
<td>C6</td>
</tr>
</tbody>
</table>
# Runtime Results

<table>
<thead>
<tr>
<th>Name</th>
<th># nodes</th>
<th># arcs</th>
<th># entry nodes</th>
<th># exit nodes</th>
<th># paths</th>
<th>total path length</th>
<th>time bound (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>42</td>
<td>50</td>
<td>22</td>
<td>11</td>
<td>45</td>
<td>176</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>84</td>
<td>116</td>
<td>34</td>
<td>1</td>
<td>186</td>
<td>6390</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>118</td>
<td>136</td>
<td>11</td>
<td>14</td>
<td>20</td>
<td>134</td>
<td>1</td>
</tr>
<tr>
<td>C4</td>
<td>172</td>
<td>210</td>
<td>15</td>
<td>17</td>
<td>45</td>
<td>402</td>
<td>1</td>
</tr>
<tr>
<td>C5</td>
<td>437</td>
<td>546</td>
<td>41</td>
<td>10</td>
<td>241M</td>
<td>1.2G</td>
<td>512</td>
</tr>
<tr>
<td>C6</td>
<td>996</td>
<td>1184</td>
<td>99</td>
<td>118</td>
<td>194</td>
<td>1452</td>
<td>1</td>
</tr>
</tbody>
</table>
Conclusions & Future Work

- Industrial STA tools need to handle combinational loops.
- Problem of computing critical paths in presence of such loops is NP-hard.
- We have proposed a simple and efficient algorithm.
- We have validated the algorithm on some industrial designs.
- Future work:
  - use output as input to static loop breaking
  - combine with functional approach