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

Ali Dasdan and **Santanu Kolay**

February 28, 2004

PrimeTime

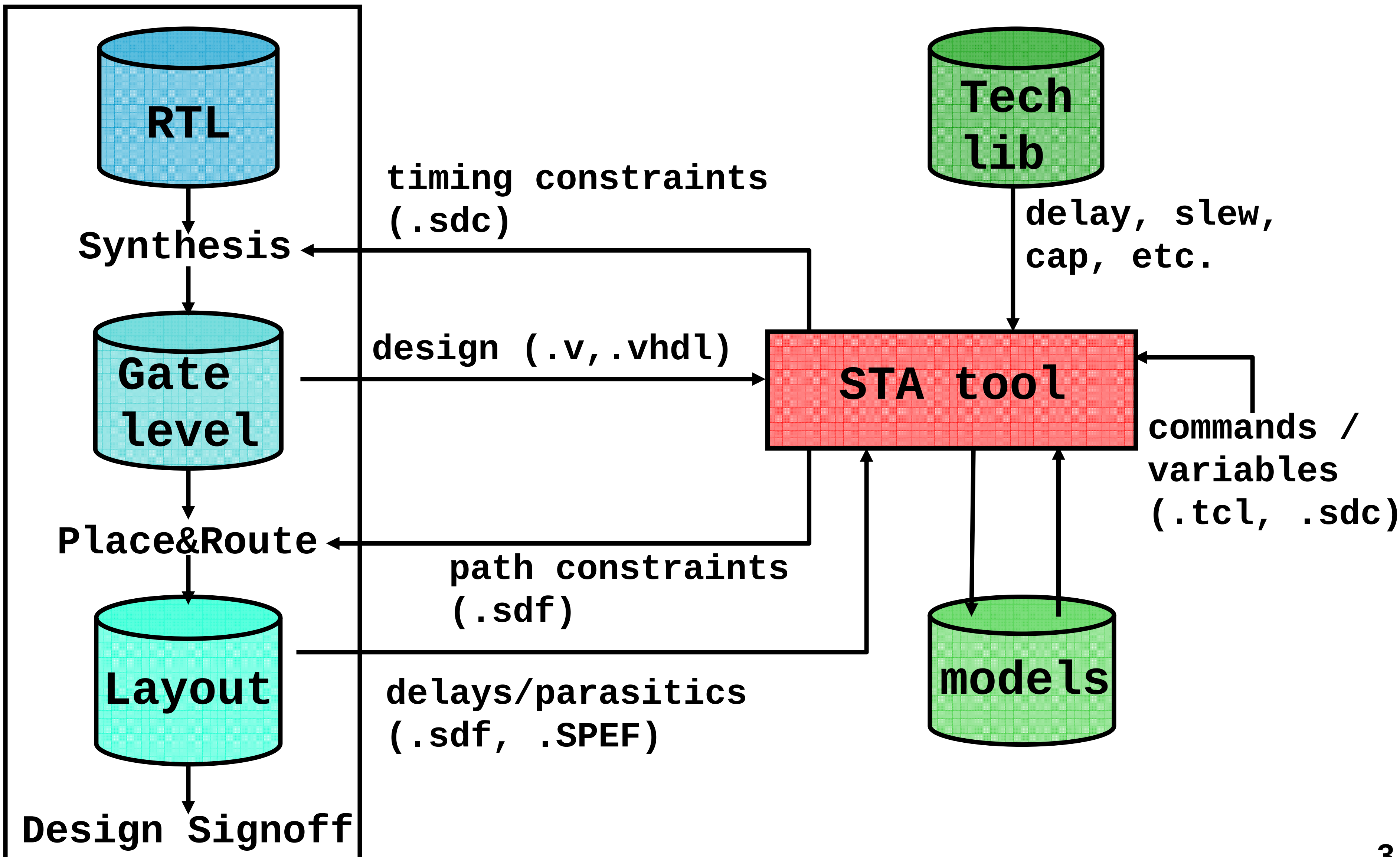
Synopsys, Inc.

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



Simple STA Tool

input circuit

tech lib

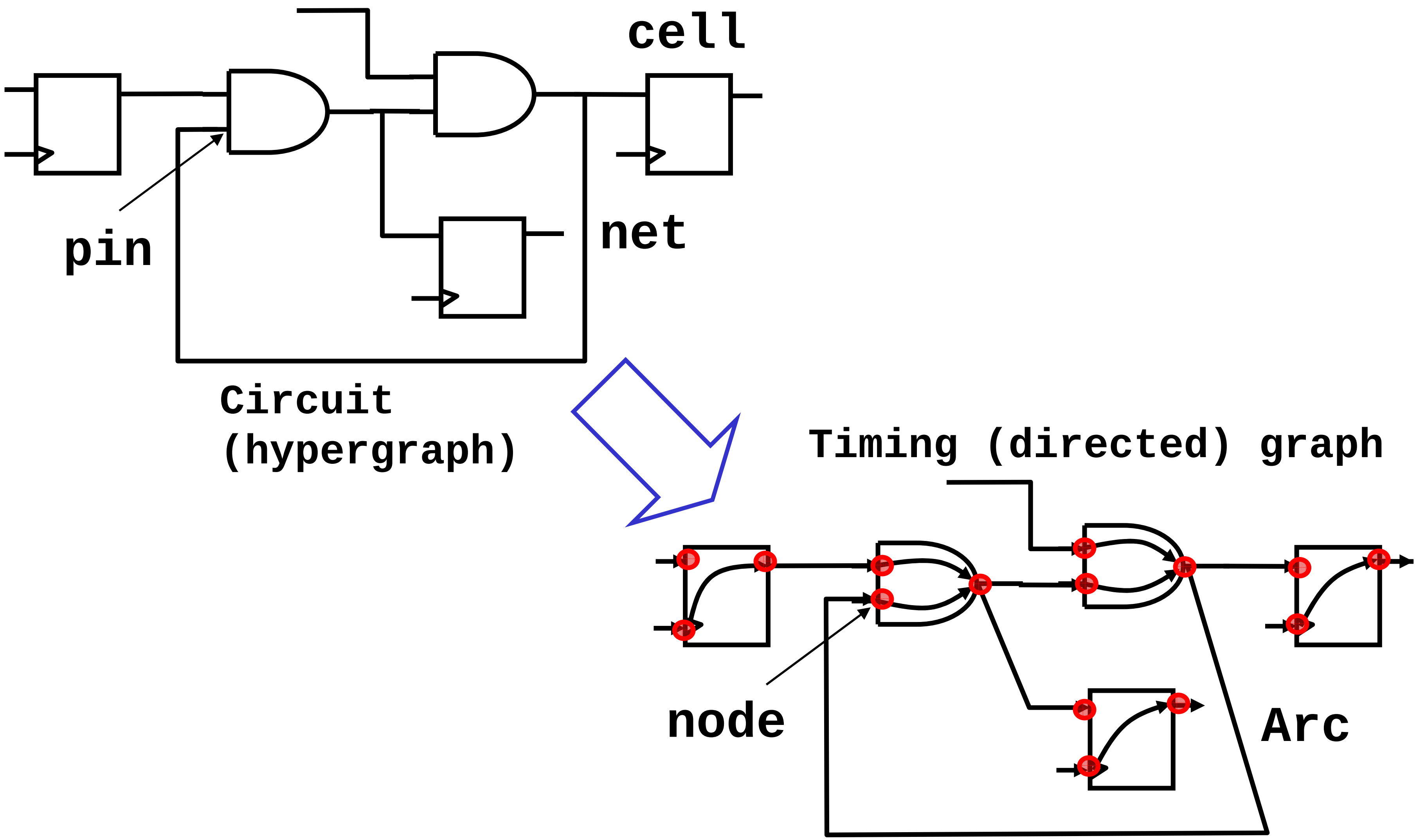
timing constraints

STA tool:

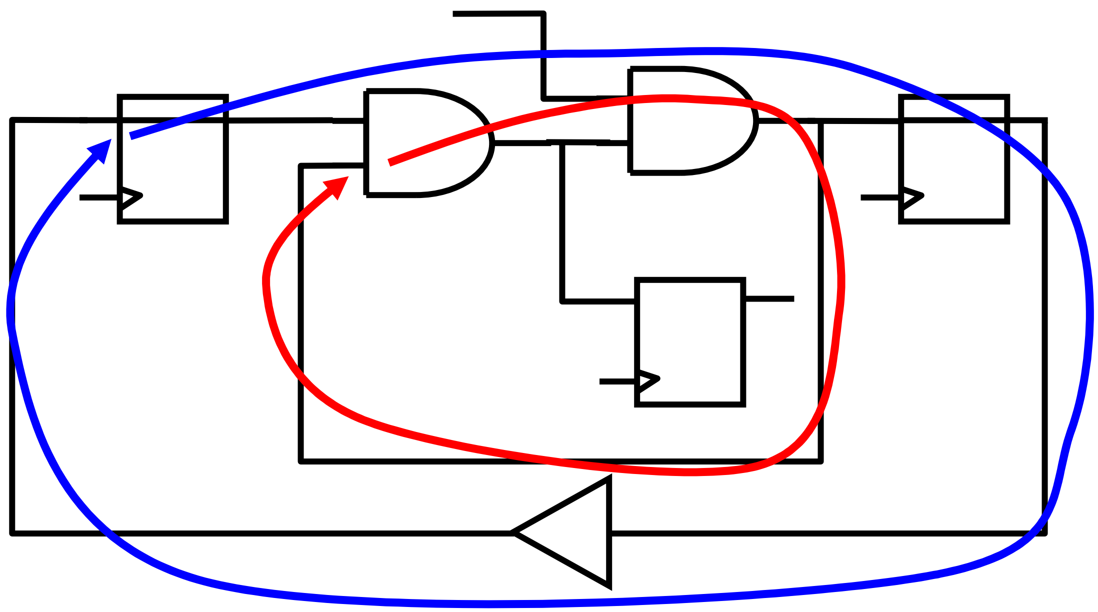
- -circuit to timing graph
- -compute delays & slews
- -compute arrival times (*compute longest paths*)
- -compute required times
- -compute slacks
- -report violating paths

pass/fail

Circuit to Timing Graph



Types of Loops

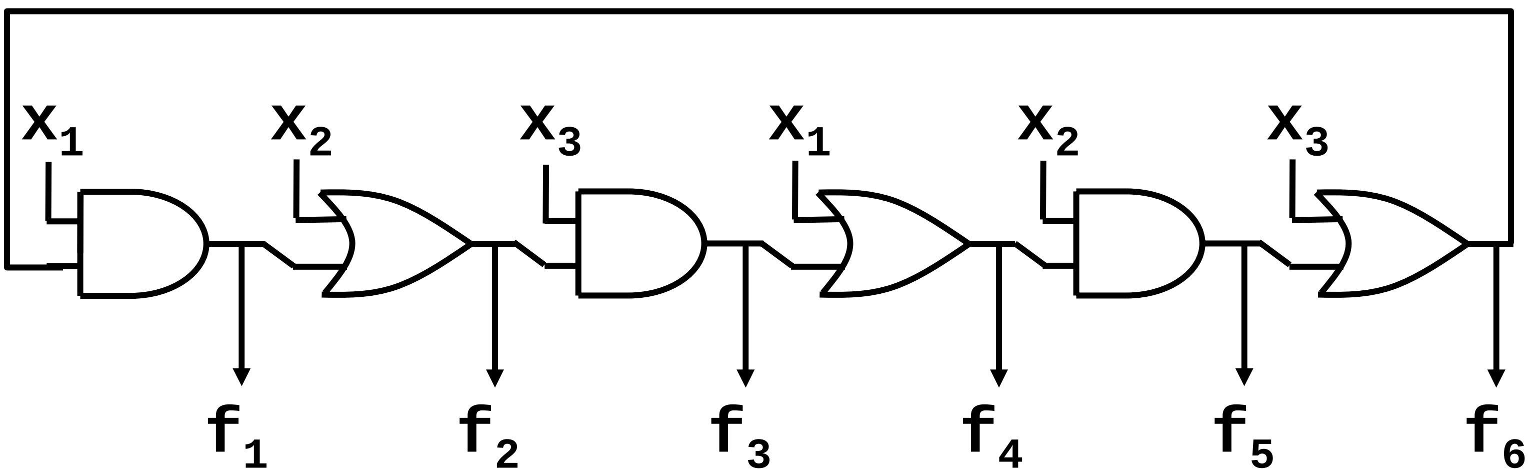


Sequential loops
(see the SMO model[IEEE-TCAD92]
and its extensions)

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



$f_1=x_1(x_2+x_3)$
$f_2=x_2+x_1x_3$
$f_3=x_3(x_1+x_2)$
$f_4=x_1+x_2x_3$
$f_5=x_2(x_1+x_3)$
$f_6=x_3+x_1x_2$

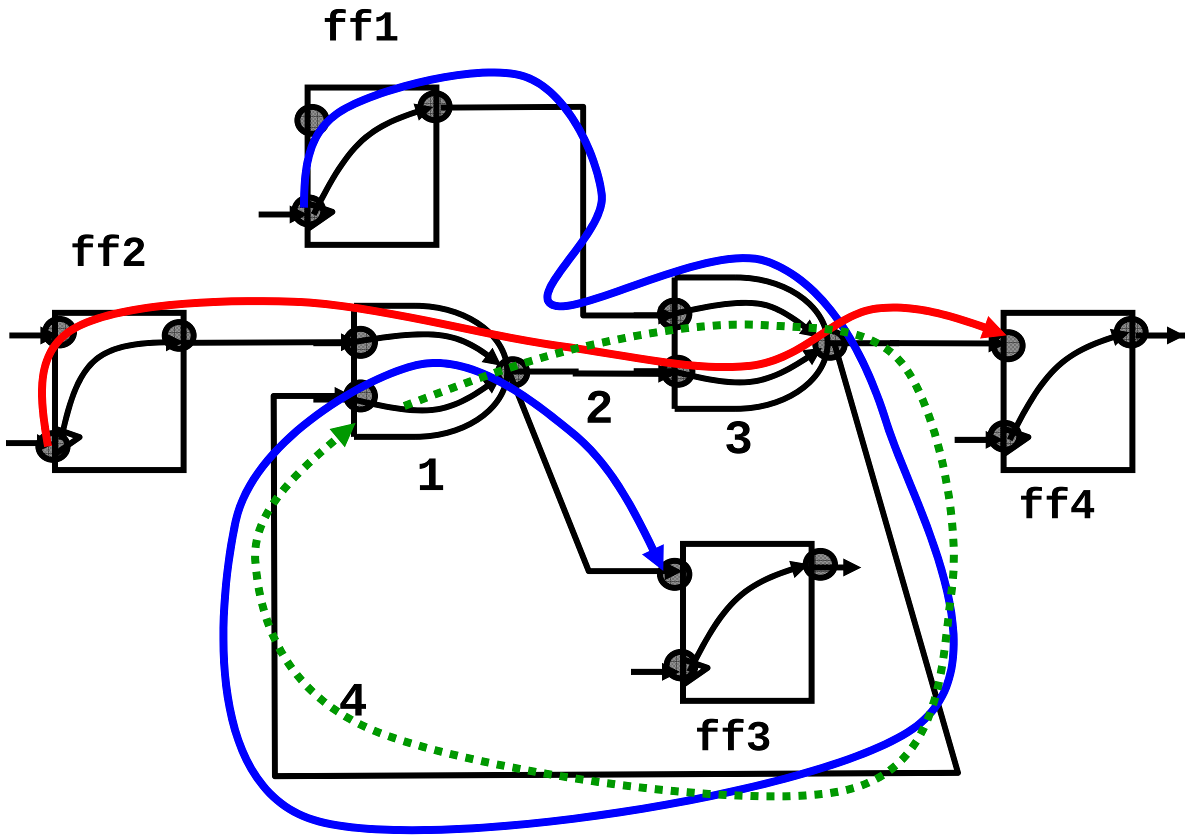
Rivest[IEEE-TC77]:
Given odd n > 1, n AND2 and n OR2:

- **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

[ref: See LONGEST PATH in Garey & Johnson’s NP-Completeness book, 1979.]

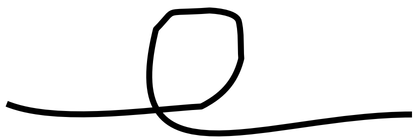
Problem Complexity

- **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^2logn) & space = O(m+n)**
      - **sets a path length to infinity if it hits a positive cycle!**
    - **Notation: n=#nodes, m=#arcs, k=#paths, c=#cycles**
- **Deciding among these versions, if necessary, also takes O(nm) time using BELLMAN-FORD.**

Simple Paths & Loops

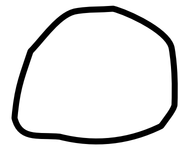
Paths:

simple

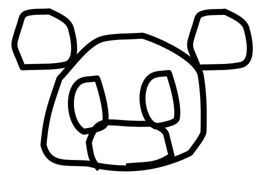


not

Loops:

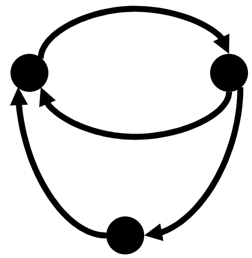


simple

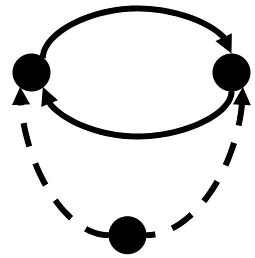


not

Strong Connectivity

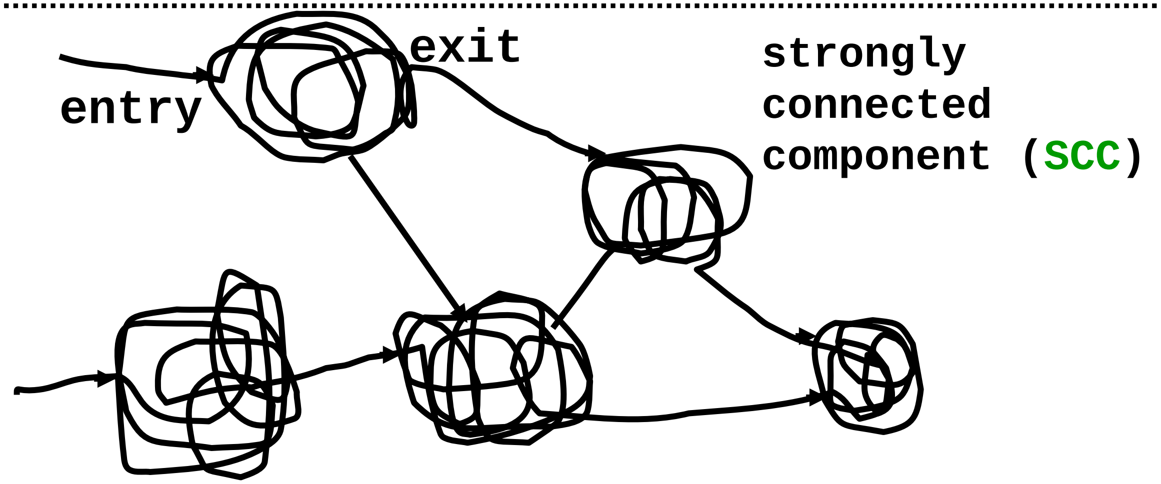


strongly connected



not strongly connected

component graph is necessarily acyclic!



component graph

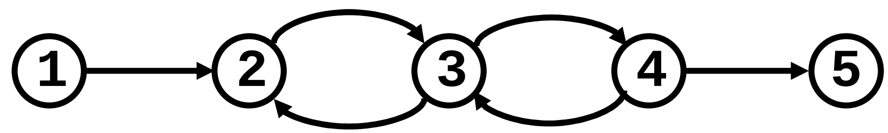
Algorithm by Hsu et al.[ICCD98]

```
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)
```

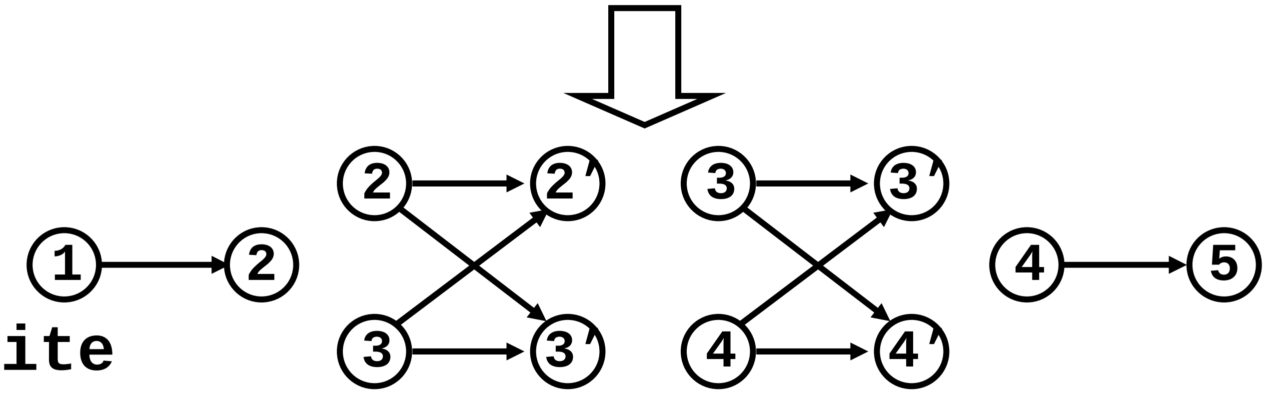
- runs for all versions of the problem
- see fig.6 & next slide for bipartite graph handling
- called YSD in the paper

Example

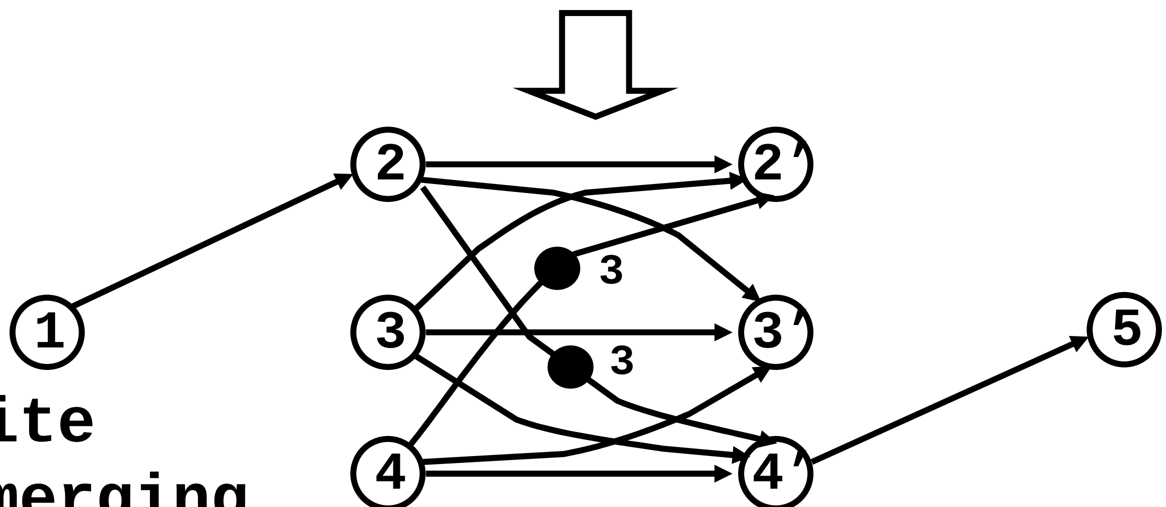
input graph



cycle bipartite graphs



final graph after bipartite graph merging

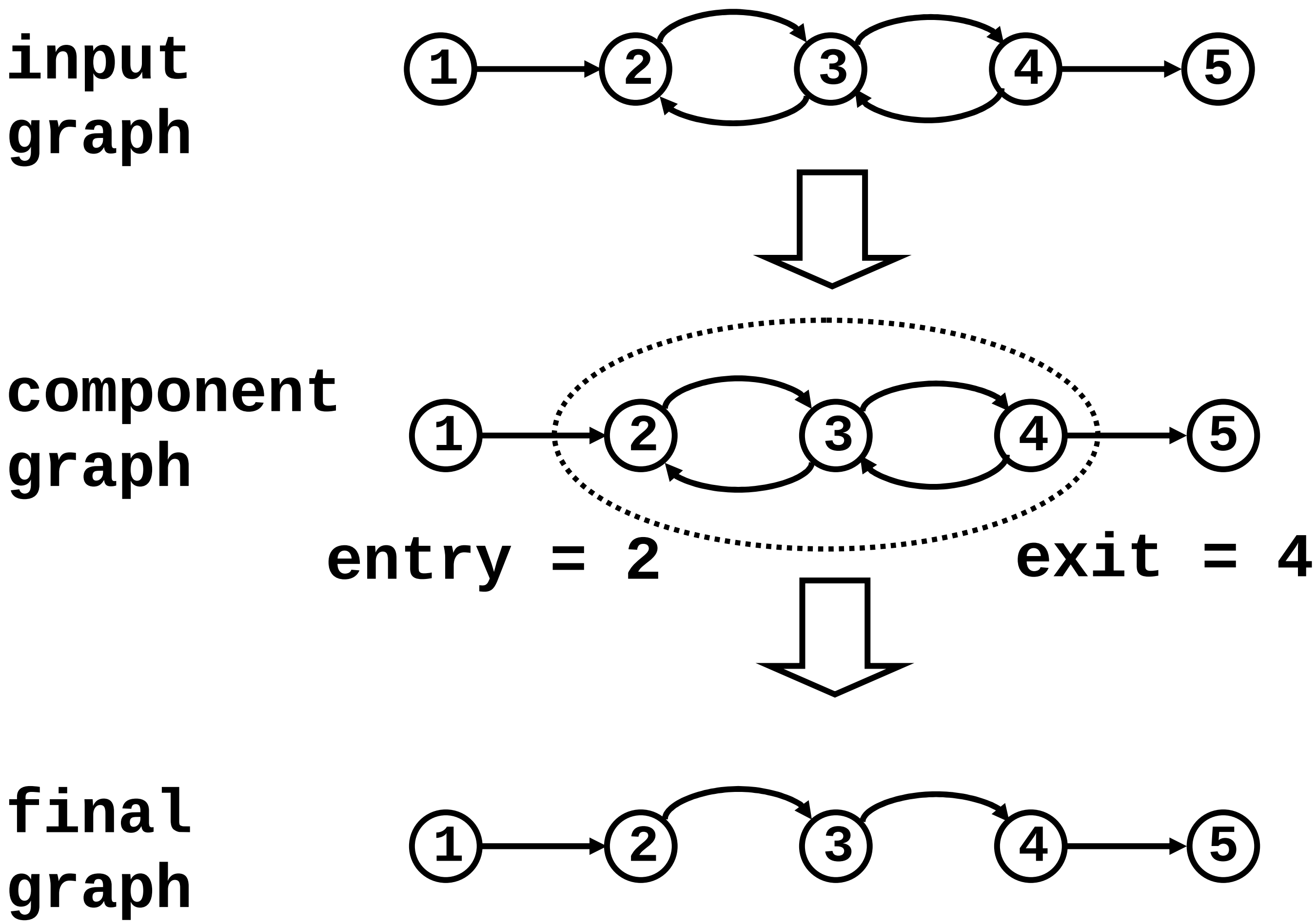


Our Algorithm

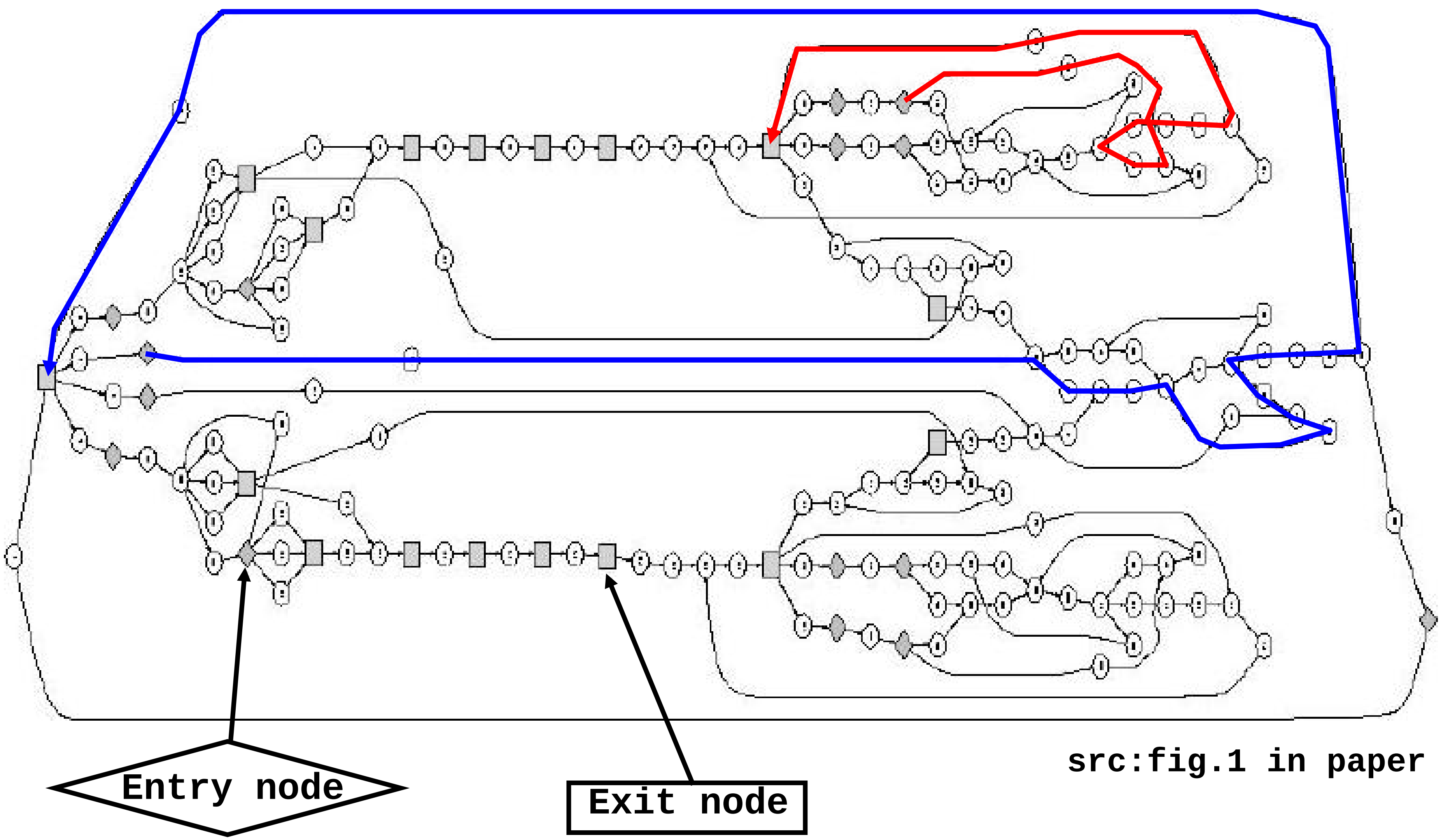
```
LONGEST-PATHS-CYCLIC(G)
   1. compute SCCs of G
   2. mark entry / exit nodes of SCCs
   3. for each SCC do
      1. enumerate all paths from entry to exit nodes
      2. replace SCC with its paths in G
   4. run LONGEST-PATHS-ACYCLIC(new, acyclic G)
```

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

Example


input graph
1
2
3
4
5
component graph
1
2
3
4
5
entry = 2
exit = 4
final graph
1
2
3
4
5

A Real Example SCC



src:fig.1 in paper

Test Suite: Industrial Designs

Name	#pins	#cells	#SCCs	Selected SCCs
D1	451,780	153,835	32	C1
D2	8,920,245	1,890,597	354	C2
D3	1,653,608	242,535	45	C3,C5
D4	1,292	336	10	C4
D5	6,205	1,354	1	C6

Runtime Results

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

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