Performance Optimization of Logic Circuits

Outline:

- Definitions and problem statement
- Overview of techniques
  - ◇ Tree-height reduction
  - ◇ Generalized bypass transform
  - ◇ Generalized cofactor transform
  - ◇ Partial collapsing
- Comparison of techniques

Timing optimization:

Factors determining delay of circuit:

- underlying circuit technology
- logical design of circuit
  - ◇ length of computation paths
  - ◇ false paths

Problem statement - Given:

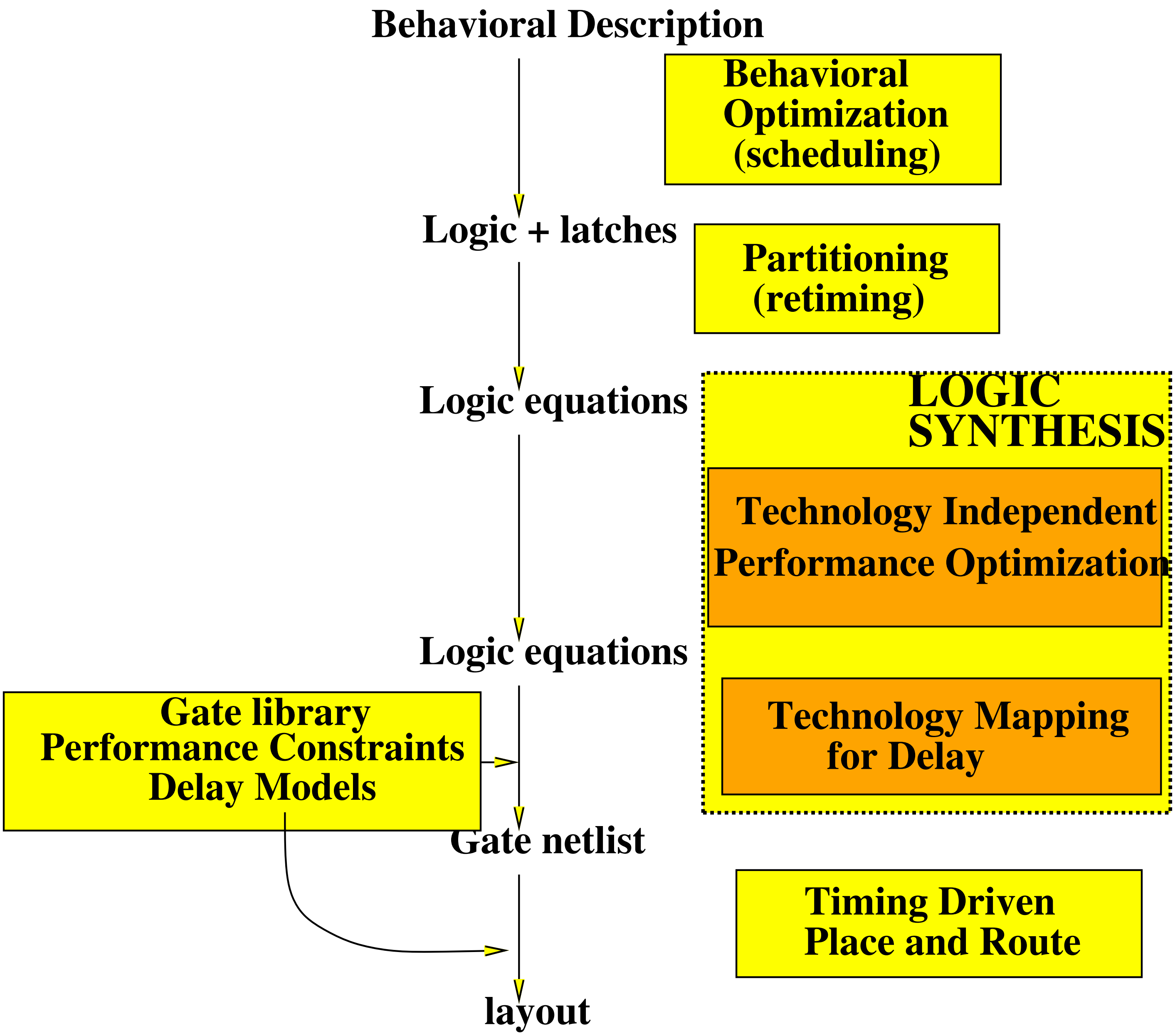
- initial circuit function description
- library of primitive functions
- performance constraints

generate an implementation of the circuit using the primitive functions,

such that

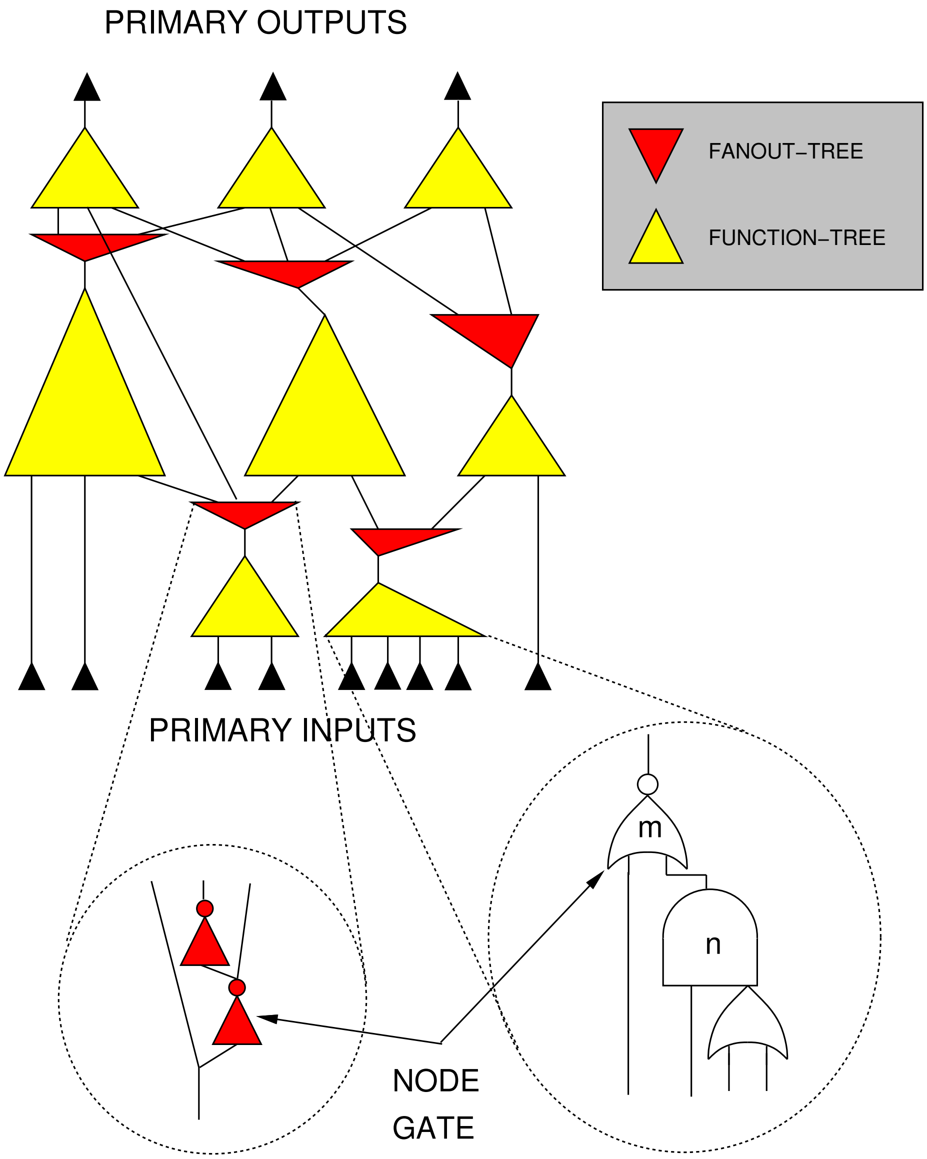
1. performance constraints are met
2. within that, circuit area is minimized

Current Design Process Timing Optimization at All Levels



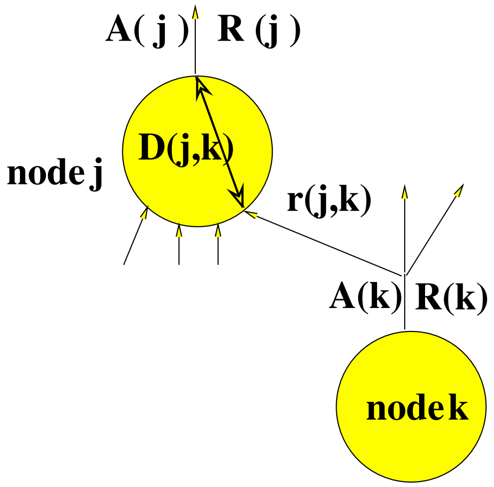
Technology Mapping for Delay

combinational circuit



Estimating Delays

$A(j)$: arrival time of signal j
$R(k)$: "deadline" for signal k
$S(k)$: slack of signal k
$D(j,k)$: delay of node j from input k



$$A(j) = \max_{k \in FI(j)} [A(k) + D(j,k)]$$

$$r(j,k) = R(j) - D(j,k) \qquad \forall k \in FI(j)$$

$$R(k) = \min_{j \in FO(k)} [r(j,k)]$$

$$S(k) = R(k) - A(k)$$

Overview of Solutions

- Circuit structure
  - ◇ rescheduling operations to reduce time of computation

- Implementation of function trees
  - ◇ selection of gates from library
    [Keutzer 88]
    [Rudell *et al.* 88]
    [Touati *et al.* 90]

- Implementation of fanout trees
  - ◇ capacitive load redistribution
    [Berman *et al.* 89]
    [Touati *et al.* 90]
    [Singh, ASV 90]

- Our focus is on **circuit restructuring**

Technology-independent Restructuring

Circuit Restructuring Approaches:

Local:

- Mimic optimization technique in adders
  - ◇ carry-lookahead (tree-height reduction)
  - ◇ conditional sum (GST transformation)
  - ◇ carry bypass (GBX transformation)

Global:

- Reduce depth of entire circuit
  - ◇ partial collapsing
  - ◇ Boolean simplification

Restructuring algorithm (local):

while (circuit timing improves) *do*

  Select regions to transform

  Transform the selected region

end

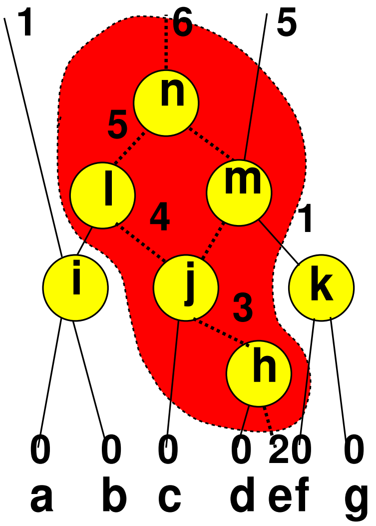
Restructuring methods

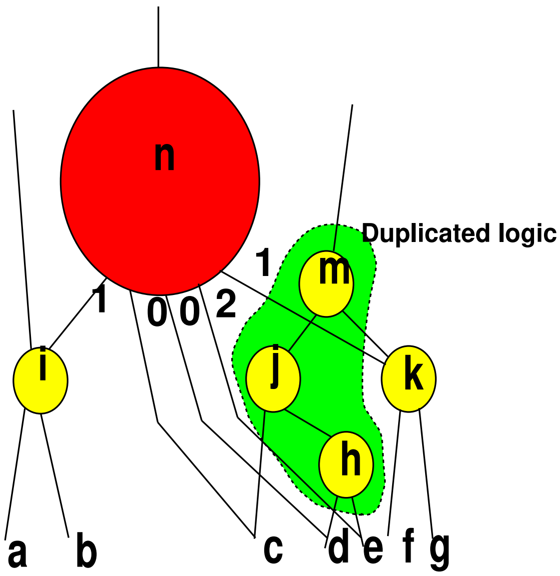
Performance measured by levels, sensitizable paths, technology dependent delays

- Level based optimization
  - ◇ Tree-height reduction
    [Singh *et al.* 88]
  - ◇ Partial collapsing and simplification
    [Touati *et al.* 91]
  - ◇ Generalized select transform
    [Berman *et al.* 90]

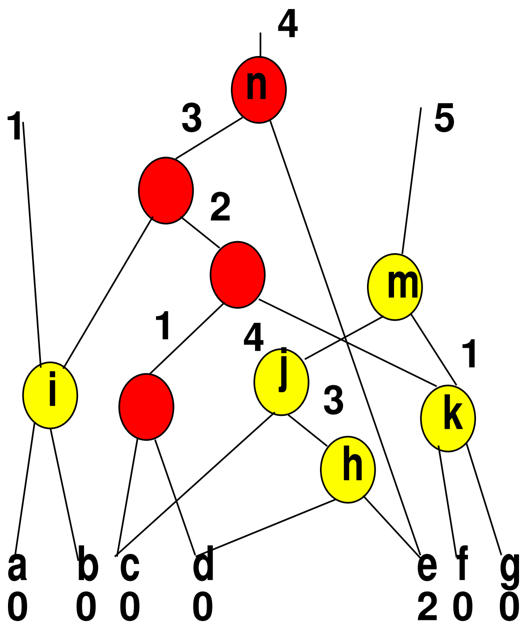
- Sensitizable paths
  - ◇ Generalized bypass transform
    [McGeer *et al.* 91]

Restructuring for delay: path reduction





Collapsing

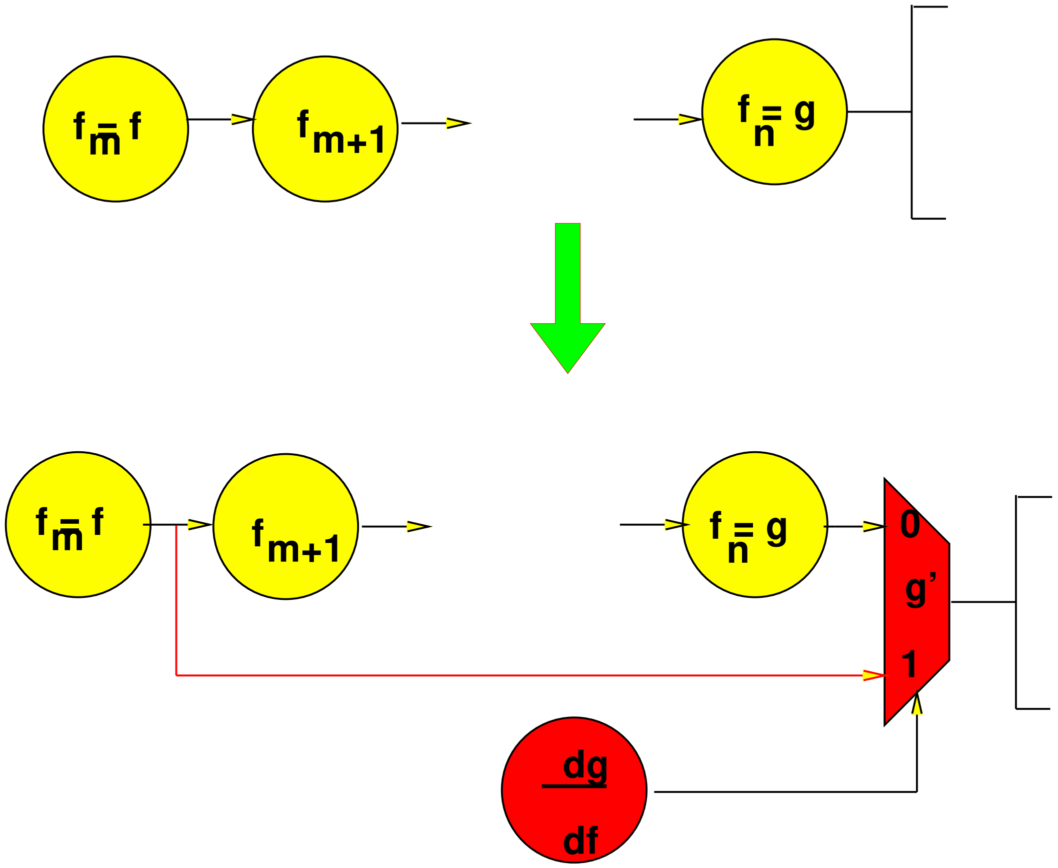


Timing Driven Decomposition

[Singh *et al.* 88]

Generalized bypass transform

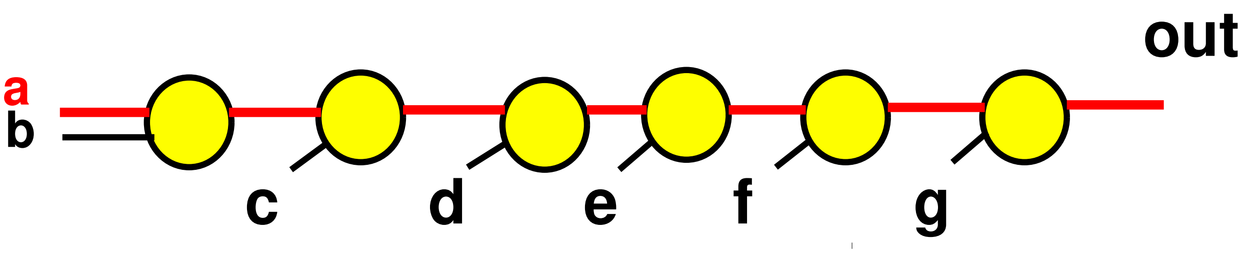
- Make critical path false
  ⇒ speed up circuit
- Bypass logic of critical path

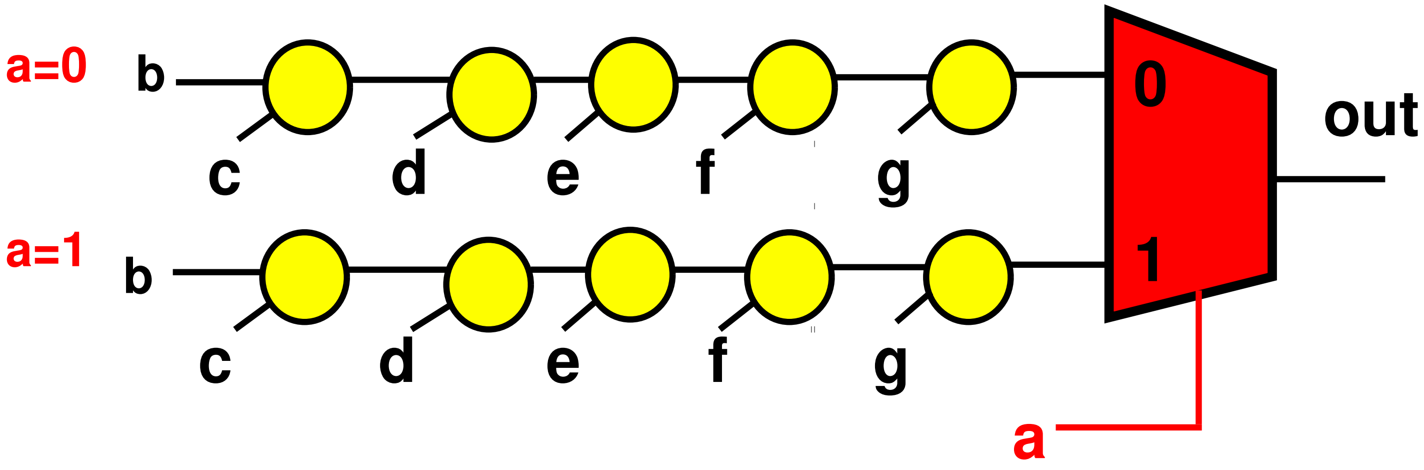


[McGeer *et al.* 91]

Generalized Select Transform

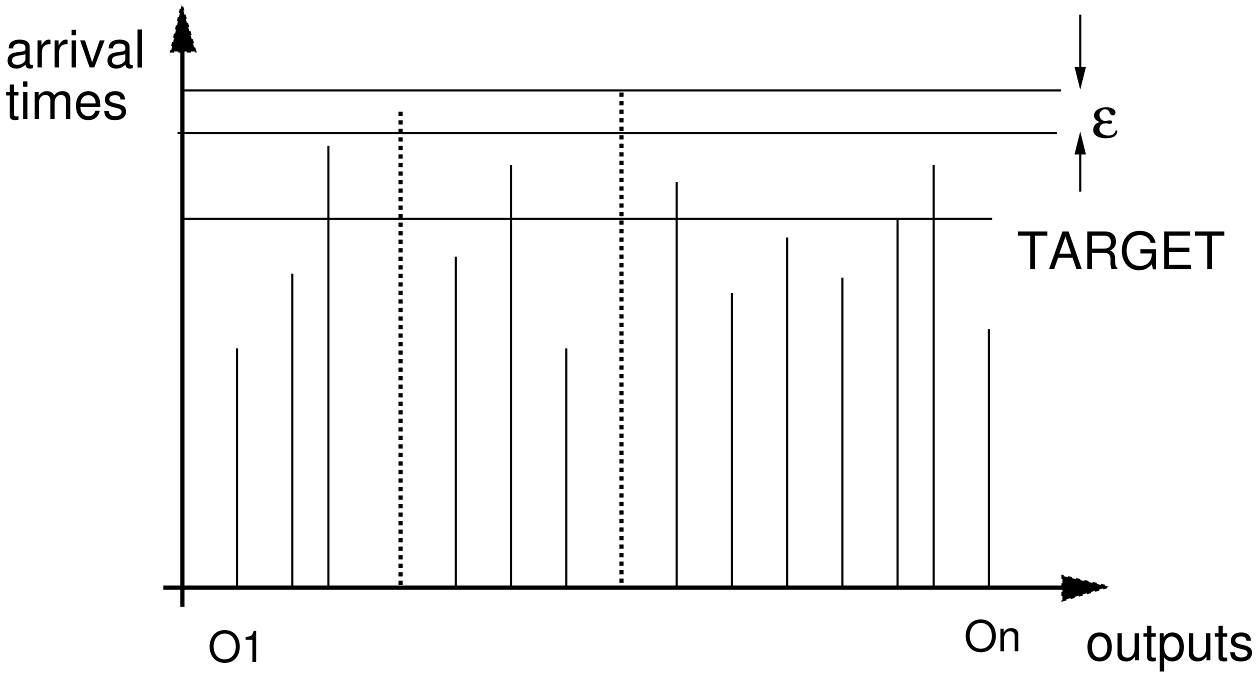
- Late signal feeds multiplexor





[Berman *et al.* 90]

First Approach - Tree Height Reduction

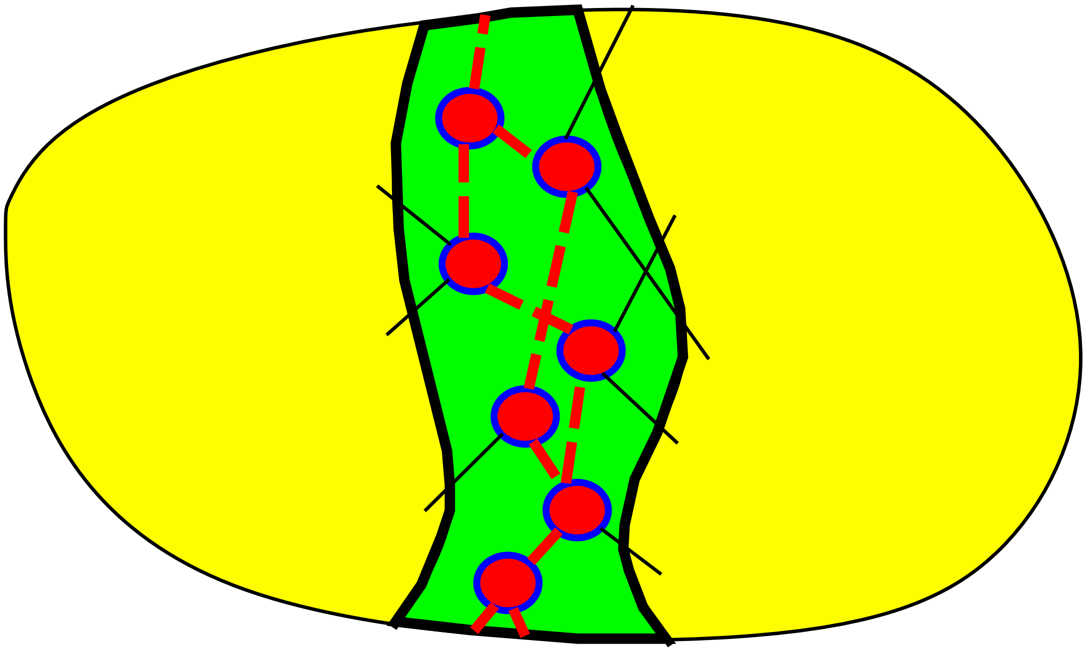


- Look at most critical sections

- All nodes with slack within ϵ of the most critical signal belong to the **ϵ-critical circuit**

Where to Restructure?

- ϵ-critical circuit

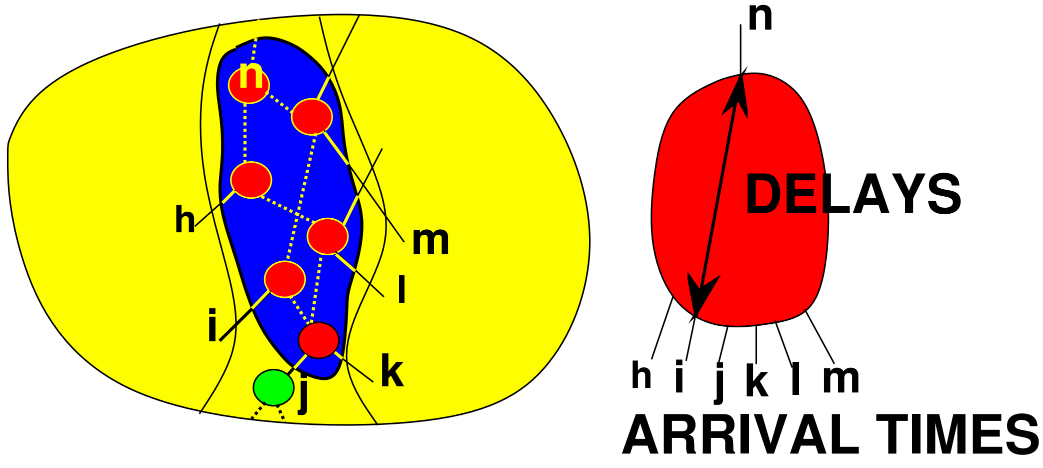


- To improve circuit delay, sufficient
  to improve delay at nodes on
  a separator-set of ϵ-critical circuit
- Put weight w_i on each node i reflecting
  area/delay trade-off
- Each node should be able to achieve at least ϵ
  delay reduction
- Find minimum weighted separator set.

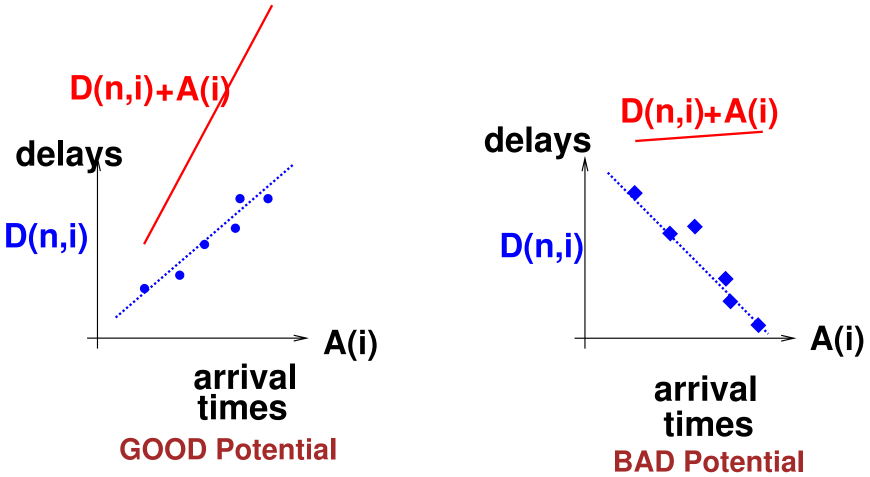
Potential for Restructuring at a Node

Area-Delay Trade-off

- Based on delays in the circuit



- Based on delays in the circuit



Problems with first approach

If each node in separator set is reduced in delay by at least ϵ, circuit delay decreases by at least ϵ (not accounting for false paths.)

- Only a sufficient condition
  - ◇ leads to large area increases
- How to choose ϵ apriori
- Weighting process (area/delay tradeoff) is ad-hoc

Second Approach: Idea

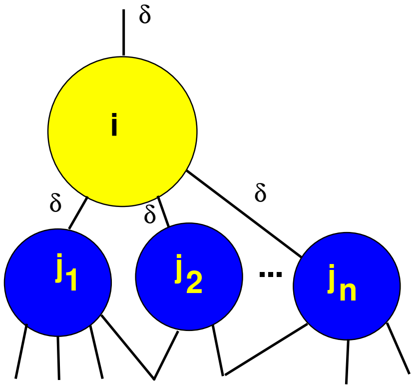
- First evaluate all possible transforms,

  then select a subset to apply to achieve

  most improvement

Selecting a region - Lower bound technique

$\delta(i)$: Amount arrival time can be reduced at node i (*computed recursively from inputs*).
$D(i)$: Delay saving at node i
$S(i,j)$: Current slack from fanin j to node i



Theorem: *Assuming that only one delay reduction transformation is made along a path from input to output, and that $D(i)$ is the delay savings at node i if a transformation is applied at i, the arrival time at i can be reduced at least by:*

$$\delta(i) = \begin{cases} 0 & \text{if } i \text{ is a PI} \\ \max[D(i), \min\limits_{j \in FI(i)} \{S(i,j) + \delta(j)\} \\ \qquad - \min\limits_{k \in FI(i)} \{S(i,k)\}] & \text{otherwise} \end{cases}$$

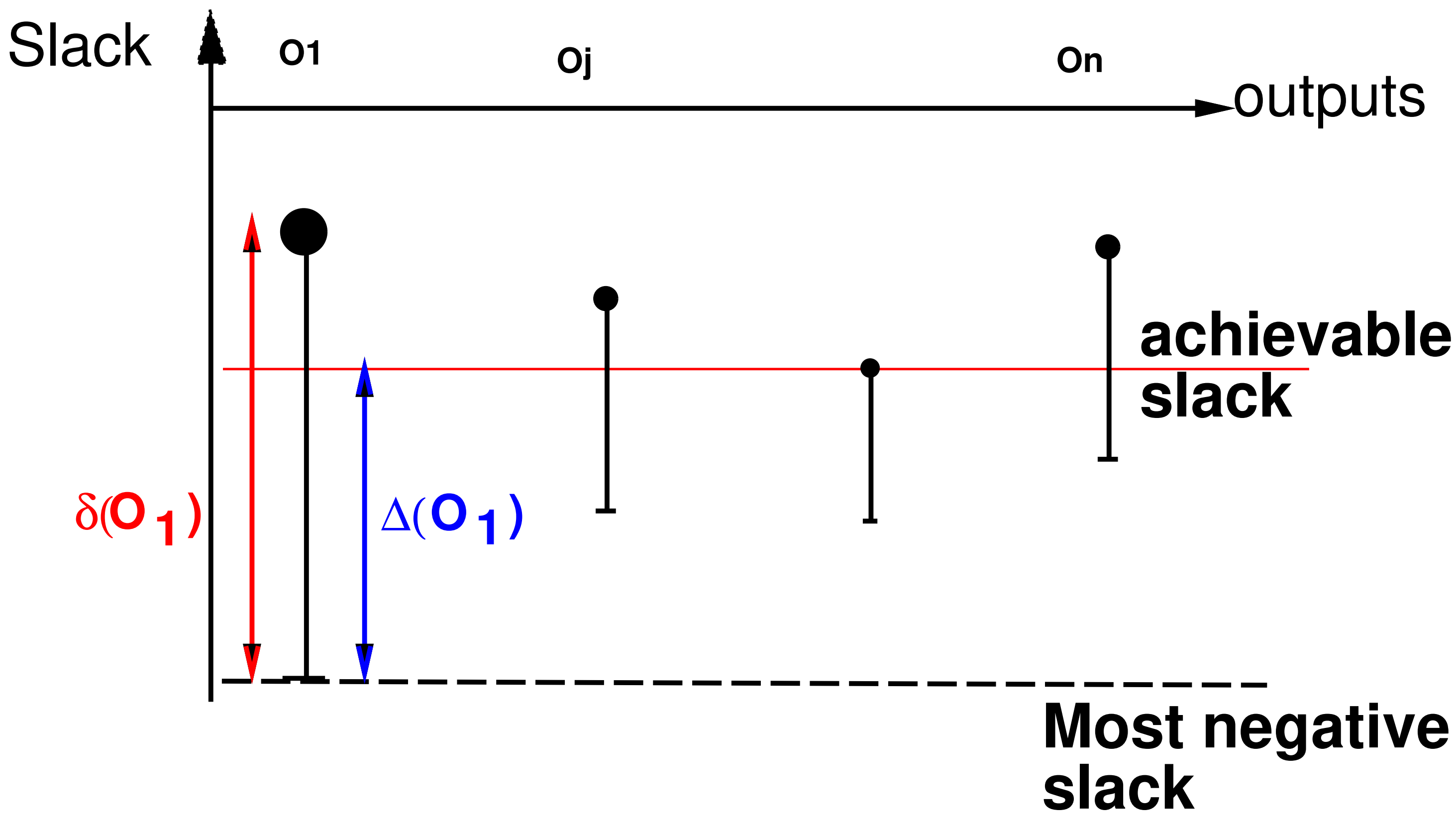
Proof: Uses savings = new arrival time - old arrival time.

Lower bound over multi-outputs

Slack s^* that can be achieved:

$$s^* = \min_{i=1,\ldots,k} \{s(O_i) + \delta(O_i)\}$$

$$\Delta(O_i) = s^* - s(O_i)$$



Choosing a subset to realize lower bound

Once required improvements are known, propagate information backwards

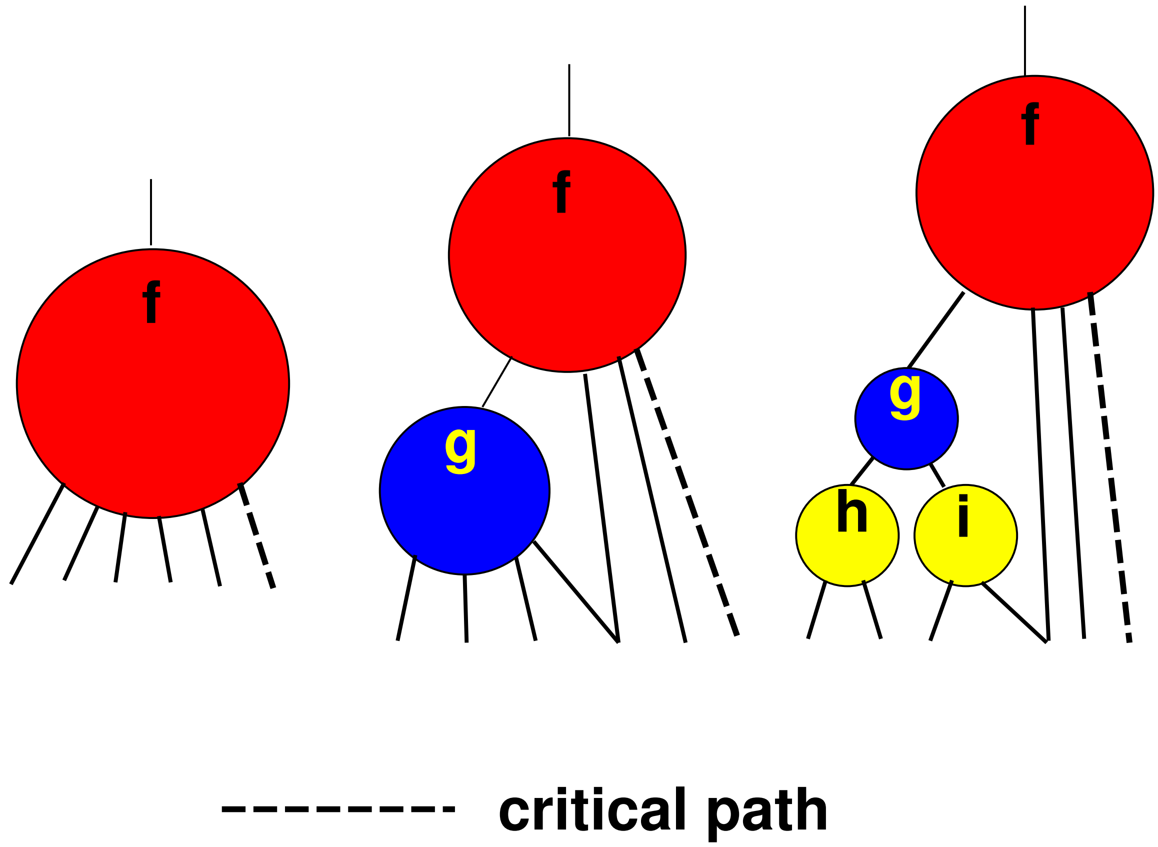
- Find a selection of nodes such that applying transform at each node yields
  - minimize area increase to achieve bound
  - minimize number of nodes in separator set
- Currently solved by shortest path in BDD representing all satisfying selections
  - one variable for each node
  - weight = 0 if not selected
    = area increase if selected

[Singh 92]

Decomposing Selected Nodes for Delay Reduction

- Decomposition based on input arrival times
- Prefer sub-expressions which have
  - ◇ large area saving
  - ◇ early arriving signals

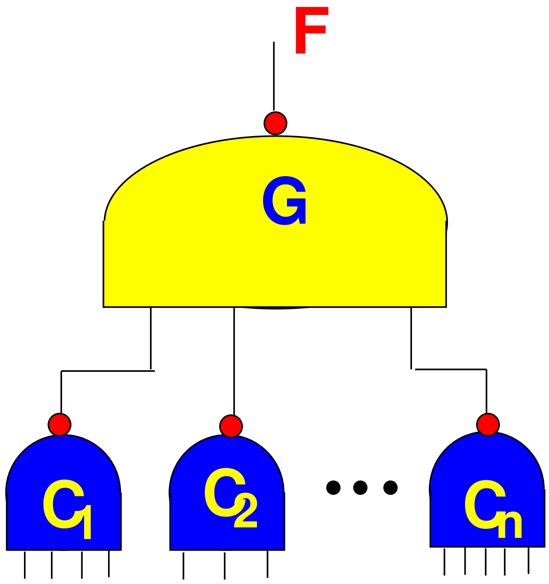
Top-down decomposition



Decomposing AND-OR Functions

$$F = C_1 + C_2 + \cdots + C_n$$

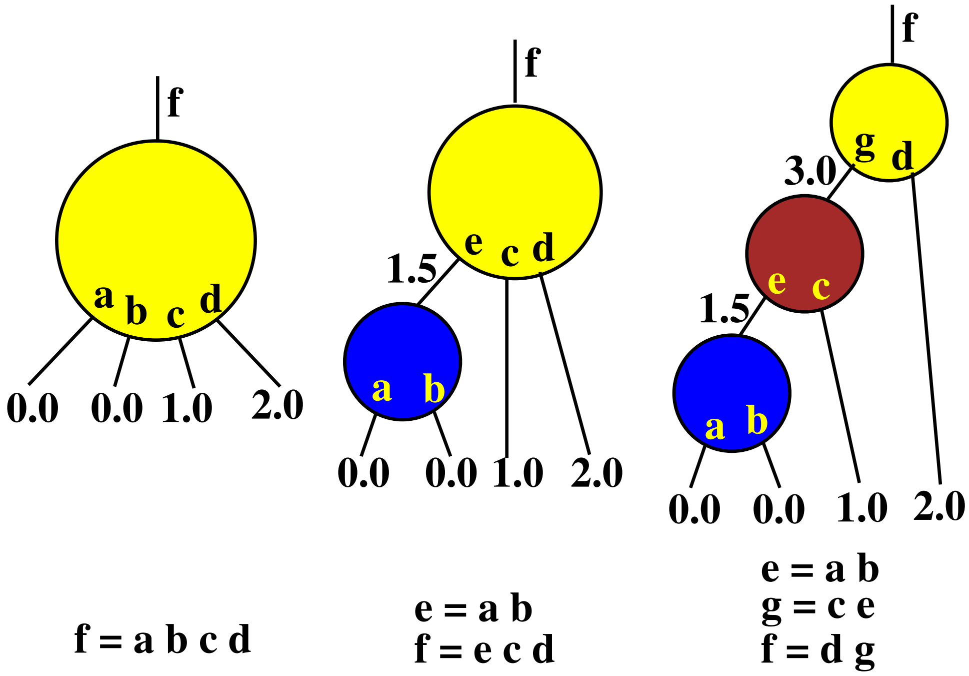
Represented as a NAND-NAND circuit.



Hence need optimum AND decomposition:

For each cube C_i in F
  AND_decompose(C_i)
AND_decompose(G)

Decomposing a Single Cube



AND_decomp(F)
    while ($|F| > 2$) *begin*
      combine two earliest inputs, $g = l_1.l_2$
      compute delay at output of g (new arrival time)
      substitute new node,g, into F
    end

Note: Each time substitute g into F, new input's arrival time is known. Hence at all times, for all inputs, arrival times are known.

Optimality of Method

Monotone Speedup Property:

Lemma 1: For a tree-decomposition of x into 2-input NAND gates or inverters, if $A_{b'} \leq A_b$ then $A_{x'} \leq A_x$

Not true in general for non-tree circuits.

Lemma 2: Let f be an n-input function, $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_n\}$ be two sets of signals such that $A_{x_i} \leq A_{y_i}$ for $i = 1, 2, ..., n$. Then $A_{T(f(x))} \leq A_{T(f(y))}$ where $T(f(x))$ is any tree implementing $f(x)$.

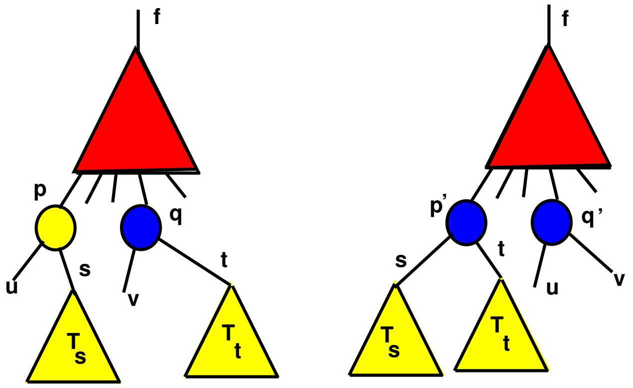
Proving optimality Single cube decomposition. Let $A_{D_f^o}$ = delay of optimum delay tree-decomposition of f.

Theorem: Let f be an $(n+2)$ input NAND gate with u and v being the earliest arriving signals, *i.e.* $f =$ NAND$_{n+2}(X, u, v)$ and $A_u \leq A_v \leq A_{x_i}$, for $i = 1, 2, ..., n$. Let $w = uv$ and $g =$ NAND$_{n+1}(X, w)$. Then, $A_{D_f^o} = A_{D_g^o}$, *i.e.* there is an optimum decomposition of f with u and v being the inputs to a NAND gate.

Proof: By case analysis (next slide).

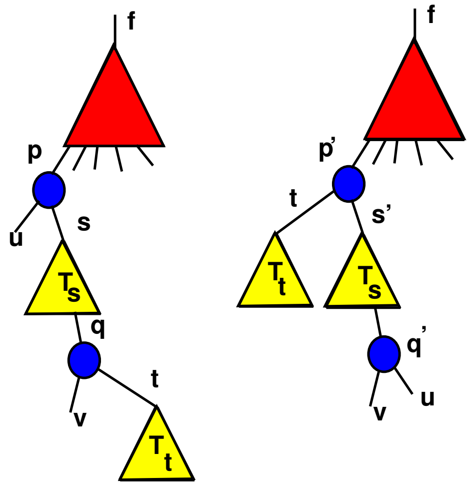
Proving optimality

Case 1:



if $A_t \leq A_s$ then $A_{p'} \leq A_p$, $A_{q'} \leq A_q$.

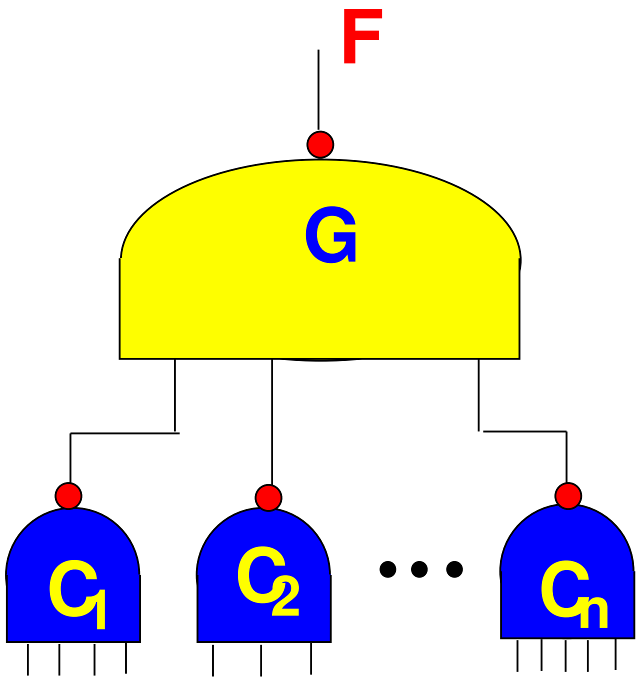
Case 2:



Slow down u to t; does not change speed of left circuit. Now switch u and t. Still the speed does not change. Now speed u back up to where it was. q' and hence s' speed up. Since $A_t \leq A_s$, speeding up $A_{s'}$ speeds up p'.

$\Rightarrow$ NAND decomposition is optimum

Proving optimality

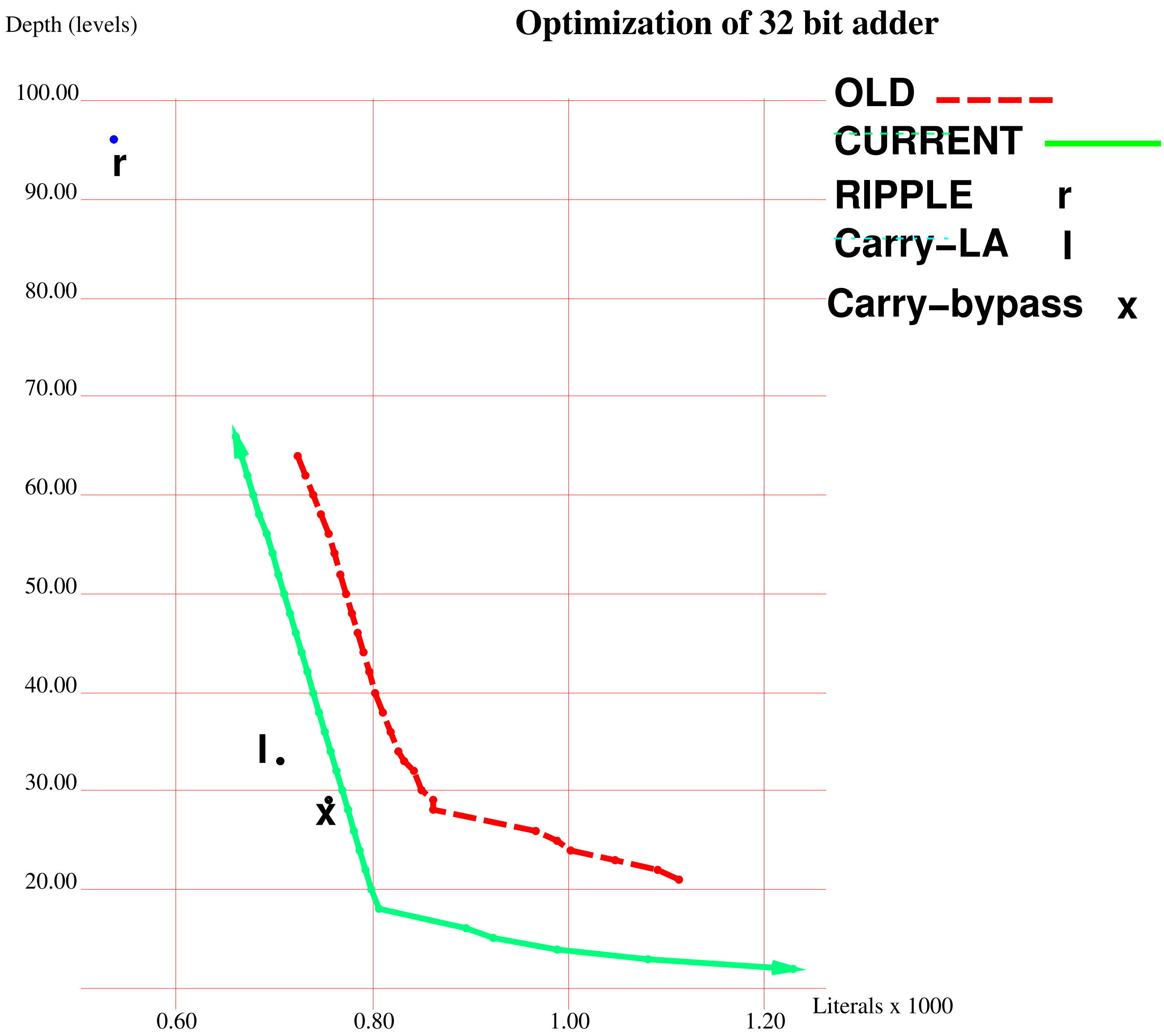


Disjoint SOP decomposition,

Theorem: Let f be the sum of orthogonal (disjoint support) cubes, $f = c_1 + c_2 + .. + c_n$ and $c_i \perp c_j$ for all $i \neq j$. Let $s_i = \overline{c}_i$ for all i and $g = \overline{s_1 s_2 ... s_n}$. Then, $(D^o_g, D^o_{s_1}, D^o_{s_2}, ..., D^o_{s_n})$ is an optimum decomposition of f, *i.e.* $A_{D^o_f} = A_{D^o_g}$.

Proof: First show that each optimum decomposition of f consists of optimum decomposition for each component cube c_i, and the single OR gate. Result follows.

Adder tradeoff



Tree height reduction results

	Initial		Final	
File	lits-fac	level	lits-fac	unit-fan
C1355	812	20	1156	15
C1908	784	31	963	22
C2670	1188	32	1276	16
C3540	2053	42	2203	30
C6288	4443	92	5389	71
C7552	3569	41	3612	25
ampbpreg	1368	19	1581	14
ampbsm	1105	20	1142	11
amppint2	893	32	928	15
ampxhdl	527	18	571	11
cps	1902	22	1928	12
dalu	1570	23	1705	13
des	6051	26	6249	15
duke2	676	23	700	12
ex1010	3942	19	3936	14
ex4	839	14	979	9
k2	1687	21	1705	16
misex3c	796	47	847	22
rot	1050	25	1157	14
spla	980	21	1014	12
t481	1183	23	1189	16

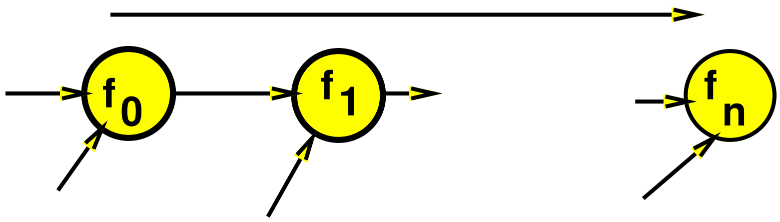
unit fan-out delay model $= \alpha + \beta|\text{fanout}|$

Generalized Bypass Tranform (GBX)

New Approach:

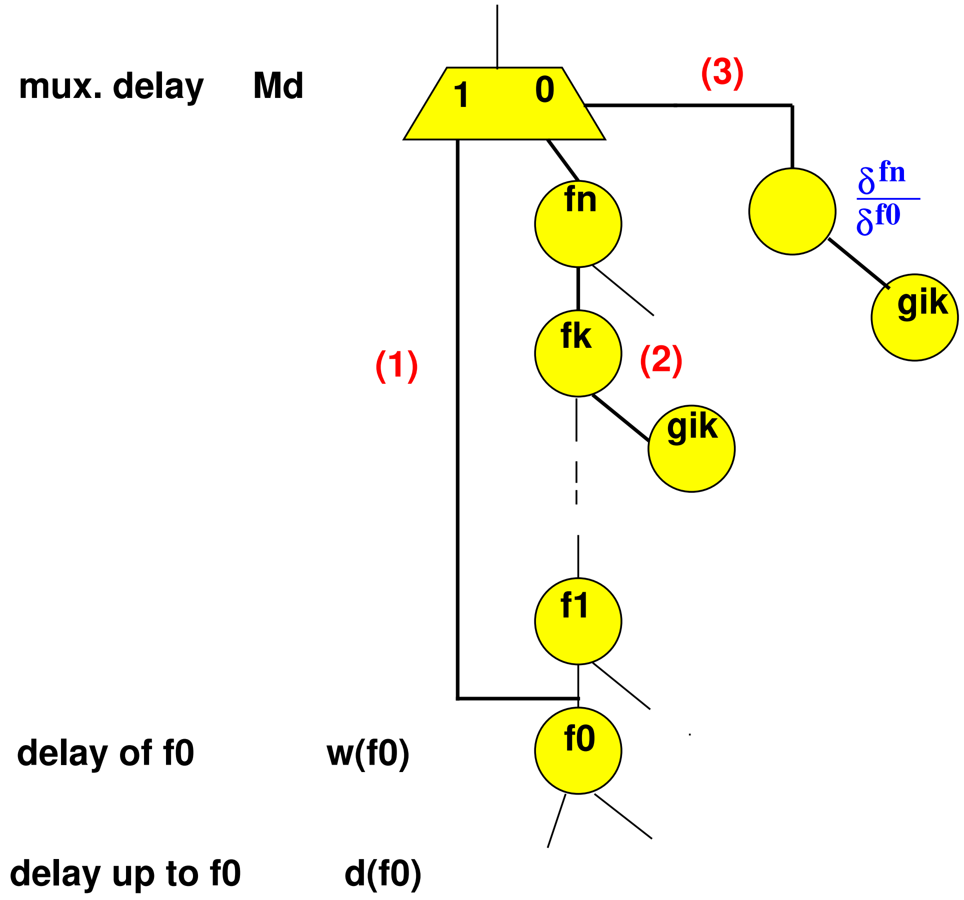
- ◇ Based on depth $\neq$ delay
- ◇ Find True Critical Paths
- ◇ Find Cutset S of partial paths covering critical paths
- ◇ Construct bypasses around partial paths
- ◇ Very little work on this:
  mostly carry-bypass adders
- ◇ New idea for speeding up circuits:
  instead of making paths **short**: make them **false**

Depth $\neq$ Delay?



- Consider some long path $P = f_0, ..., f_n$.
- Suppose the delay to each side input g_{ik} is substantially less than the delay to f_i
- Why do we have to wait for f_0?
  f_0 might control value of f_n
- But we can compute the case when f_n depends on f_0: $(\frac{\partial f_n}{\partial f_0})$,
- When this is true just tie f_n to f_0.
  (*build a bypass*).

Generalized bypass transform (GBX)



Consider each edge in F before and after transform. The function of each edge in F is *unchanged*. Hence GBX is *invariant* transform and the transform is correct.

Generalized Bypass Speedup

- Three sorts of delay to consider:
- Delay through f_0 through multiplexer:

$$(1) \qquad d(f_0) + w(f_0) + M_d$$

- Delay through $g_{i,k}, f_i, ..., f_n$, multiplexer:

$$(2) \qquad d(g_{i,k}) + \sum_{j=i}^{n} w(f_j) + M_d$$

- Delay through $g_{i,k}, \frac{\partial f_n}{\partial f_0}$, multiplexer:

$$(3) \qquad d(g_{i,k}) + d(\frac{\partial f_n}{\partial f_0}) + M_d$$

- We can always arrange for (3) $<$ (2) by reasonable construction of $\frac{\partial f_n}{\partial f_0}$ (and by choice of nodes to bypass)

Notes: (1) is usually short. (2) is only increased by MUX, but decreased by side slack. For (3) we have to build this function to be fast enough.

Candidate Bypasses and Prime Bypasses

- Bypass effective if: $slack(g_{i,k}) \geq$ MUX_delay
- Note: can ensure by choosing bypass properly
- Bypasses satisfy inequality: **candidates**
- Maximal candidate bypasses: **primes**

Choosing the Bypass:
Want: A *minimal* set of path segments such that

- Each long true path in network covered by segment
- Each segment has a positive speedup on bypass

Therefore:
Want minimal set of prime bypasses that cover (cut) every long path in the network

Min-cut problem:

Need: minimum set of prime bypasses which cuts network

GBX

GBX is effective and gives little area increase, but:

Problem: Have now created an untestable fault (on control input to multiplexer)

Can use KMS Transformation (remove all false paths without increasing delay)

Quick summary of KMS:

- f_k is last node on segment that fans out
- Duplicate false path $\{f_1, ..., f_k\}$
- f'_j fans out to every fanout of f_j except f_{j+1}
- Set f_0 input to f_1 to controlling value and propagate constant (can do this because path is false).

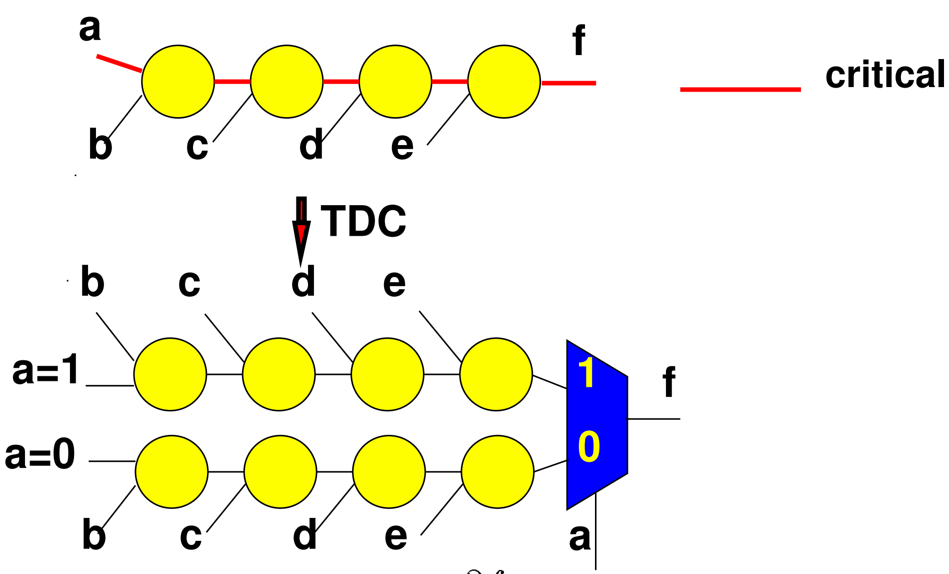
KMS Results:

1. function of every node except $f_1, ..., f_k$ unchanged
2. added $k-1$ nodes to network
3. Area added is linear in size of length of false paths

Generalized Select Transform (GST)

- Berman, Trevillyan, LaPaugh, Hathaway, ISCAS 90
- GBX: Boolean difference is applied to control of MUX
- GST: Boolean difference is contained in data lines of MUX
- GST effectively the dual of GBX
- Basic design of the carry-select adder
- Area-Delay tradeoffs *very* similar to GBX: path selection technique used for GBX should also work for this transform, too.

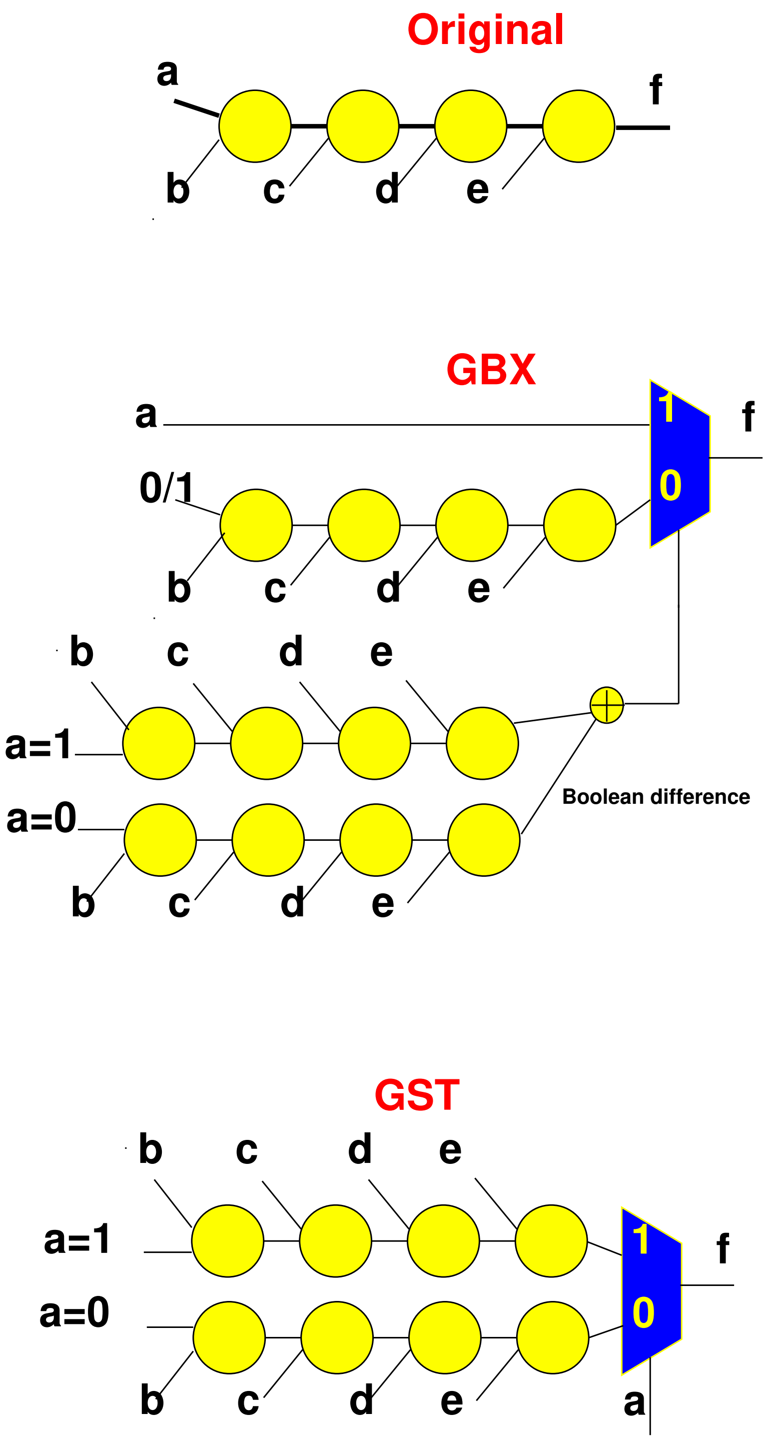
GST



Note: Boolean difference $\dfrac{\partial f}{\partial s} = f_a \oplus f_{\overline{a}}$

Use GBX path selection algorithm for this technique

GST vs. GBX


Original
a
f
b
c
d
e
GBX
a
1
f
0/1
0
b
c
d
e
b
c
d
e
a=1
Boolean difference
a=0
b
c
d
e
GST
b
c
d
e
a=1
1
f
a=0
0
b
c
d
e
a

GST vs. GBX

- Select transform appears more area efficient
- But Boolean difference generally more efficiently formed in practice. Experiments show slight advantage for GBX
- Need one MUX per fanout in GST, only one MUX in GBX
- No delay/speedup advantage for either transform

Conclusions

- Generalized Bypass Transform *appears* to have potential to be effective speedup technique
- Theoretical bounds on size of area increase
- Redundancies locally introduced, locally removed
- Bypass selection technique can be used for Generalized Select Transform

Review of technology independent delay reduction

Generally, technology delay reduction works well but not great.

Why is it hard?

- Lack of fast and accurate delay models
  - ◇ # levels: fast but crude
  - ◇ # levels + correction term (fanout, ...): little better, but what coefficients to use
  - ◇ technology mapped: reasonable, but very slow
  - ◇ place and route: better, but extremely slow
  - ◇ silicon: best, but infeasible except for FPGA's

Clustering/Partial-Collapse - Motivation

[Touati *at al.* 91]

- Traditional critical-path based methods require:
  - ◇ well defined critical path
  - ◇ good delay information
  - ◇ hard slack information
- Problems:
  - ◇ good delay information comes from mapper
  - ◇ delay estimates and models are weak
- Possible Solutions:
  - ◇ better delay modeling at technology independent level
  - ◇ **make speedup insensitive to actual critical paths, mapped delays**

Clustering and partial collapse:

- Two level circuits are fast
  - ◇ collapse circuit to 2-level - but huge area and capacitive loading on inputs
- To avoid huge area penalty
  - ◇ identify clusters of nodes
  - ◇ each cluster has some fixed size
  - ◇ perform collapse of cluster
  - ◇ simplify each node
- Details:
  - ◇ How to choose the clusters?
  - ◇ How to choose cluster size?
  - ◇ How to simplify each node?

Lawler's Custering Algorithm

- Optimal in delay:
  - ◇ for a given cluster
- May duplicate nodes: (hence possible area penalty)
  - ◇ not optimal w.r.t. duplication
  - ◇ use a heuristic
- Fast: $O(m * K)$
  - ◇ $m =$ number of edges in network
  - ◇ $K =$ maximum cluster size

Clustering algorithm: (K is cluster size)

- **Label phase:** (from inputs to outputs):

◇ if node u is an input $L \leftarrow 0$
  else $L \leftarrow$ max label of fanin of u

◇ if —nodes in TFI(u) with label $= L$—$< K$
    label(u) $\leftarrow L$
  else label(u) $\leftarrow L + 1$

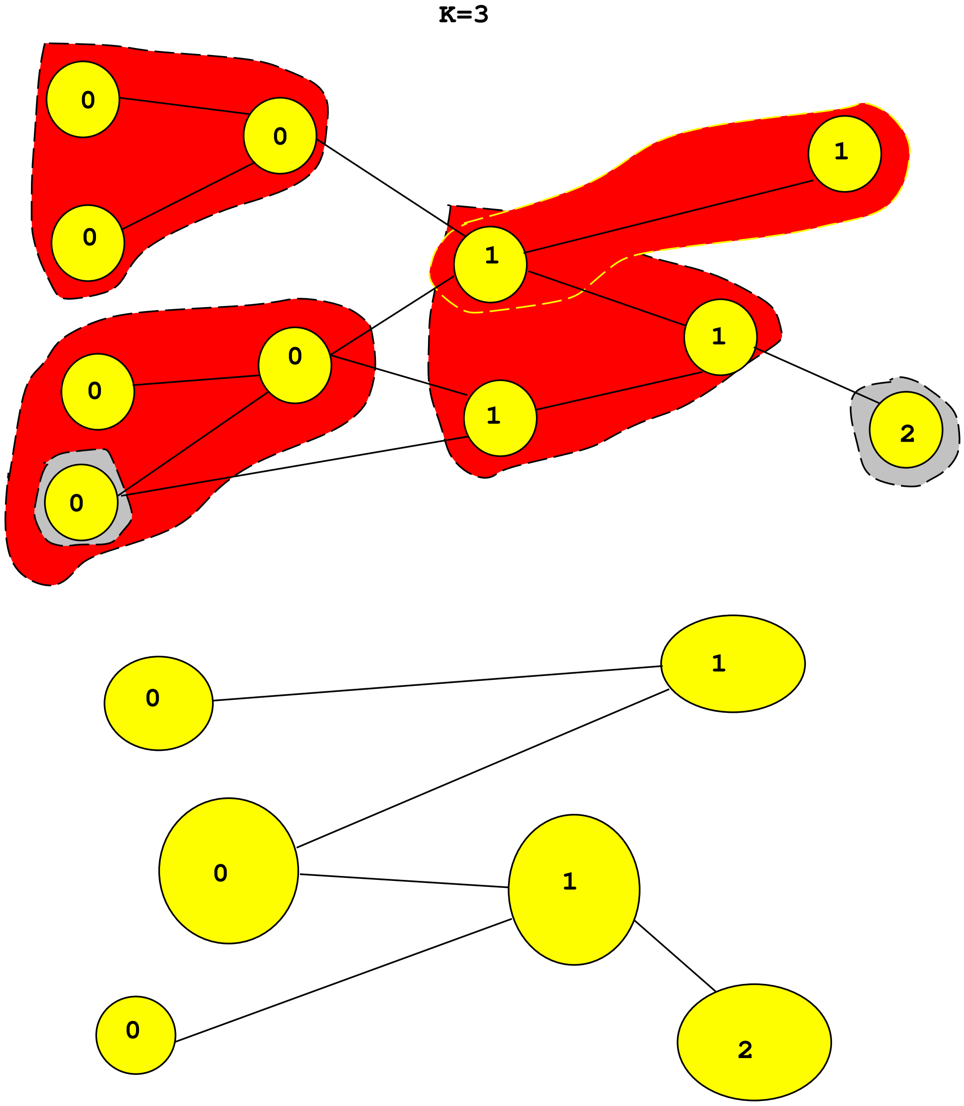
- **Cluster phase:** (outputs to inputs):

◇ if node u is an output, $L \leftarrow \infty$
  else $L \leftarrow$ max label of fanout of u

◇ If label(u) $< L$ then create a new cluster
    with "root" u and with members all the nodes
    in TFI(u) with label $=$ label(u)

- **Collapse phase:** (order independent):
  - ◇ collapse each node in a cluster into single node
  - ◇ **note:** a node may be in several clusters.

Example of clustering

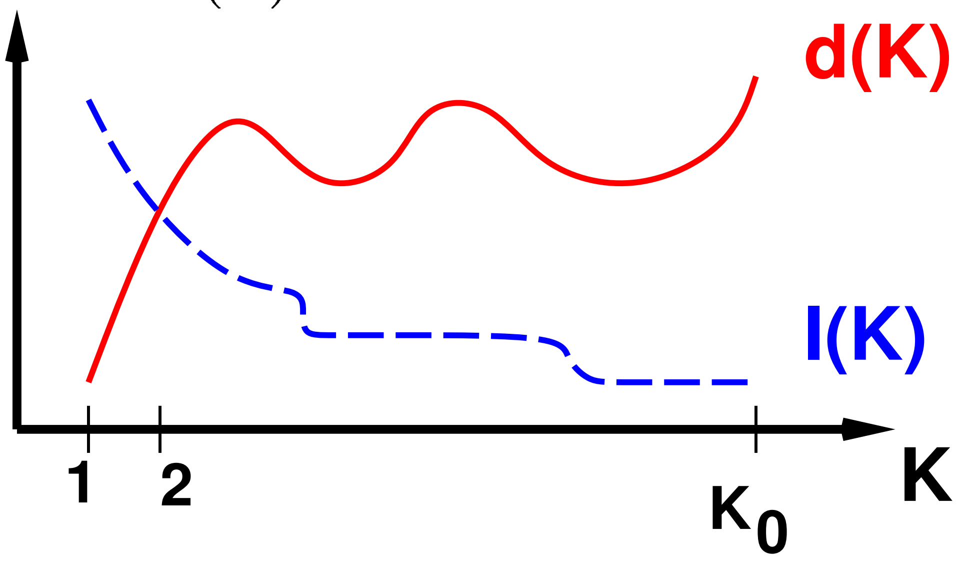


Result: Lawler's algorithm gives minimum depth (number of clusters in path from input to output) circuit.

Typically we decompose initial circuit into 2-input NAND's and invertors. Then cluster size K reflects # 2-input NAND's to be collapsed together.

Choosing K

- I(K): number of levels, given K
  I(3) = 3 (for example)
- d(K): duplication ratio
  number of gates in clustered network ÷
  number of gates in original network
  d(3) = 13/11
- For every K from 2 to K_0 where $\{K_0/d(K_0) \approx 2.0\}$
  compute $d(K), I(K)$
  use exhaustive simulation: label and
  cluster without collapse for each K
  Each iteration is $O(|E|K)$
- Choose K such that $I(K)$ minimized
  break ties using $d(K)$
  minimize $d(K)$



Area recovery

- Area increase due to
  - ◇ node duplication
    this occurs when node is in multiple clusters

- Two solutions:
  - ◇ Break clusters into smaller pieces
    off critical path
  - ◇ After cluster and collapse, recover area

- Relabeling procedure: Attempt to increase node labels without exceeding cluster size

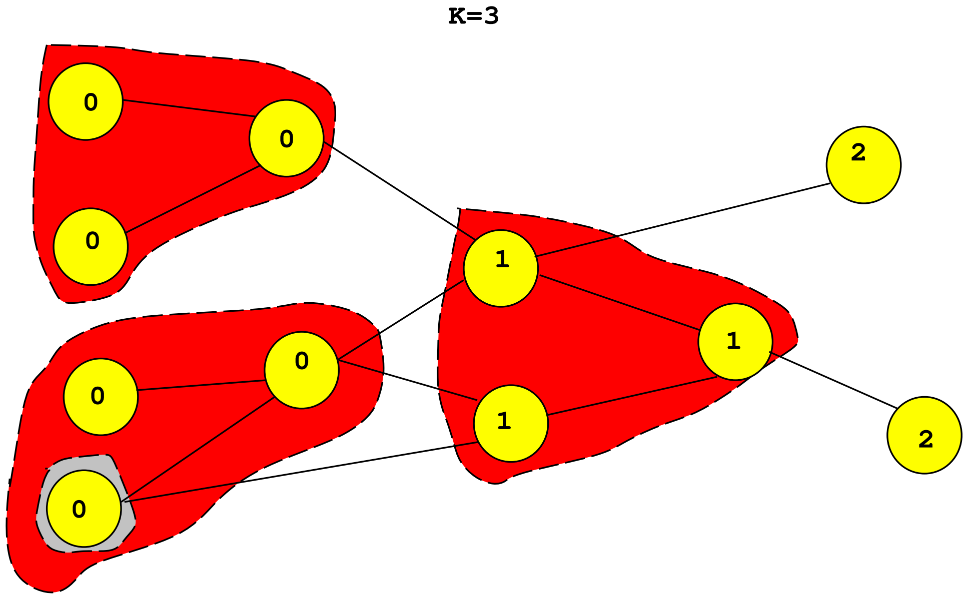
◇ In reverse topological order

Assign

$$\text{label}(O_i) = \max_{j \in PO}\{\text{label}(O_j)\}$$

Increase the label of node u if
* new-label(u) $\leq$ label(v) for each fanout v, and
* new-label(u) $=$ new-label(v) for each fanout v **only if** label(u) $=$ label(v) before relabeling, and
* no cluster size violated

Relabeling example



Area reduction is 8%
No effect on delay

Post-collapse area recovery:

- Algebraic factorization
  - Undo factorization if depth increases
- Simplification
  - Only consider node v as fanin (introduced by simplify using don't cares) if level of v is less than level of node
- Redundancy removal

Comparison: THR-PC-GBX

Name	Initial Delay	Final Delay			Initial Area	Final Area		
		THR	GBX	PC		THR	GBX	PC
5xp1-hdl	16.0	10.0	15.0	15.0	139	210	140	126
5xp1	29.0	17.0	17.0	22.0	309	312	345	265
alupla	18.0	12.0	14.0	19.0	288	355	301	342
f51m-hdl	17.0	10.0	16.0	13.0	133	186	134	119
f51m	34.0	21.0	23.0	31.0	313	350	347	289
misex1	13.0	10.0	12.0	12.0	114	121	120	99
misex2	13.0	7.0	10.0	8.0	182	223	183	208
rd84-hdl	20.0	14.0	19.0	15.0	184	194	187	128
seq	26.0	19.0	20.0	20.0	1842	1890	1875	1869
sao2-hdl	45.0	23.0	35.0	32.0	462	299	483	396
C1908	36.0	24.0	31.0	28.0	894	1500	908	1112
C2670	33.0	19.0	30.0	32.0	1688	1594	1692	2166
C3540	46.0	38.0	40.0	38.0	4178	3056	4214	3396
C880	38.0	20.0	34.0	32.0	836	1006	849	966
C5315	40.0	32.0	35.0	34.0	3280	3360	3494	4221
C6288	130.0	77.0	129.0	88.0	7374	7596	7376	5369
C7552	53.0	31.0	44.0	58.0	4394	4326	4446	4740
9sym	12.0	13.0	12.0	13.0	548	515	548	444
9symml	18.0	15.0	13.0	16.0	532	491	549	468
bw	30.0	18.0	16.0	18.0	312	344	414	314
duke2	20.0	17.0	14.0	17.0	946	946	1034	848
misex3c	44.0	34.0	26.0	44.0	1090	1044	1165	1097
rd84	15.0	15.0	14.0	18.0	268	261	277	235
sao2	30.0	23.0	21.0	23.0	420	378	437	269
9sym-hdl	31.0	19.0	27.0	19.0	260	275	263	152
rd73	21.0	12.0	20.0	12.0	196	192	199	164
misex3	35.0	25.0	32.0	24.0	1172	946	1176	822
C432	40.0	36.0	37.0	34.0	591	501	617	498
C499	21.0	20.0	21.0	17.0	796	860	796	844
C1355	21.0	20.0	21.0	17.0	796	860	796	844

THR: Tree-height reduction, GBX: Generalized bypass, PC: Partial collapsing. All ckts of 2-input gates, initial/final circuits irredundant, Initial ckts area optimized, Delay = #levels, Area (= 2 $\times$ # of 2-input gates)

Conclusions

- Variety of methods for delay optimization
  - ◇ No single technique dominates
- When applied to ripple-carry adder get
  - ◇ carry-lookahead adder (tree-height reduction)
  - ◇ carry-byapss adder (GBX)
  - ◇ carry-select adder (GST)
  - ◇ ? (Partial collapse)
- All techniques still ignore false paths