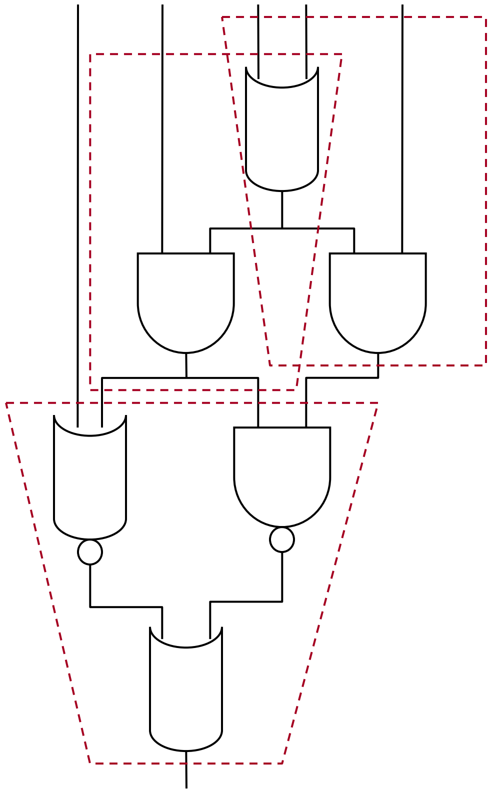
Technology Mapping

Outline

- Logic Synthesis
- Basics of Technology Mapping
- DP Tree-map algorithm
- Greedy Tree-map algorithm
- FlowMap algorithm
- DAOmap algorithm
- WireMap algorithm



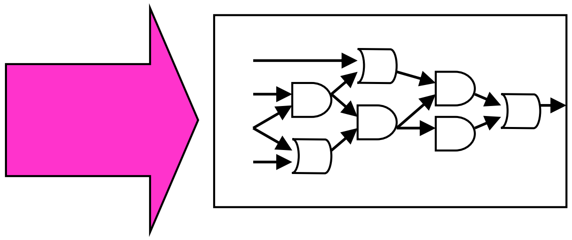
HDL Synthesis

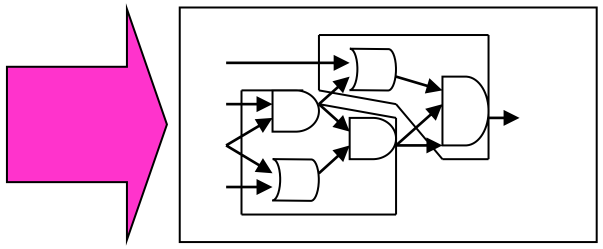
Synthesis = Domain Translation + Optimization

Domain translation

Optimization (area, timing, power...)

```
--VHDL                          //Verilog
if(A='1') then                  if(A==1)
  Y<=C + D;                       Y=C + D;
elseif (B='1') then             else if(B==1)
  Y<=C or D;                      Y=C | D;
else Y<=C;                      else Y=C;
endif
```





Behavioral domain

Structural domain

Syntax-directed Translation

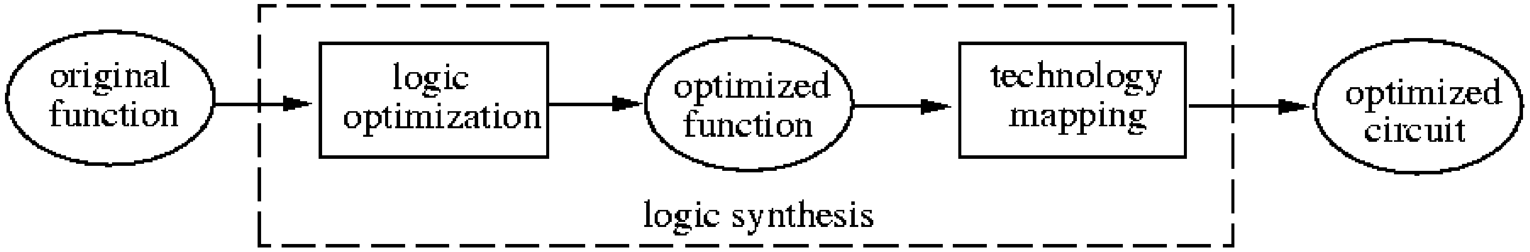
- Translate HDL into logic directly.
    - ab + ac
- Generally requires *optimization*.

Macros

- Pre-designed components
  - Generally identified by language features.
  - E.g. + operator, xxx()
- *Hard macro*: includes placement.
- *Soft macro*: no placement.

Logic Synthesis Phases

- *Logic optimization* transforms current gate-level network into an optimized gate-level network.
- *Technology mapping* transforms the gate-level network into a network of cells in the target technology library.



Logic Optimization

- Work on Boolean expression equivalent.
- Estimate size based on literal count.
- Estimate delay by simple delay models.
- Use *factorization, substitution, decomposition, extraction*, etc. to optimize logic.

Logic Optimization

1. **Decomposition**
(single function)

$f = abc + abd + c'd' + a' + b'$

$\Downarrow$

$f = xy + (xy)'$
$x = ab$
$y = c + d$

2. **Extraction**
(multiple functions)

$f = (az + bz')cd + e$
$g = (az + bz')e'$
$h = cde$

$\Downarrow$

$f = xy + e$
$g = xe'$
$h = ye$
$x = az + bz'$
$y = cd$

Logic Optimization

3. Factoring

(series-parallel decomposition)

$f = ac + ad + bc + bd + e$

$$\Downarrow$$

$f = (a + b)(c + d) + e$

4. Substitution

$f = ga + g'b$

$g = c + d$

$$\Downarrow$$

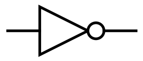
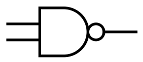
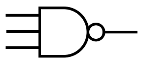
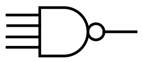
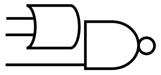
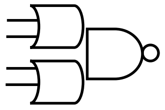
$f = ac + ad + bc'd'$

$g = c + d$

Technology Mapping

- Map Boolean expressions into a particular *cell library*.
- Mapping takes into account area, delay, etc.
- May perform some optimizations in addition to mapping e.g. retiming.
- Allow more accurate delay models.

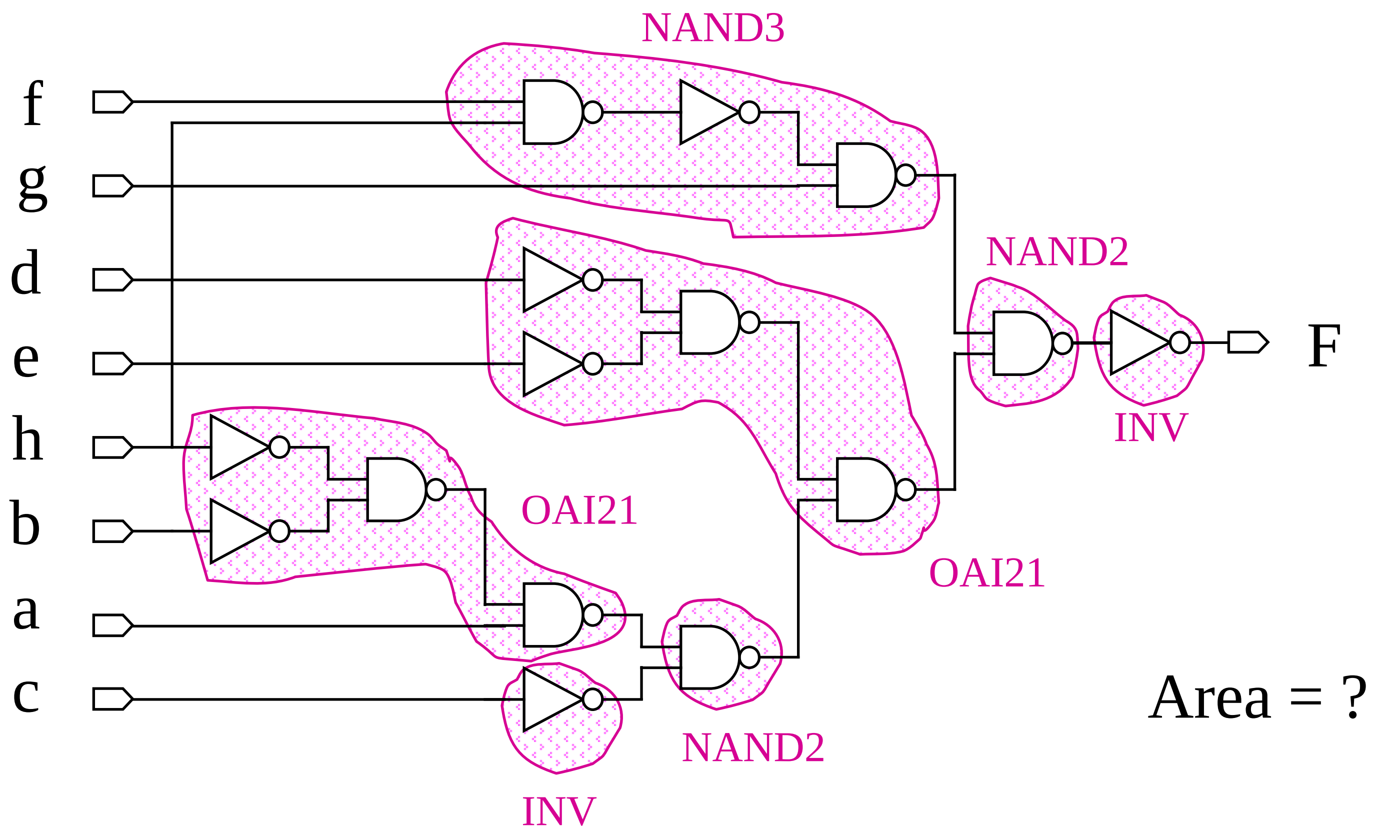
An Example Cell Library

INV	2		a’
NAND2	3		(ab)’
NAND3	4		(abc)’
NAND4	5		(abcd)’
OAI21	4		((a+b)c)’
OAI22	5		((a+b)(c+d))’

Library Element

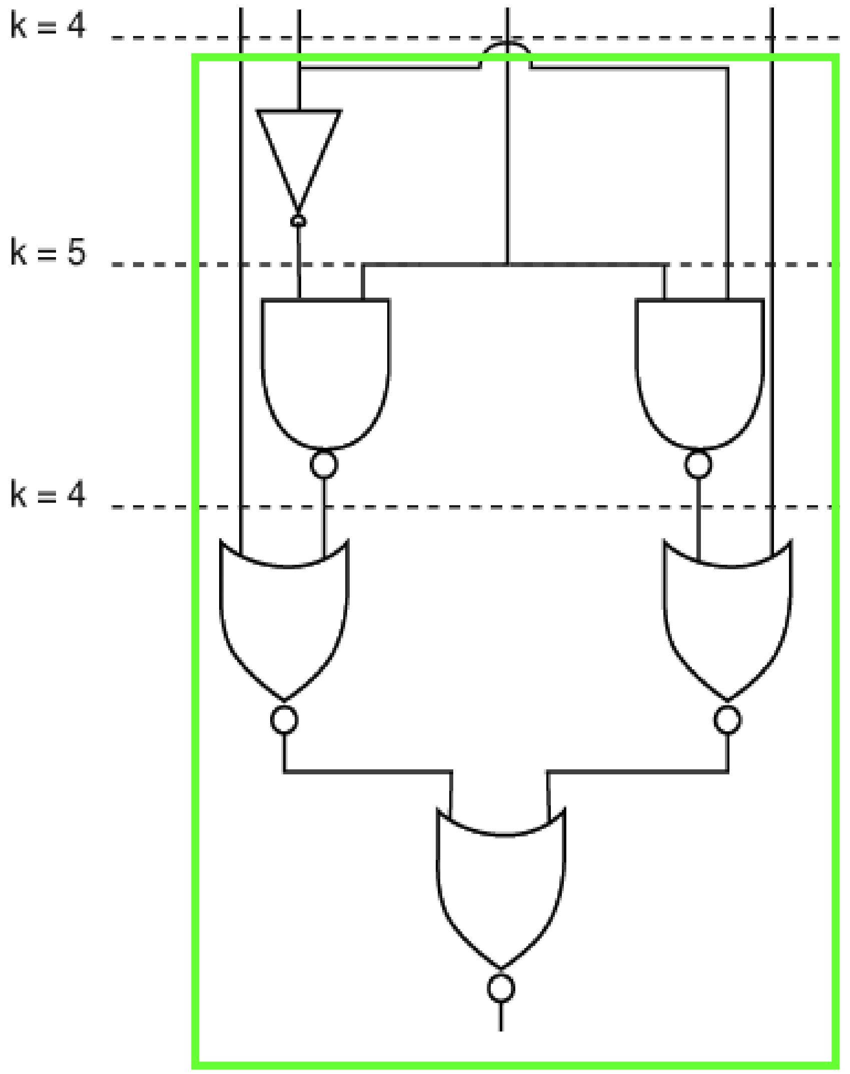
Technology Mapping

- Cover function with library cells
- To optimize area, delay, etc.



LUT-based Technology Mapping

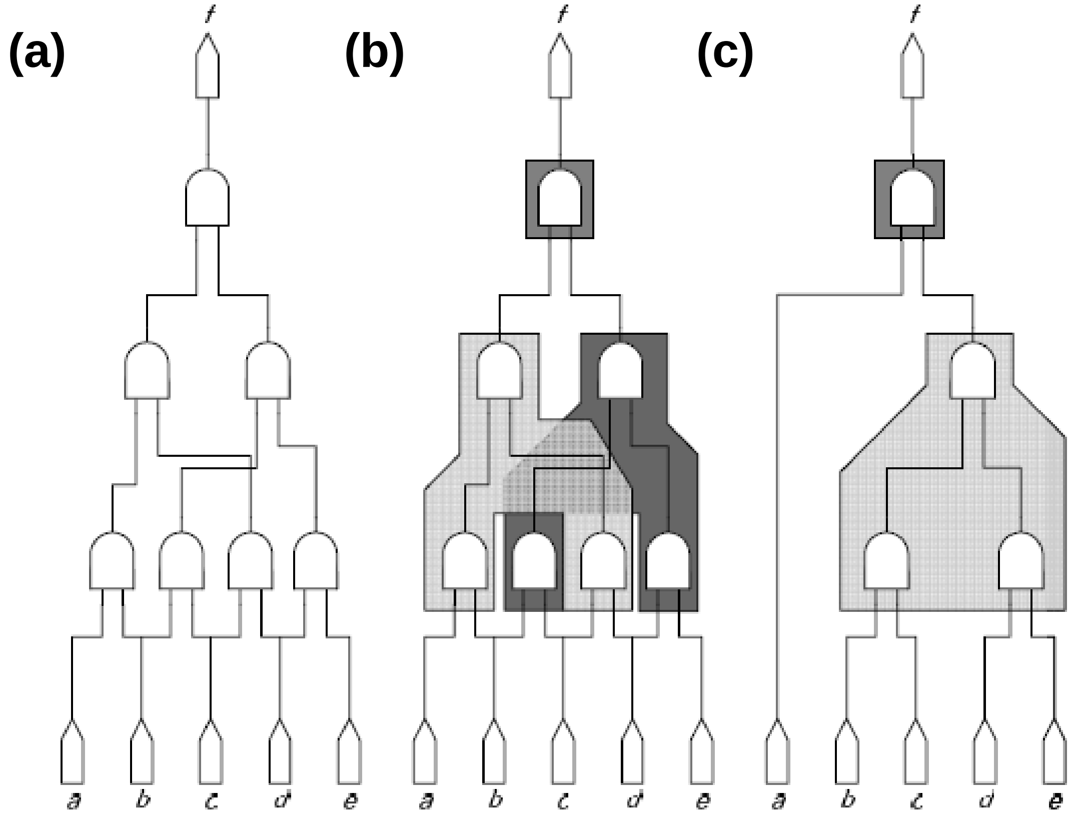
- # inputs *doesn't* always increase with added functions.
- Useful to find the largest cone that will fit into a LUT.



FPGAs vs. Custom Logic

- Area cost metric for static gates is literal:
  - ax + bx’ has four literals, requires 8 transistors.
- Area cost metric for FPGAs is logic element:
  - All functions that fit in an LE have the same cost.

Logic Optimization & Technology Mapping



(a) Original network

(b) Mapping without logic optimization

(c) Mapping with logic optimization

Technology Mappers for LUT-Based FPGAs

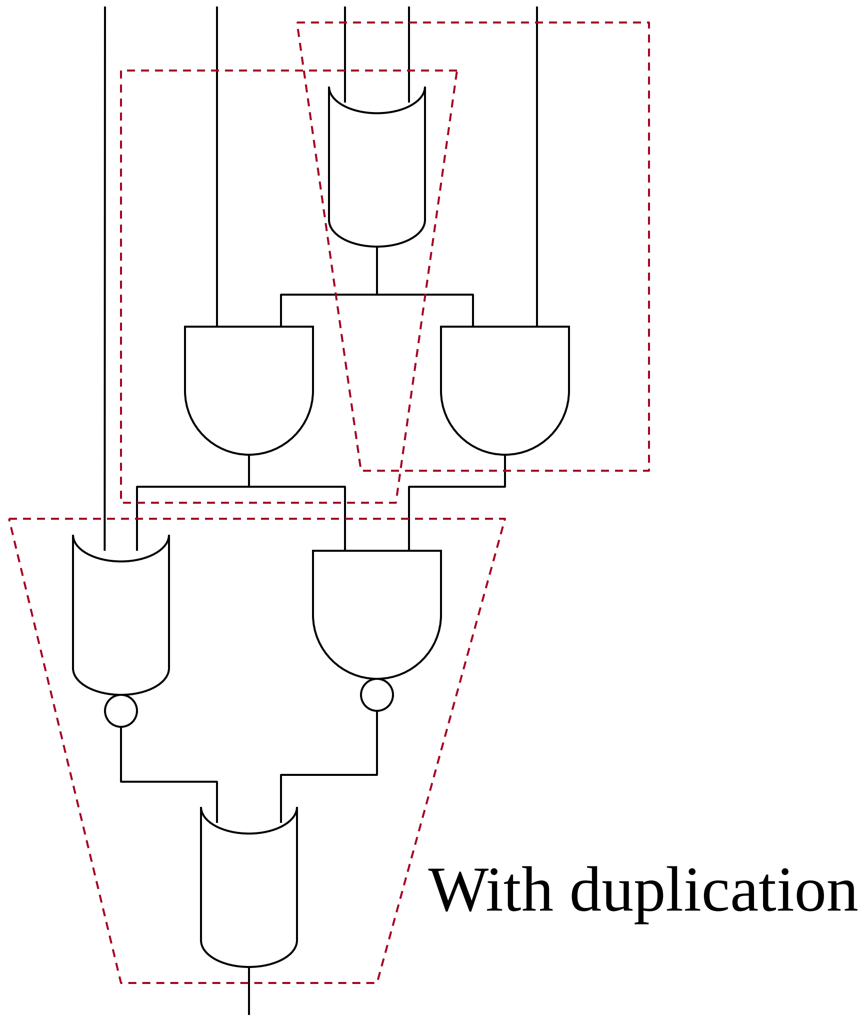
- Input can be a sequential circuit.
- Most algorithms work on the combinational portions which are *directed acyclic graphs* (*DAGs*).
- For mapping with *K*-input LUTs
  - Remove sequential elements to break into combinational portions
  - Covering *K*-bounded DAGs by *K*-feasible cones
- Property: a *K*-LUT can implement any function of $\leq K$ inputs.

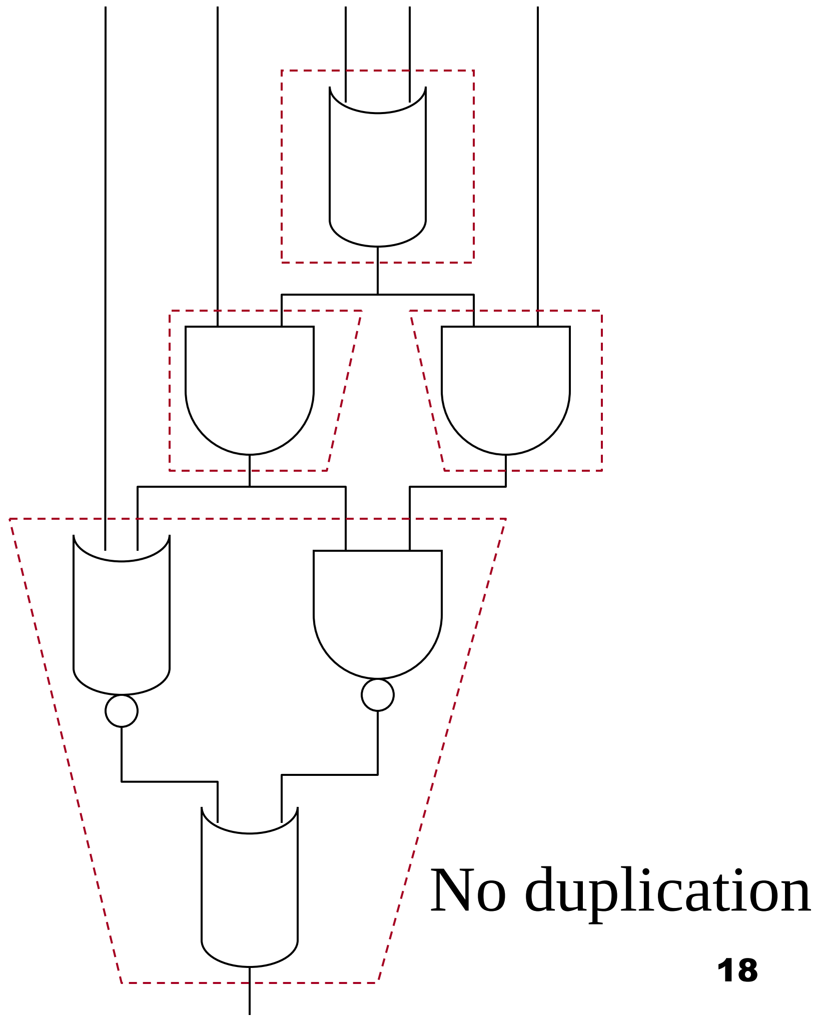
Technology Mapper Classification

- Classification based on objectives:
  - *Area* minimization
  - *Delay* minimization
  - *Power* minimization
  - *Routability* optimization
- Some algorithms perform duplication-free mapping, others allow *duplication* (i.e., a node may be covered by more than one LUT).
- Duplication can lead to reduced no. of LUTs and/or delay.

Node Duplication

- 3-LUTs mappings:





Hardness of Area /Power/Delay Minimization

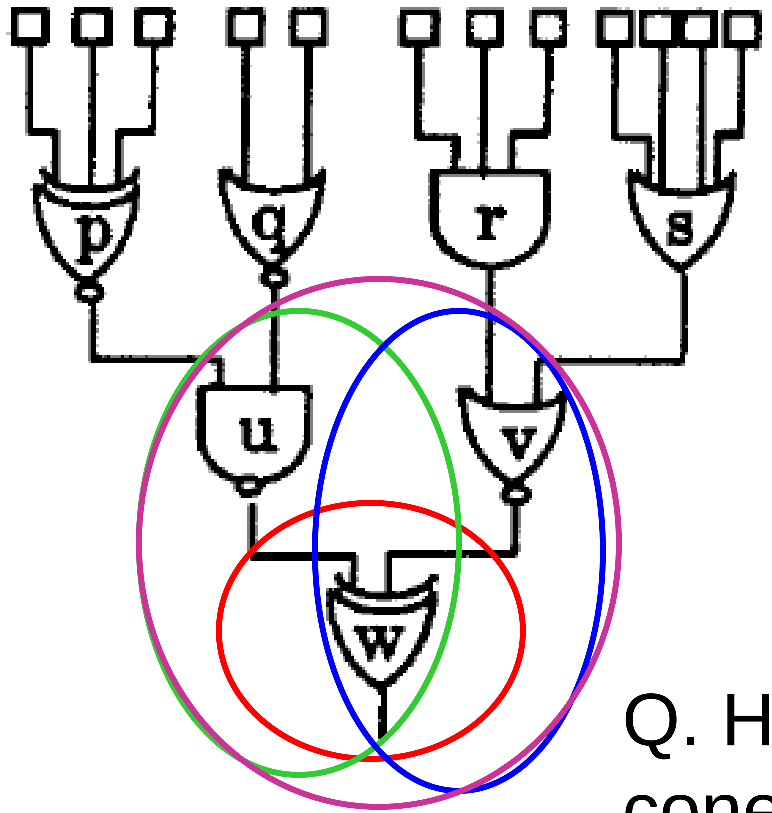
- Area-optimal mapping problem (node duplication allowed) is NP-hard.
- Power-optimal mapping problem is NP-hard.
- But duplication-free area-optimal mapping is polynomial time solvable.
- Delay-optimal mapping problem is polynomial time solvable.

DP Tree-map: duplication-free area-optimal mapping for tree

- Cover a *K*-bounded tree with minimum no. of *K*-LUTs.
- Optimal *dynamic programming* approach.
- Process nodes in *topological order* starting from PIs.
- Recursive assumption: When computing the best mapping of tree T_i, the best mapping of all its subtrees are known.
- Enumerate all *K-feasible cones* rooted at node *i*.
- If mapping MT_i uses a LUT_i to implement node *i*:
  - $\text{Area}(MT_i) = 1 + \sum_{j \in \text{input}(\text{LUT}i)} \text{Area}(MT_j)$.
- A best mapping MT_i* is one such that Area is minimum.

DP Tree-map

- Min-area tree mapping with K=5



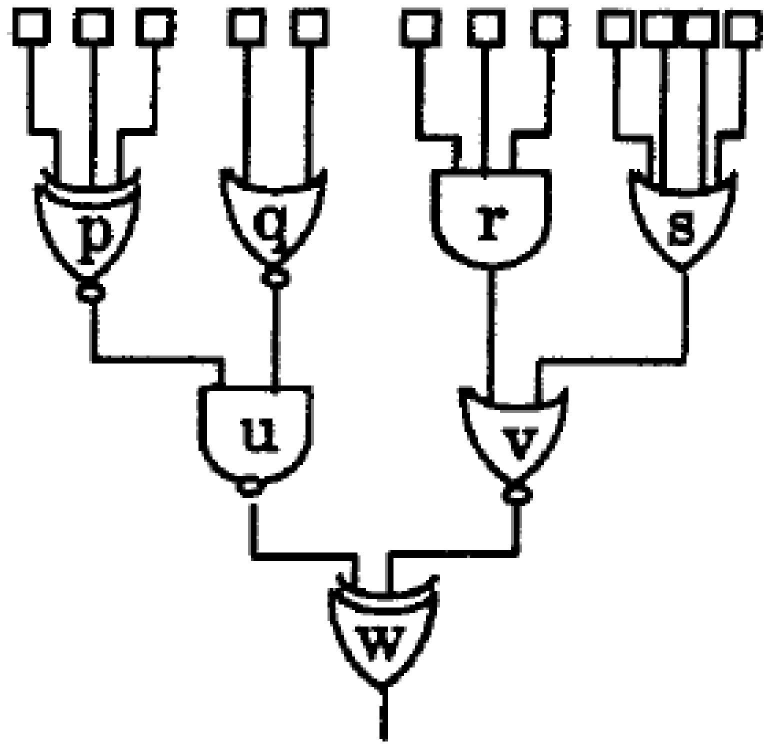
Q. How many 5-feasible cones rooted at *w*?

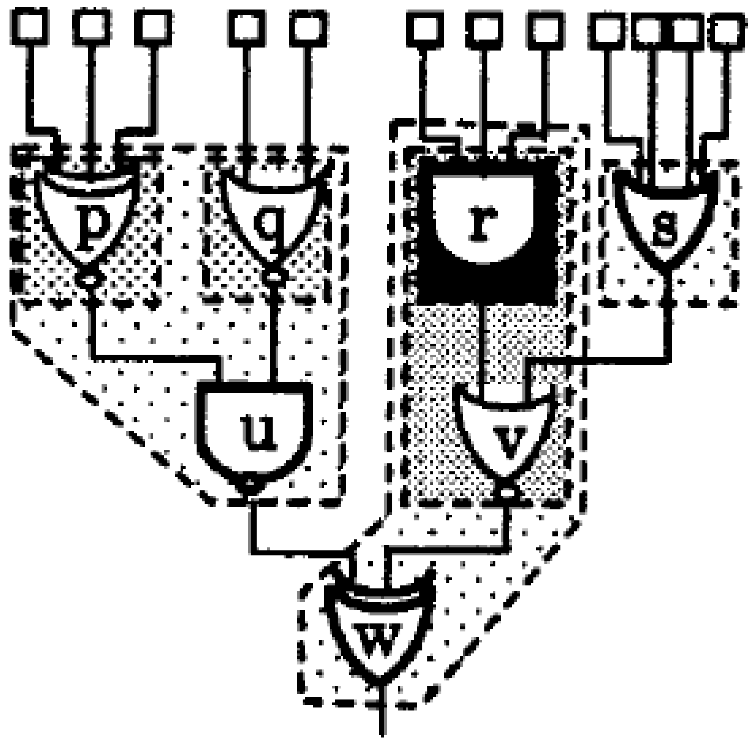
Greedy Tree-map: duplication-free area(delay)-optimal mapping for tree

- An optimal greedy algorithm without enumeration for min-area tree mapping.
- Let $w1, \ldots, wm$ be the fanin nodes of node i and $|\text{input}(\text{LUT}_{w1})| \leq \ldots \leq |\text{input}(\text{LUT}_{wm})|$.
- Greedy packing:
  - Cover i by $\text{LUT}_i = \{i\} \cup_{j\leq s} \text{LUT}_{wj}$ where s is the largest index s.t. LUT_i remains K-feasible
- Time complexity is $O(\max\{K, \log n\}n)$.
- Modification for depth-optimal tree mapping:
  - Order the fanin nodes of node i by their LUTs' depths in decreasing order.

Greedy Tree-map

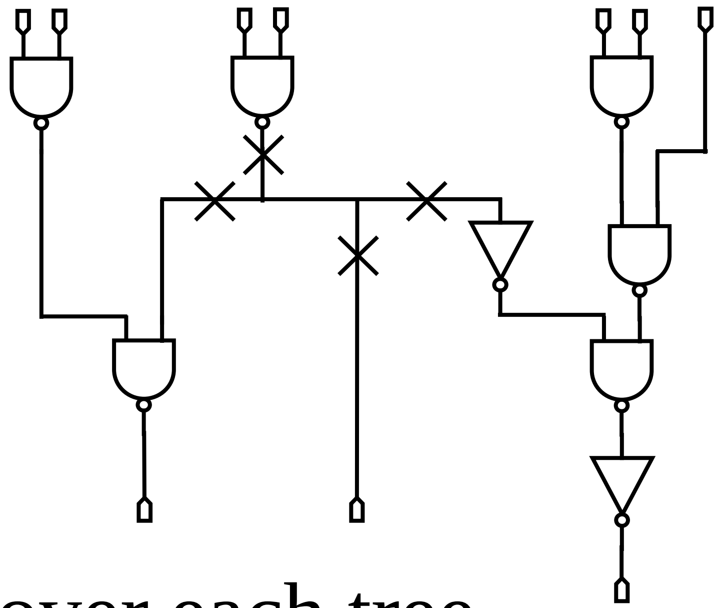
- Min-area tree mapping with *K*=5





Heuristic for General Network Mapping

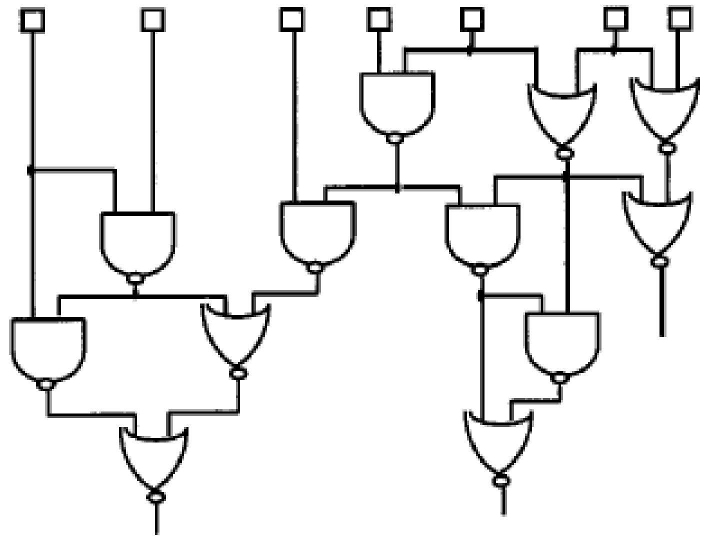
- Partition a network into trees
  - Cut the network at all multiple fanout points



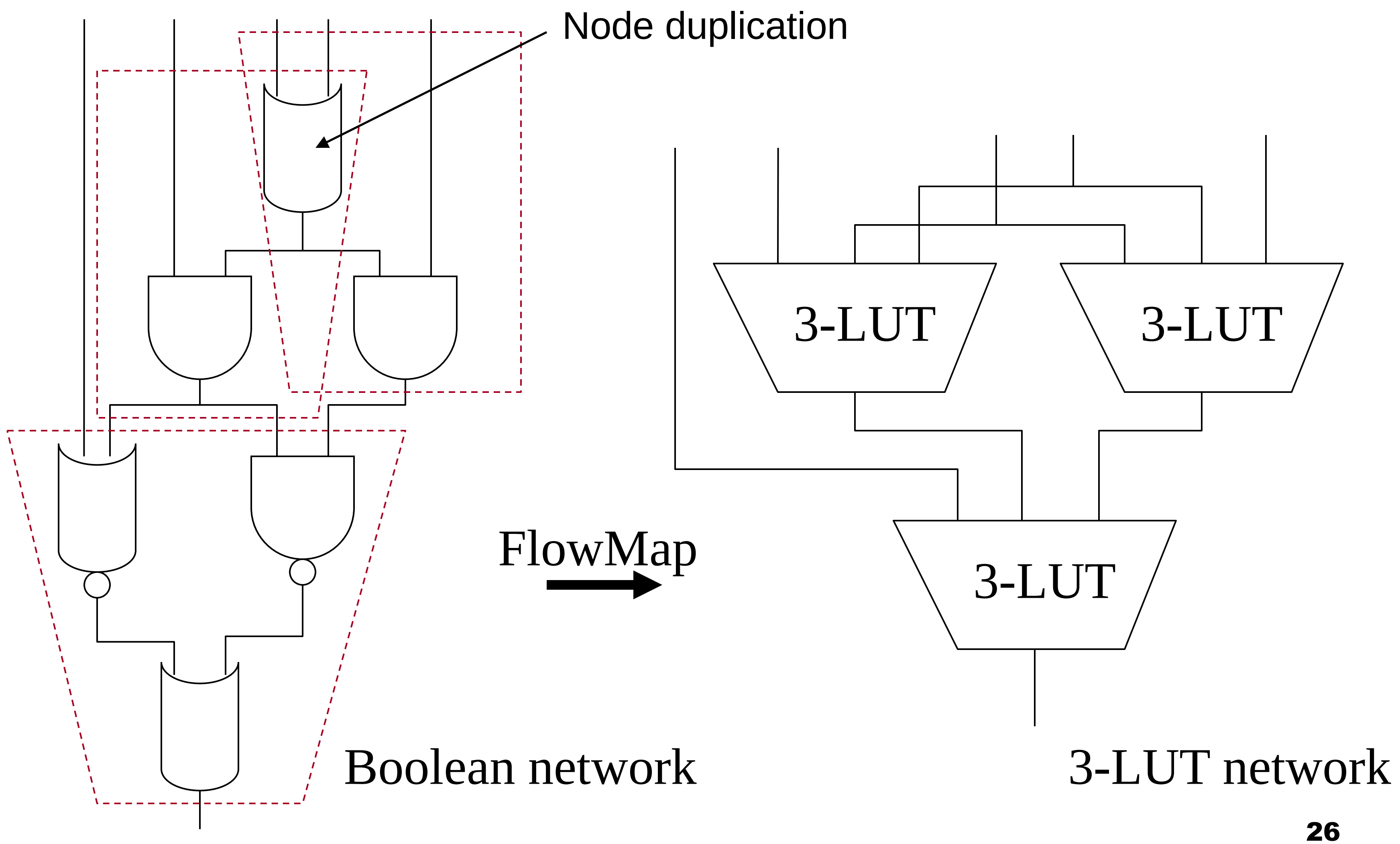
- Optimally cover each tree
- Piece the tree-covers into a cover for the subject graph

FlowMap: delay-optimal mapping for DAG

- Guarantee *depth-optimal* mappings for general DAGs.
- Automatically consider *duplication* for depth minimization(i.e., a node may be covered more than once).
- Use *network flow*.



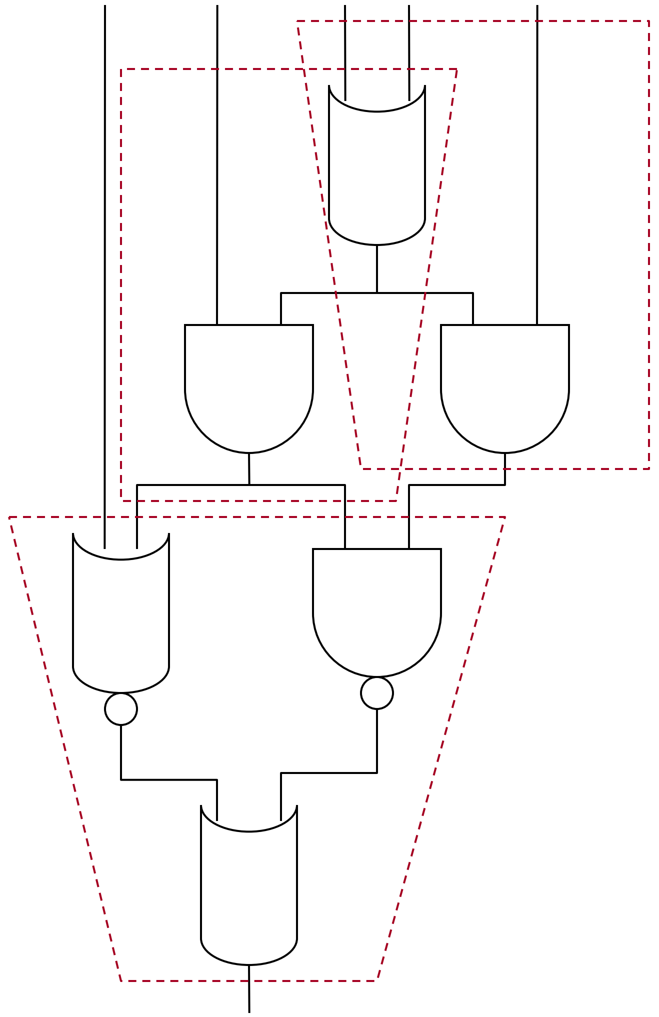
FlowMap

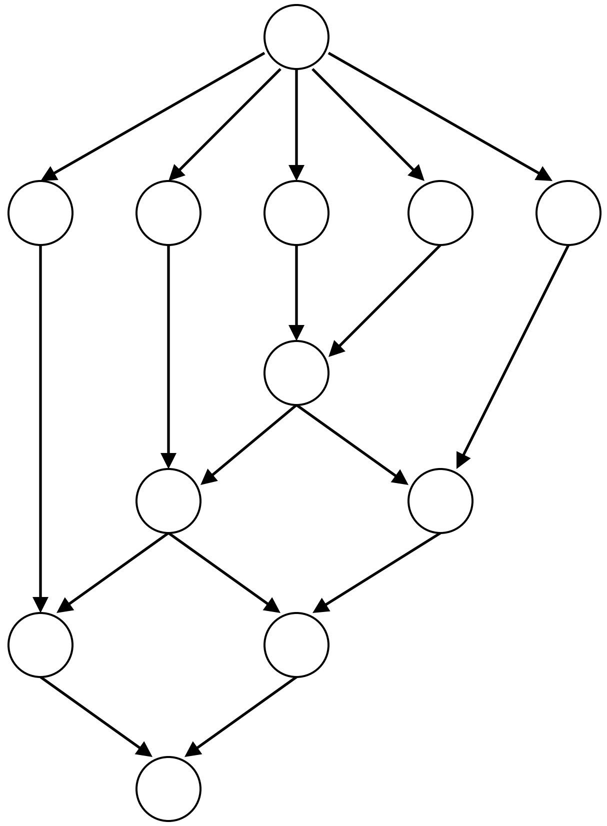


Node duplication
FlowMap
3-LUT
3-LUT
3-LUT
Boolean network
3-LUT network

FlowMap

- Graph model





Represent the network by a graph.

FlowMap

- Terminologies

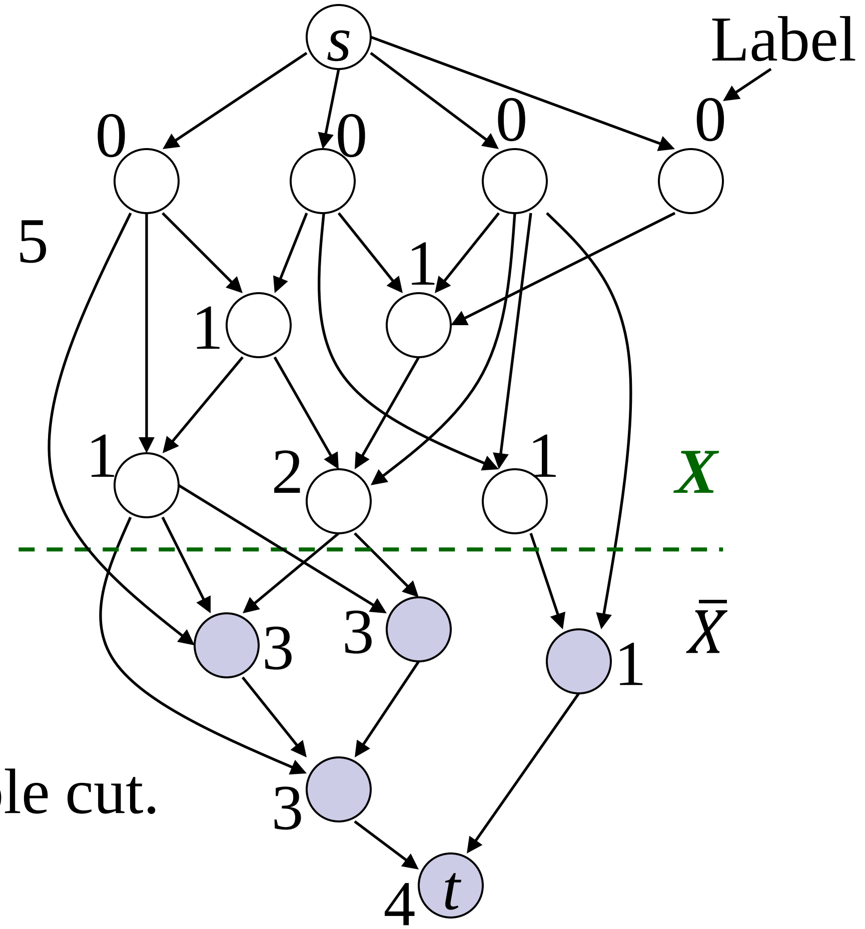
Node cut size $n(X, \bar{X}) = 5$

Edge cut size $e(X, \bar{X}) = 8$

Volume of a cut, $vol(X, \bar{X}) = |\bar{X}| = 5$

k-feasible cut: $n(X, \bar{X}) \leq k$

Height of a cut, $h(X, \bar{X}) = \max\{\text{label}(x) : x \in X\} = 2$



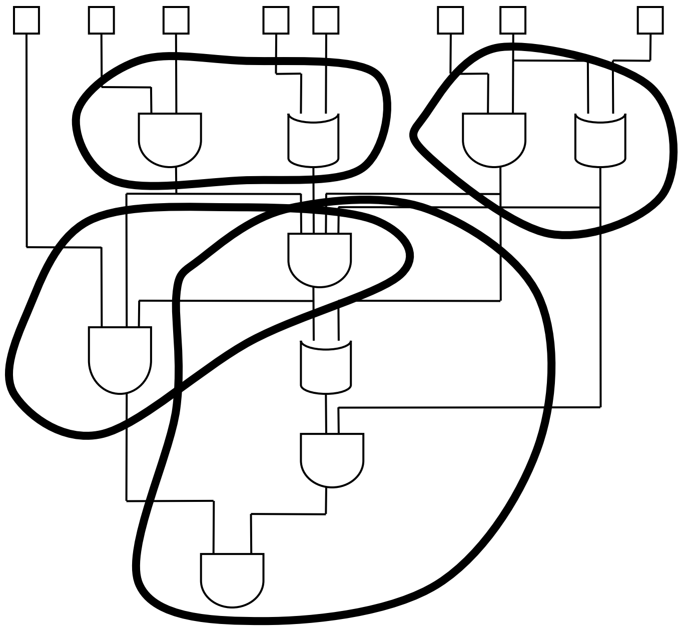
This example shows a 5-feasible cut.

FlowMap

- Assumptions:
  - Map to *K*-input LUTs.
  - Input Boolean network is *K*-bounded (each node has fanin $\leq K$).
  - *Unit delay* model.
- *Labeling*: Compute a label for each node, that reflects the level of the *K*-LUT implementing that node in an optimal mapping solution.
- *Matching*: Generate the *K*-LUT mapping solution based on the node labels.

Delay Model of FlowMap

- Unit delay model
  - Delay minimization is equivalent to depth minimization under unit delay model

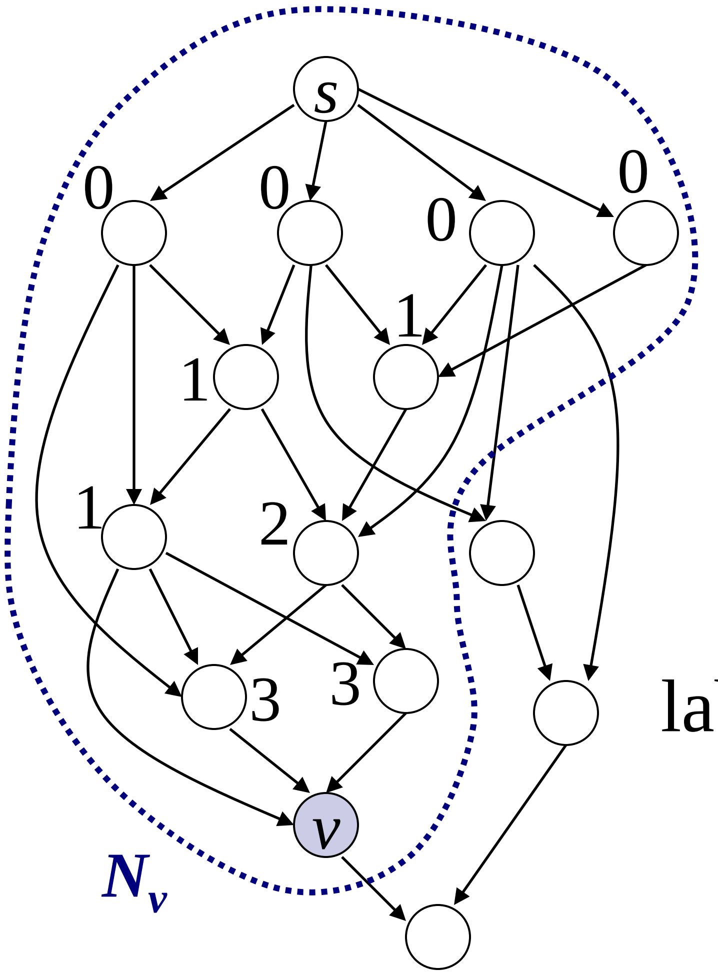


Delay = 3

Labeling Phase of FlowMap

- Labeling Phase
  - Process the nodes in *topological order*.
  - label(v) = depth of the optimal K-LUT mapping solution for node v.

Labeling Phase of FlowMap



Want to find:

(i) label(v), and

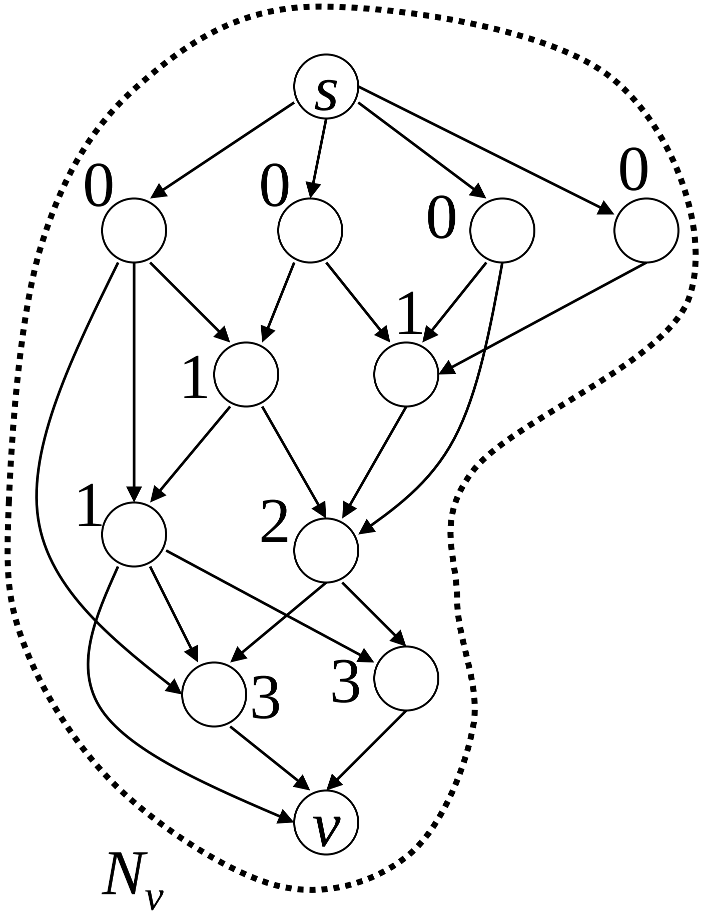
(ii) the K-LUT for v in the optimal mapping solution of N_v.

$$\text{label}(v) = \min_{\substack{(X, \bar{X}) \text{ is a} \\ K\text{-feasible cut in } N_v}} \{h(X, \bar{X})\} + 1$$

Labeling Phase of FlowMap

- Lemma 1: label(v) = p or p+1, where p is the maximum label of the input nodes to v. (Why?)
- Computation of label(v):
  - Check if there is a K-feasible cut $(X, \bar{X})$ in N_v of height p-1.
  - If yes, label(v) = p, $\bar{X}$ and v are pushed into the same K-LUT.
  - Else, label(v) = p+1, use a new K-LUT for node v.

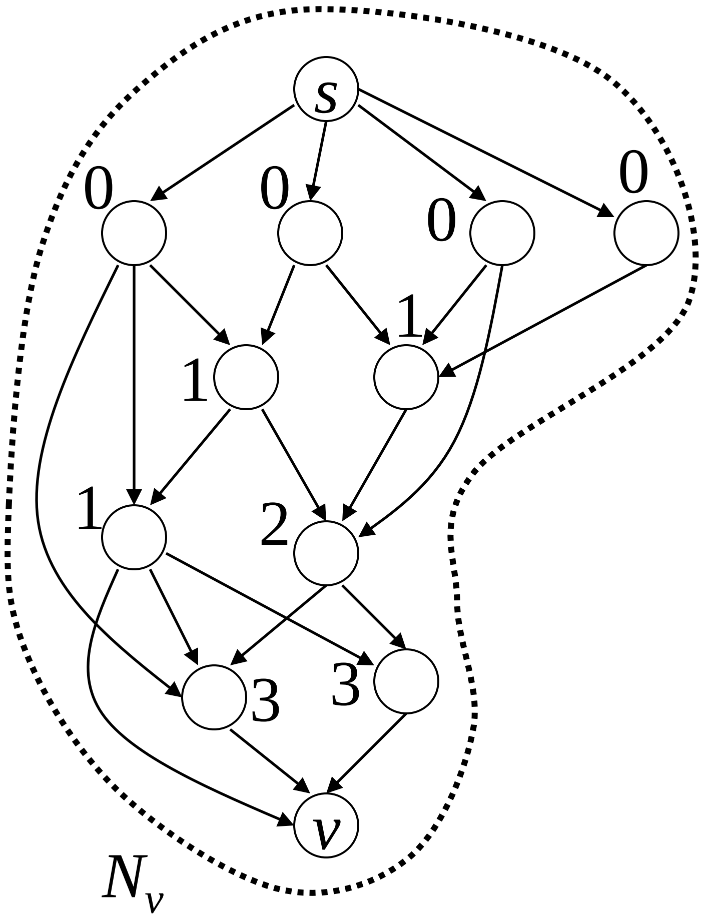
Labeling Phase of FlowMap



In this example, $p = 3$, i.e., label(v) = 3 or 4

How to determine if there is a K-feasible cut for N_v with height 2?

Labeling Phase of FlowMap



Let K = 3.

How to determine if there is a 3-feasible cut for N_v with height 2?

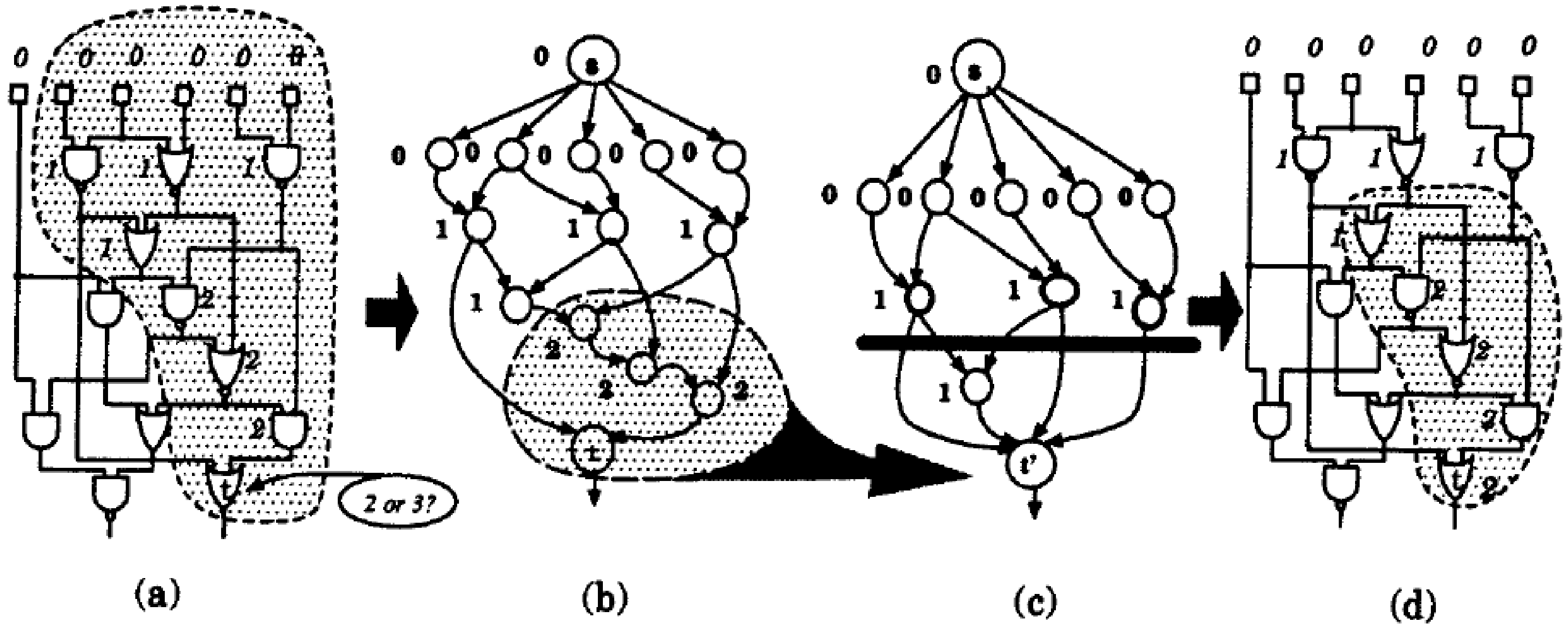
Note: If label(v) = 3, which nodes must be in the same LUT as v?

Labeling Phase of FlowMap

- Question: How to determine if there is a K-feasible cut of height p-1 in N_v?

  (Hint: You can use the *network flow* method.)
- What is the time complexity of this method?

Example



Assume K=3

(a) Original network and labels of nodes in N_t

(b) Transform into flow network

(c) Collapse nodes with largest labels, compute K-feasible cut

(d) Get label(t) and LUT(t)

Time Complexity of Labeling Phase

Labels of all the nodes in the subject tree can be computed in O(Kmn) time where n = no. of nodes and m = no. of edges. Since m = O(n) for any K-bounded input network, the time complexity is O(n^2).

Mapping Phase of FlowMap

1. S = Set of primary output nodes
2. Choose any $t \in S$.
3. Use K-LUT(t) as one of the K-LUTs.
4. Delete t from S.
5. Add *Input*(K-LUT(t)) to S.
6. Goto step 2.

FlowMap Complexity

1. Labeling phase takes O(Kmn) time.
2. Mapping phase takes O(n) time.
3. Hence, the total time complexity is O(Kmn), i.e., O(n^2).

Minimization of #LUTs in FlowMap

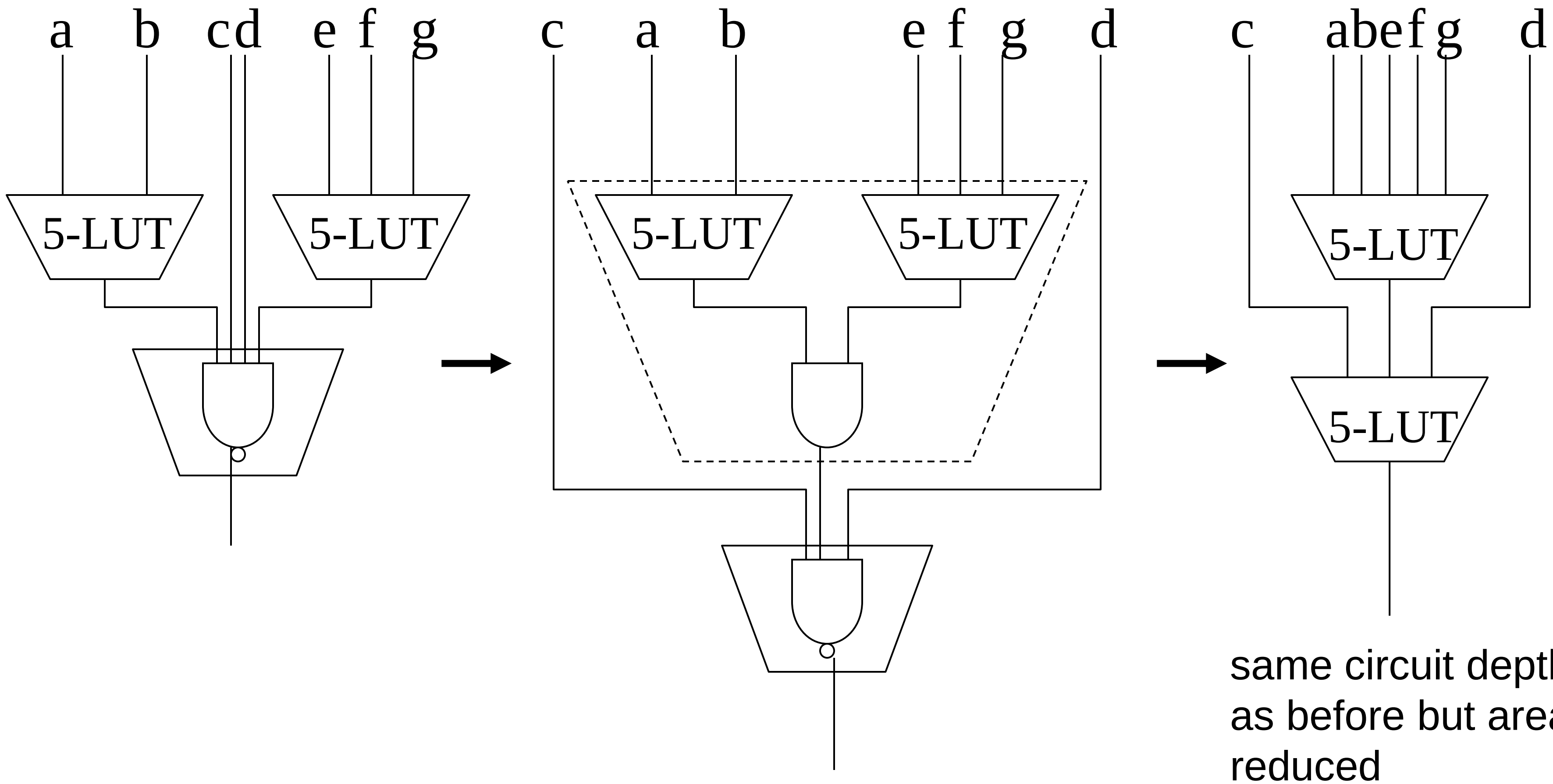
- In the Labeling Phase, we use the network flow method to obtain min-cuts.
- Should we take the *min-area min-cut* or the *max-area min-cut* in order to reduce the total no. of LUTs used?
- How to obtain a max-area min-cut?

Minimization of #LUTs in FlowMap

- In order to reduce the total no. of LUTs used, we may maximize the cut volume during labeling phase.
- For each node, if min-cut size $\leq K$
  - Find a max-area min-cut.
  - Increase the volume of the cut until the node-cut size exceeds K.
  - Use the last K-feasible cut as the solution.

Post-Processing for Area Reduction

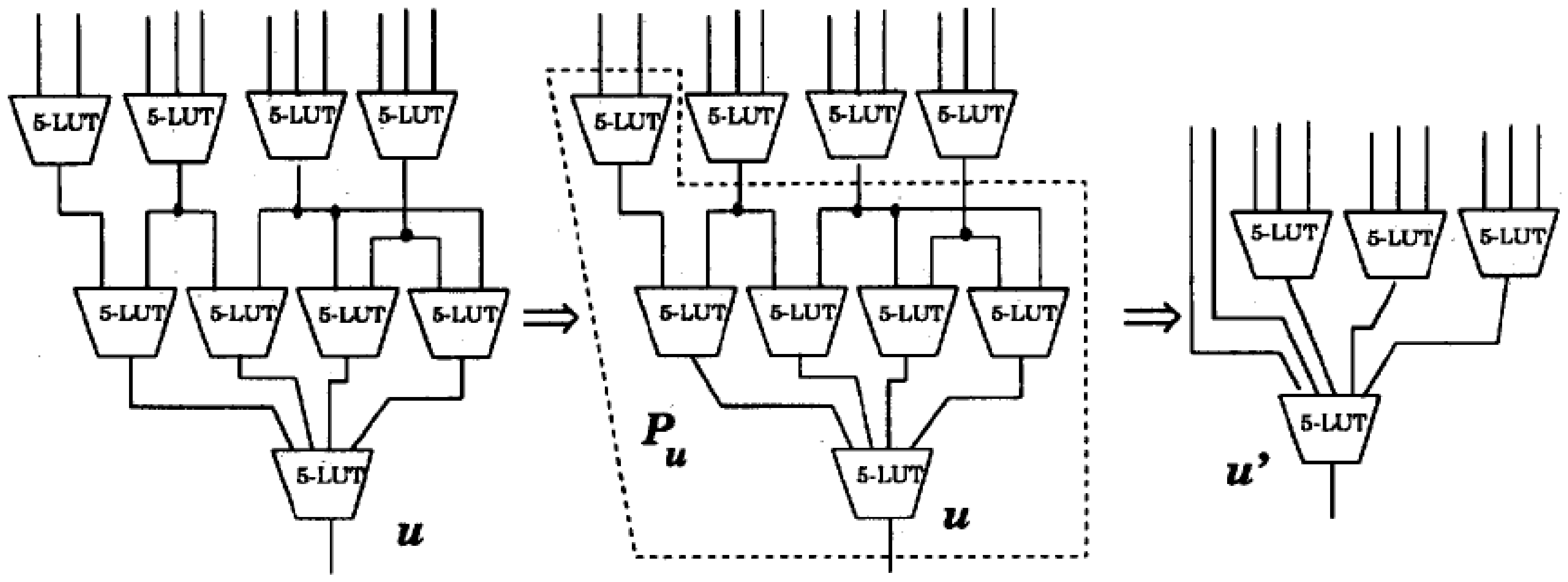
- Gate decomposition



same circuit depth as before but area reduced

Post-Processing for Area Reduction

- Flow pack
  - Any *K*-feasible cone in an initial mapping solution can be replaced by a single *K*-LUT



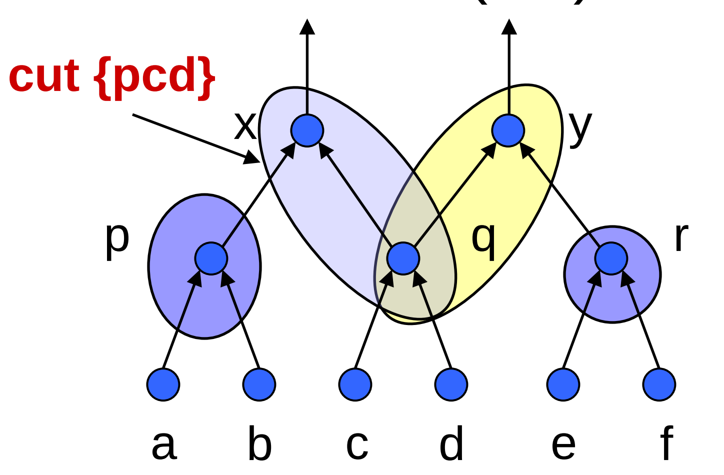
DAOmap: optimal depth mapping with area optimization

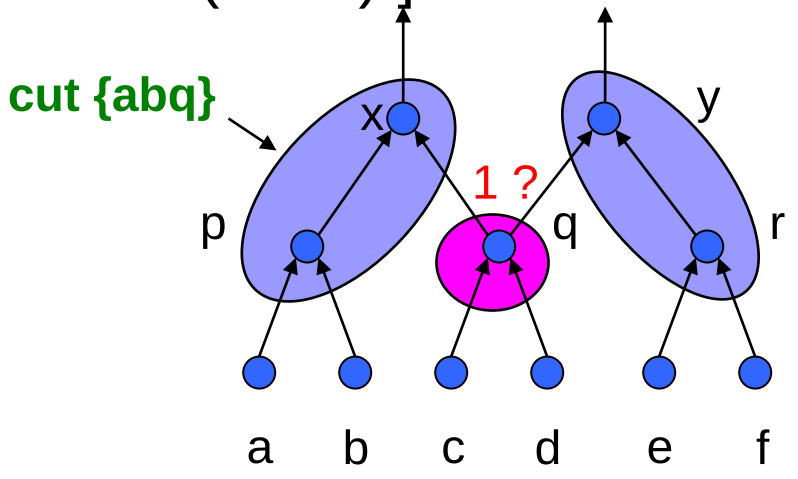
- In a solution of FlowMap, many nodes have extra *slack*.
- For node with slack
  - Pick a cut with *minimum area cost* s.t. timing increase does not exceed the slack
- For node with no slack
  - Among all cuts with *minimum timing cost*, pick one with *minimum area cost*
- Discourage unnecessary logic duplication.
- Re-run mapping phase multiple times and adjust area costs heuristically.

How to Measure Area?

Suppose we use the naïve definition:

Area (cut) = 1 + [Σ area (fanin)]





Area of **cut {pcd}**
= 1 + [1 + 0 + 0]
= 2

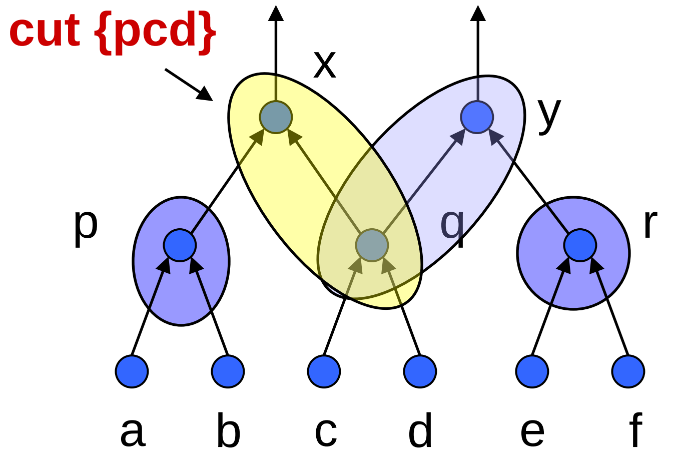
Area of **cut {abq}**
= 1 + [0 + 0 + 1]
= 2

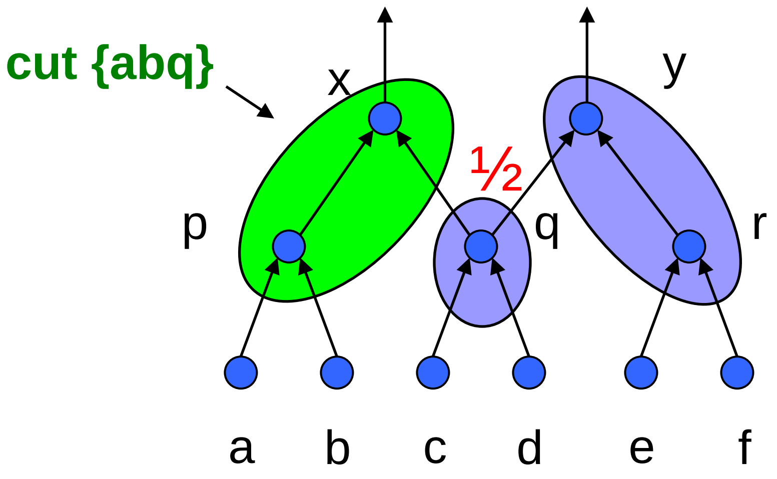
Naïve definition says both cuts are equally good in area

Naïve definition ignores sharing due to multiple fanouts

Area-flow ("Effective Area")

area-flow (cut) = [1 + Σ area-flow (fanin)] / no. of fanouts





Area-flow of **cut {pcd}**
= [1 + 1+ 0 + 0]/1
= 2

Area-flow of **cut {abq}**
= [1 + 0 + 0 + ½]/1
= 1.5

Area-flow recognizes that **cut {abq}** is better

Area-flow of a node is given by the area-flow of its best cut and accounts for sharing

WireMap: mapping for improved routability

- Fewer pin-to-pin connections should make the design easier to place and route
  - Could we come up with a mapping algorithm to minimize the total # of connections in a design?
- Consider *edge-flow* for cut ranking
  - edge-flow (cut) = [cut size + Σ edge-flow (fanin)] / no. of fanouts

WireMap Algorithm

Input: And-Inverter Graph

1. Enumerate all *K*-feasible cuts for each node
2. Compute depth of all cuts and select the minimum one for each node initially
3. Perform area and edge recovery w/o violating timing to select a better cut for each node
   - Heuristic using area flow and edge flow
   - Heuristic using exact local area and exact local edge count
4. Choose the best mapping

Output: Mapped Netlist

Choose cut w/ smallest area-flow, break ties with lower edge-flow

Choose cut with smallest area, break ties with lower no. of edges

References

- “FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs”, *TCAD*, Jan. 1994
- “DAOmap: A Depth-Optimal Area Optimization Mapping Algorithm for FPGA Designs”, *ICCAD*’04
- “WireMap: FPGA Technology Mapping for Improved Routability”, in *FPGA*’08