FPGA Technology Mapping Algorithms

FlowMap

FlowMap

- **Objective:**
  - Minimizing signal delays of mapped designs
    - First polynomial-time depth-optimal algorithm
- **Signal delay:**
  - Delay in the LUTs
  - Interconnection delay
- **LUT placement is not known**
  - → Only LUT delay is considered
  - → Interconnection delay:
    - assumed to be the same for all signals

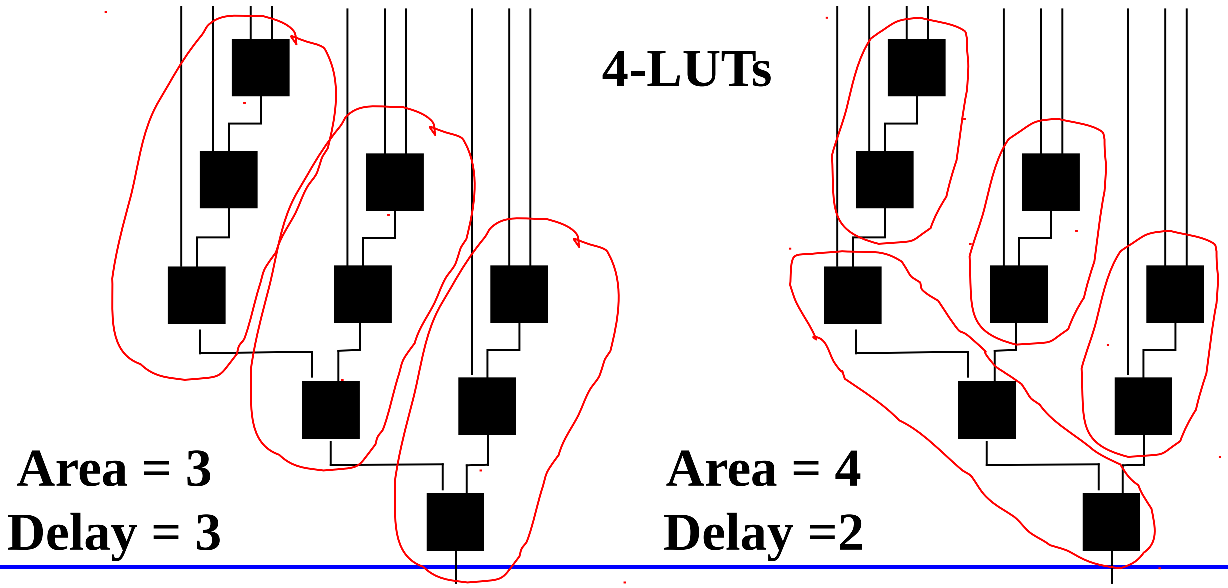
  - → The delay of a signal = the number of LUTs that the signal traverses on a path from input to output

→ **minimization of the depth of the resulting DAG**

- **Two Steps:**
  - Node labelling
  - Node mapping

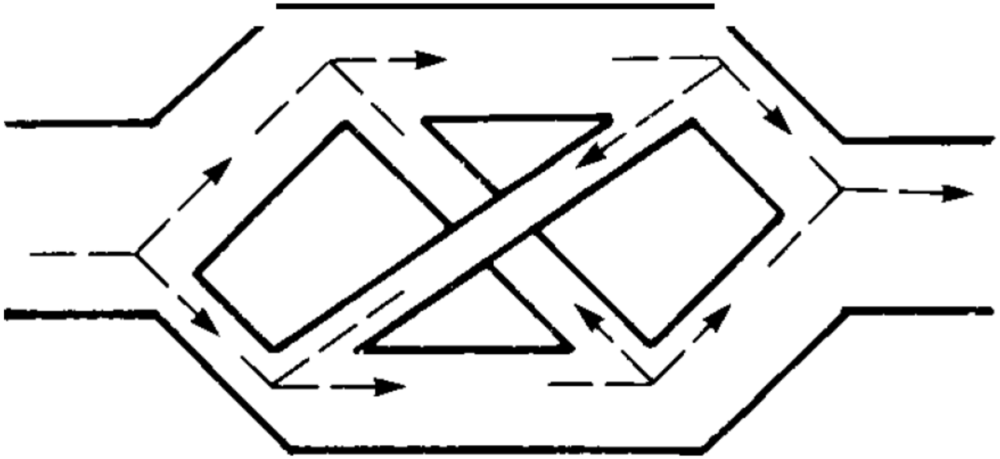
Mapping for Area

- Optimizing for area vs. optimizing for delay
  - Reducing LUTs (area) may increase delay
- Based on network flow problem



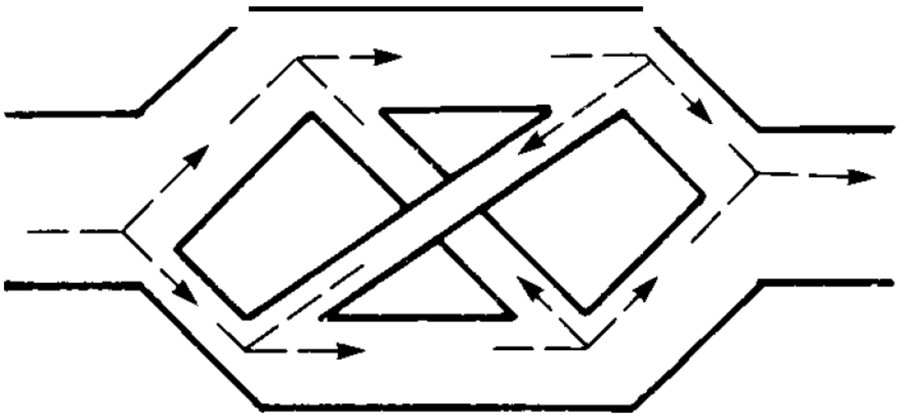
Network Flow Problem

- **Input:**
  - A network with a single source (say, an oil field) and a single destination (say, a large refinery)
  - All of the pipes ultimately connected to them
- **Problem:**
  - What switch settings will maximize the amount of oil flowing from source to destination?



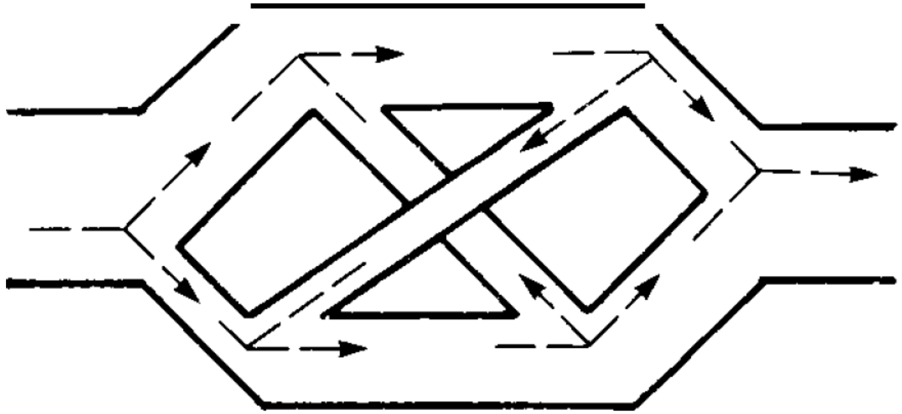
Network Flow Problem

- **Assumptions:**
  - Pipes are of fixed capacity proportional to their size
  - Oil can flow in them only in the direction indicated
  - Switches at each junction control how much of the oil goes in each direction.
  - The system reaches a state of equilibrium (no matter how the switches are set)
    - amount of oil flowing into the system become equal to the amount flowing out



Network Flow Problem

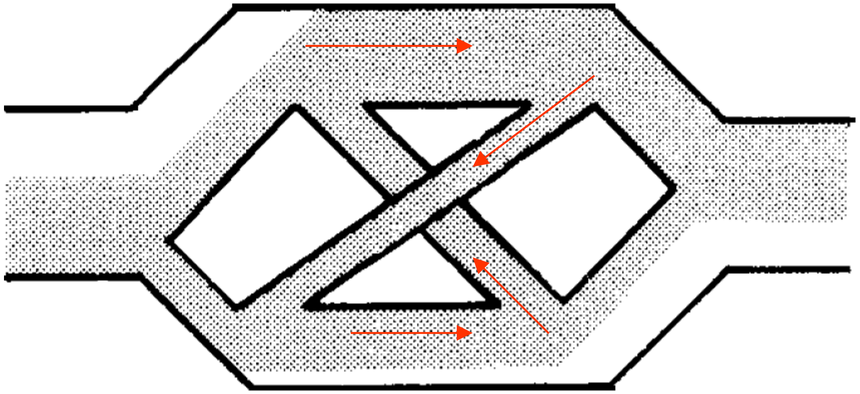
- **Goal:**
  - Maximize this amount of flow



Network Flow Problem

- How can switch settings affect the total flow?

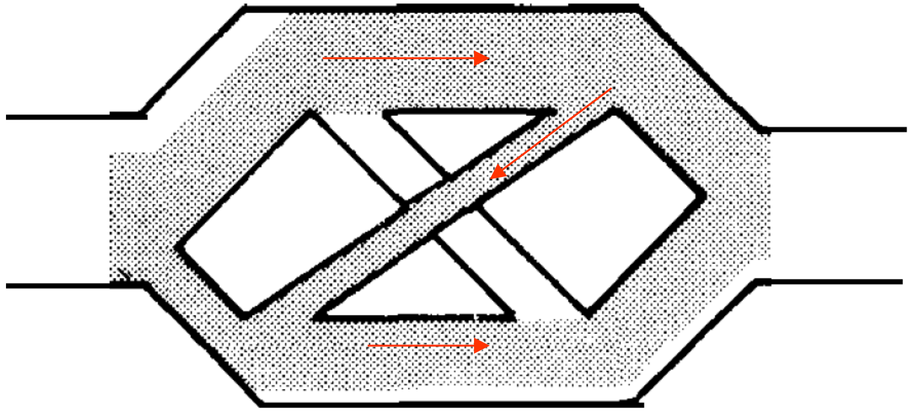
1. Suppose all switches are open.
   - → Diagonal pipes are full



   - ~ half of the input pipe capacity is used

Network Flow Problem

- How can switch settings affect the total flow?
  2. Suppose upward pipe is shut-off



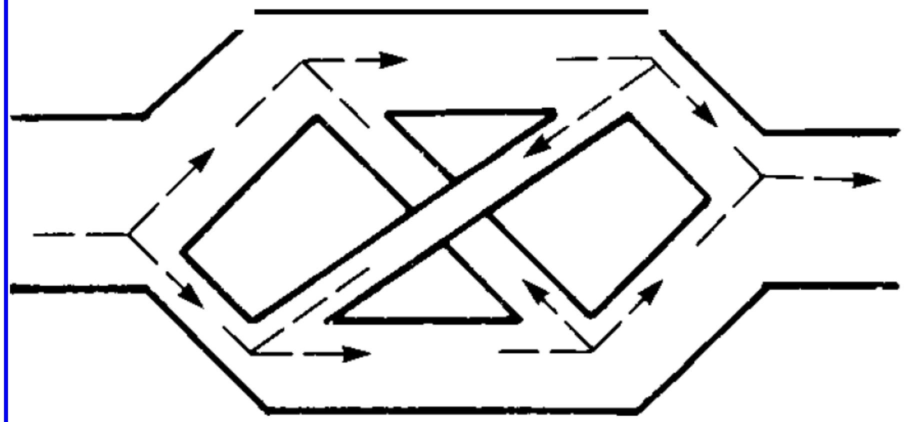
  - Substantial Increase in total flow into and out of the network.

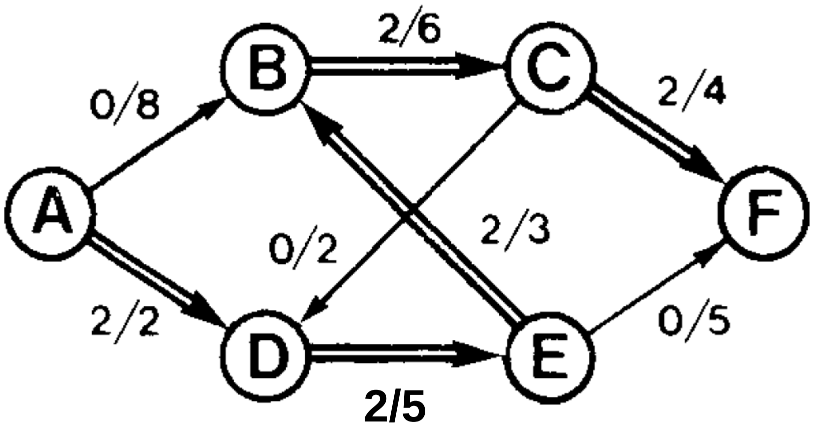
Graph Model of Network Flow

- **Graph model:**
  - Weighted directed graph
  - Nodes:
    - Source (with no input edge)
    - Sink (with no output edge)
    - Pipe junctions
  - Edges:
    - Pipes
    - Directions: oil flow
    - Weights:
      - (a) pipe capacities
      - (b) flow on each edge (≤ capacity)
    - Flow in a node = Flow out of it
- **Network flow problem:**
  - Maximize flow out of the output node

Graph Model of Network Flow

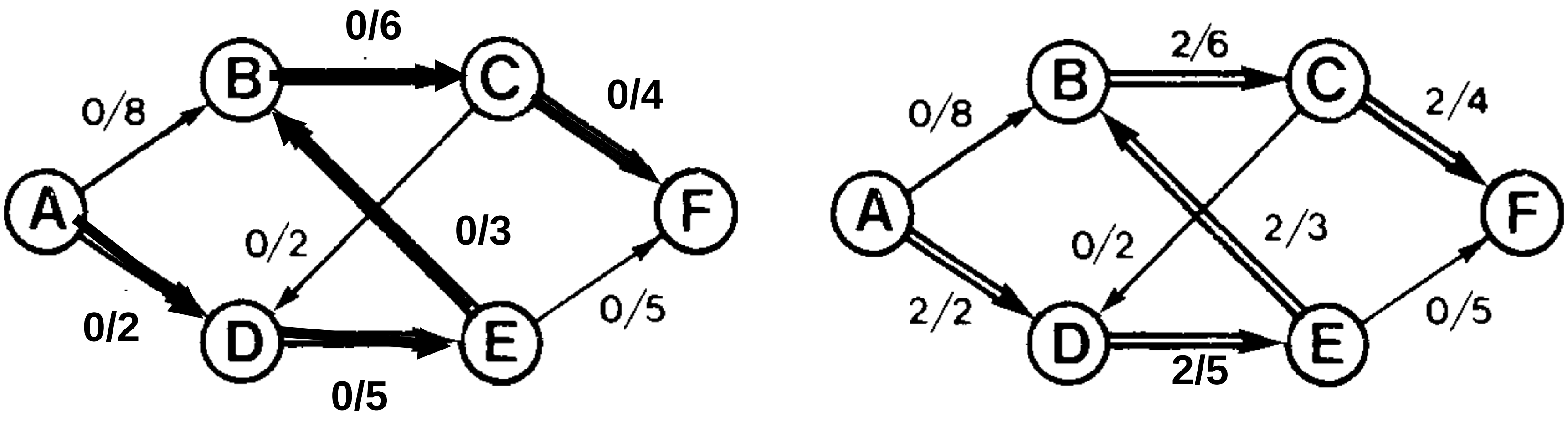
- **Graph model:**
  - Edges can be undirected:
    - (x → y), capacity *s*, flow *f* =
    - (y → x), capacity -*s*, flow -*f*





Ford-Fulkerson Method

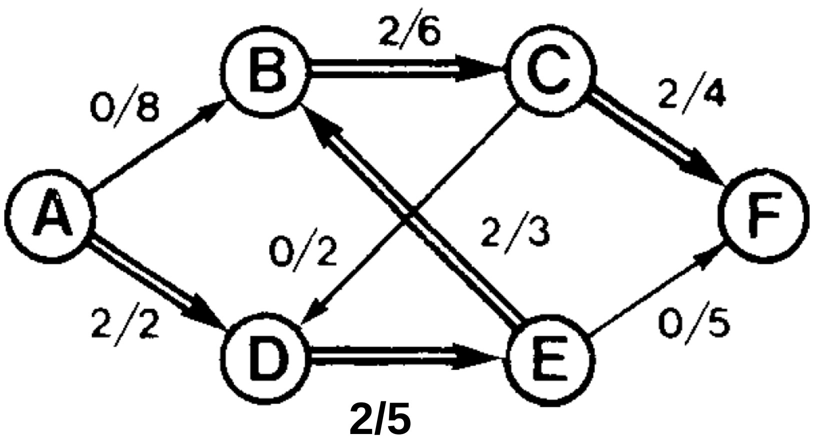
- **FF Algorithm:**
  - ➢ Start with a zero flow
  - ➢ Try to increase flow repeatedly
  - ➢ Repeat until no increase possible
    - → Maximum flow found

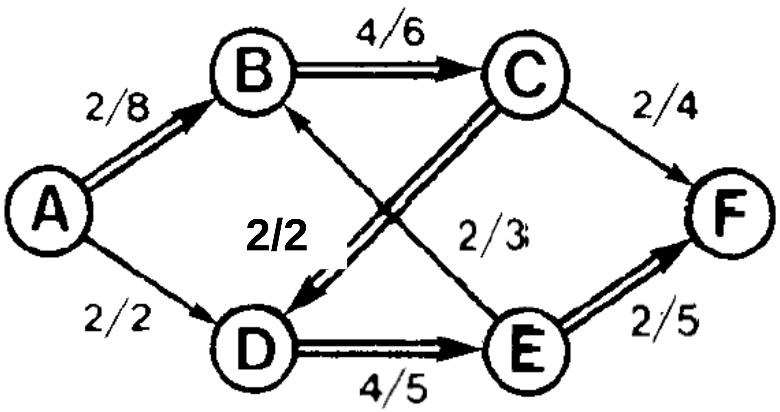


➢ Increase flow along the path ADEBCF

Ford-Fulkerson Method

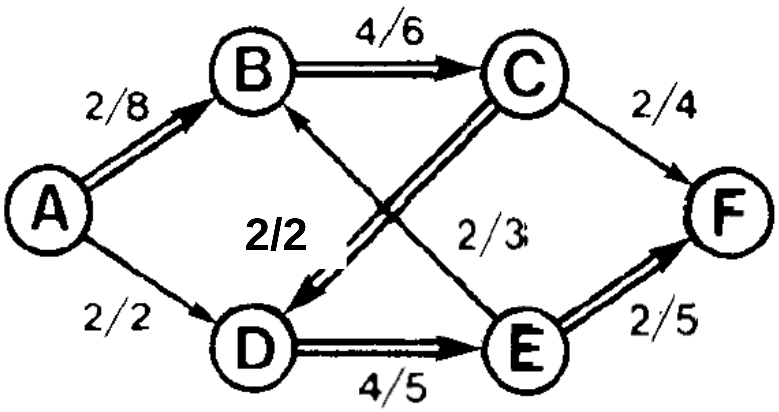
- Increase flow along the path ABCDEF

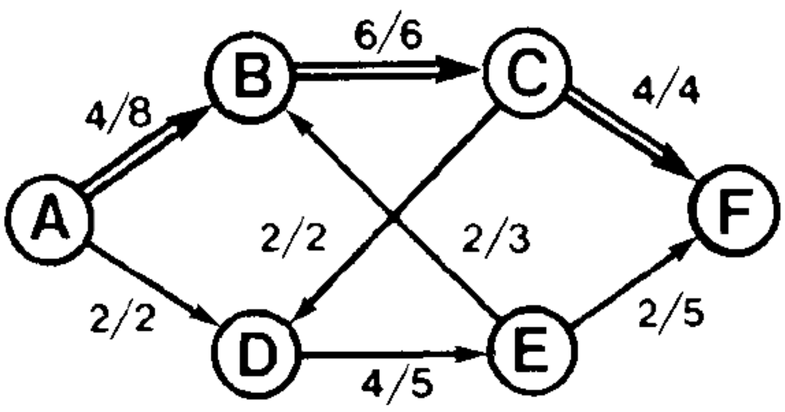




Ford-Fulkerson Method

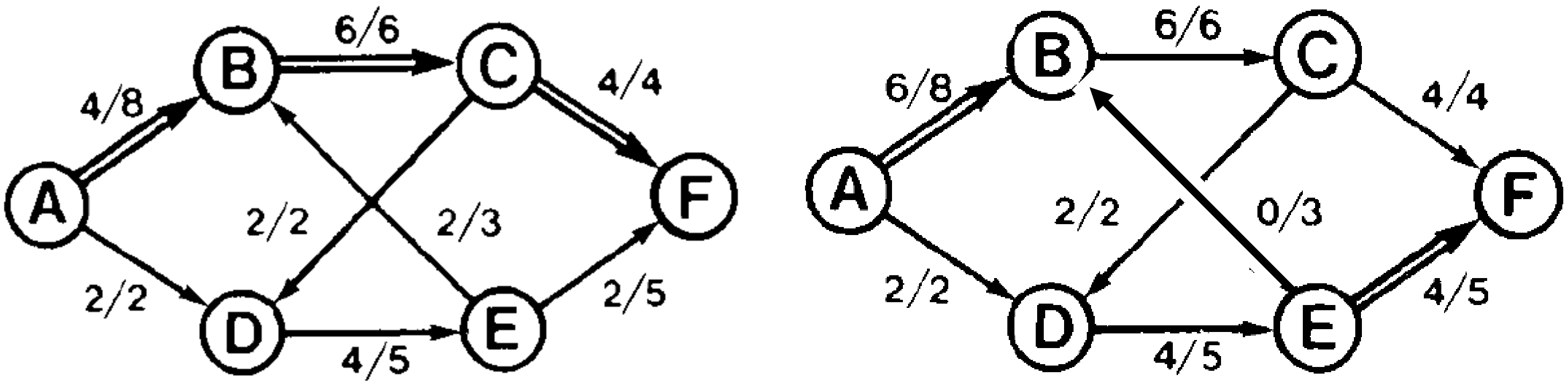
- Increase flow along the path ABCF





Ford-Fulkerson Method

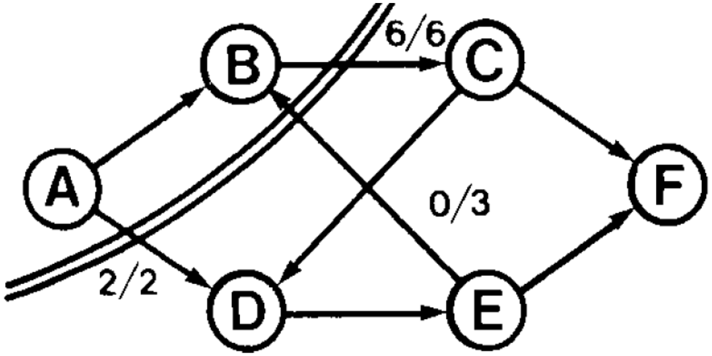
- Increase flow along the path ABEF



- **Condition to stop:**
  - At least one of the forward edges along the path becomes full or at least one of the backward edges along the path becomes empty

Maxflow-Mincut Theorem

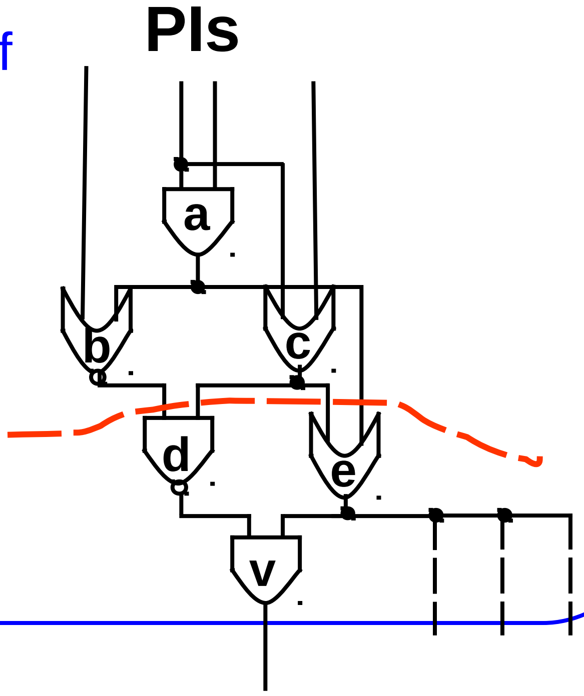
- **Cut:**
  - Go through the network (from source to sink) and find the first full forward edge or empty backward edge on every path.



- **Maxflow-Mincut Theorem:**
  - Whenever the cut flow equals the total flow, we know not only that the flow is maximal, but also that the cut is minimal.
    - Count only the forward edges in cut.

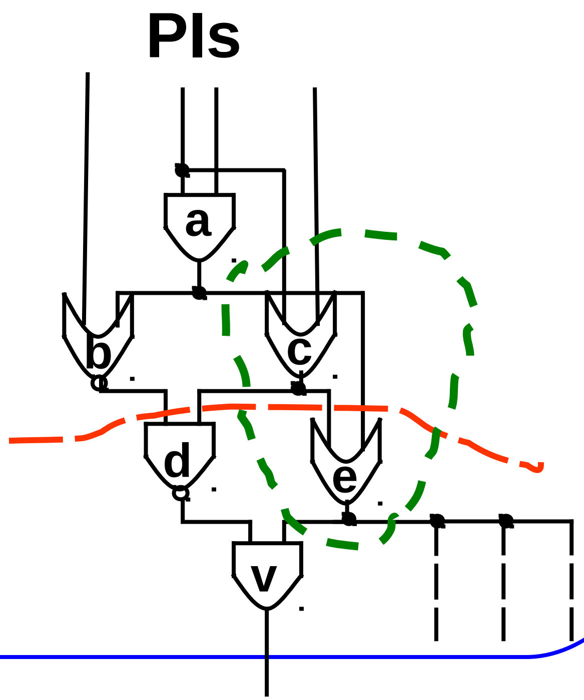
Basics of Network Flow

- **FlowMap: a network flow-based method.**
- **Basics of network flow:**
- **Given a network $N = (V, E)$ (a graph)**
  - Cut: a partition (X, X_b) of N with source $s \in X$ and target $t \in X_b$
  - Node cut-size $n(X, X_b)$ of a cut (X, X_b): # of nodes in X adjacent to some nodes in X_b
  - K-feasible cut: iff $n(X, X_b) \leq K$
  - Edge cut-size $e(X, X_b)$: weighted sum of crossing edges



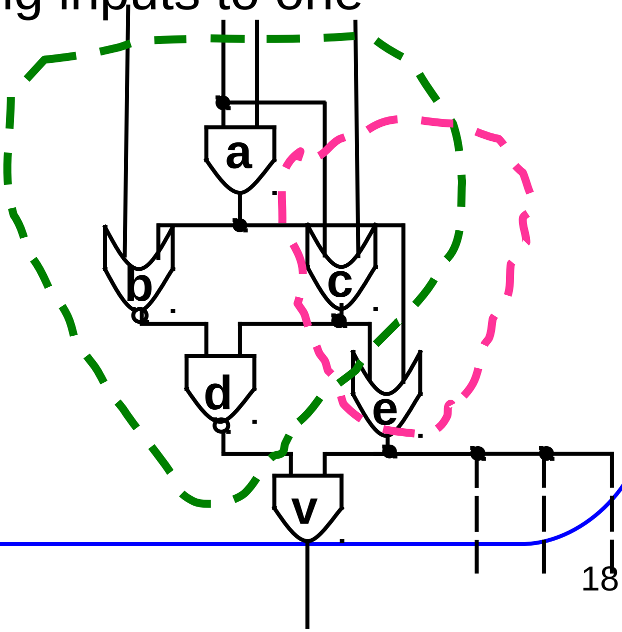
Basics of Network Flow

- fanin cone O_v rooted at node v: a sub-network consisting of v and some of its predecessors, such that for any node $u \in O_v$, there is a path from u to v that lies entirely in O_v
- Label of a node t: the depth of the optimal LUT which implements t in an optimal mapping of the sub-graph C_t of N
  - C_t is the cone at t.
- Height $h(X,X_b)$ of a cut (X,X_b): the maximum label in X
- Volume $vol(X,X_b)$: # of nodes in X ($|X|$)



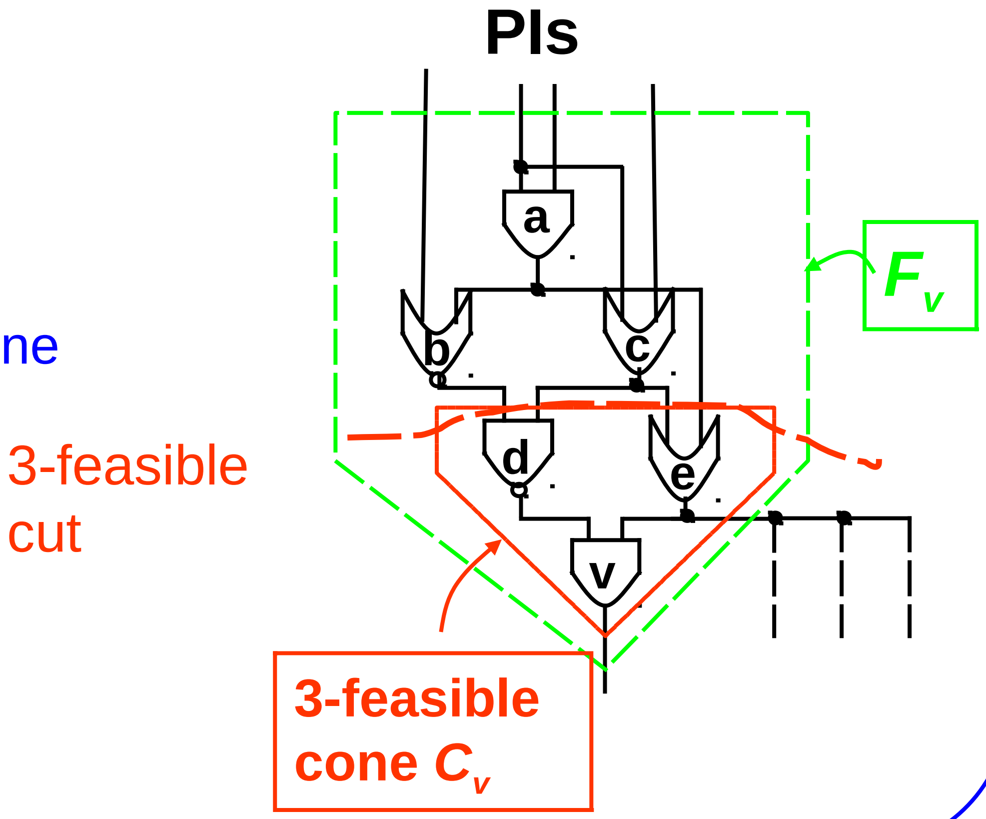
Basics of Network Flow

- Maximum fan-in cone F_v: The largest cone rooted at v (Largest O_v)
  - Consisting of all the predecessors of *v.*
- MFFCv (Maximum fanout-free cone):
  - For each node *v*, there is a *unique* maximum fanout-free cone which contains every fanout-free cone rooted at *v*.
- *input*(C_v):
  - Set of distinct nodes outside of O_v supplying inputs to one or more gates in O_v.



Basics of Network Flow

- ✂ → O_v is K-feasible if $|\text{input}(O_v)| \leq K$.
- **Cut:**
  - ➢ partition (X, X_b) of the fanin cone F_v of v such that X_b is a cone of v
- **Cutset of the cut:**
  - ➢ input(X_b)
- ***K*-feasible cut (*K*-cut):**
  - ➢ if X_b is a K-feasible cone

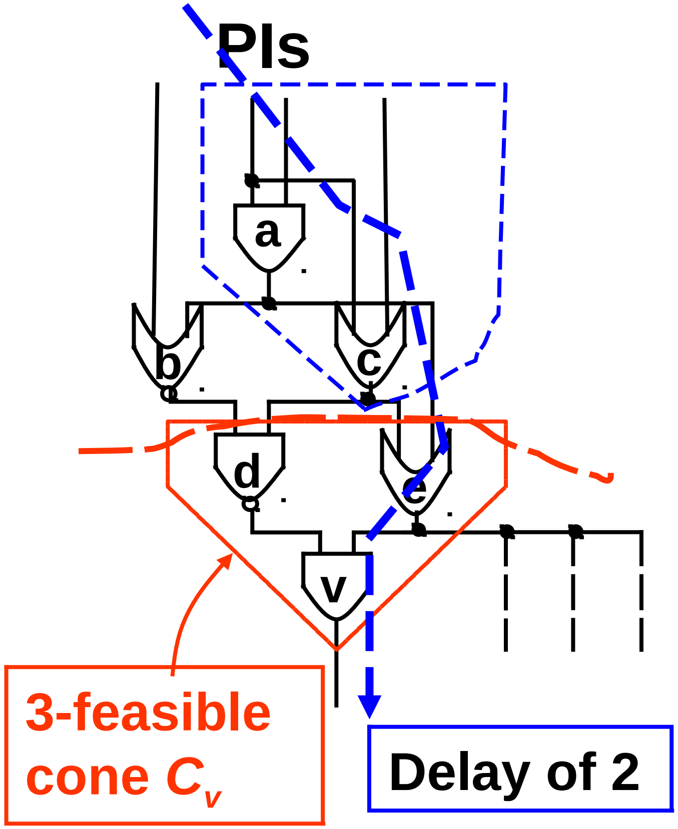


[Chen04]

Basics of Network Flow

- ***K*-LUT:**
  - X_b is a K-LUT that implements v with the inputs in the cutset.
- We use cuts, cutsets, cones, and LUTs interchangeably
- ***t-bounded* Boolean network:**
  - if $|\mathrm{input}(v)| \le t$ for each node v
  - For Flowmap, the input network must be 2-bounded
    - Otherwise, it should be decomposed before Flowmap

Basics of Network Flow: Example



FlowMap: Basic Approach

- **Node labelling:**
  - Labels every node in a topological order
  - → Each node is processed after all its predecessors
    - Label: minimum possible depth of the node in any mapping solution
  - ➢ Dynamic Programming:
    - Starting from PI nodes, compute node labels in topological order:
    - Compute the label of a node based on labels of its predecessors
- **Labels of PO nodes:**
  - ➢ Depth of the optimal mapping solution

algorithm FlowMap

/* phase 1: labeling network */

for each PI node v **do**

$\quad l(v) := 0;$

T := list of non-PI nodes in topological order;

while T is not empty **do**

- remove the first node t from T;
- construct the network N_t;
- let $p = \max\{l(u) : u \in input(t)\}$;
- transform N_t into N'_t by collapsing all nodes in N_t with label p into t;
- transform N'_t into N''_t as follows:
  - split every node in $\{ x : x \in N'_t, x \neq s, x \neq t \}$ into two and connect them with a bridging edge of capacity 1;
  - assign all non-bridging edges capacity ∞;
- compute a cut $(X'', \bar{X}'')$ in N''_v s.t. $e(X'', \bar{X}'') \leq K$ using the augmenting path algorithm;
- **if** $(X'', \bar{X}'')$ is not found in N''_t **then**
  - $\bar{X}_t := \{t\}; \quad l(t) := p + 1$
- **else**
  - induce a cut $(X, \bar{X})$ in N_t from the cut $(X'', \bar{X}'')$ in N''_t;
  - $\bar{X}_t := \bar{X}; \quad l(t) := p$
- **endif**

endwhile;

/* phase 2: generate K-LUTs */

FlowMap Algorithm

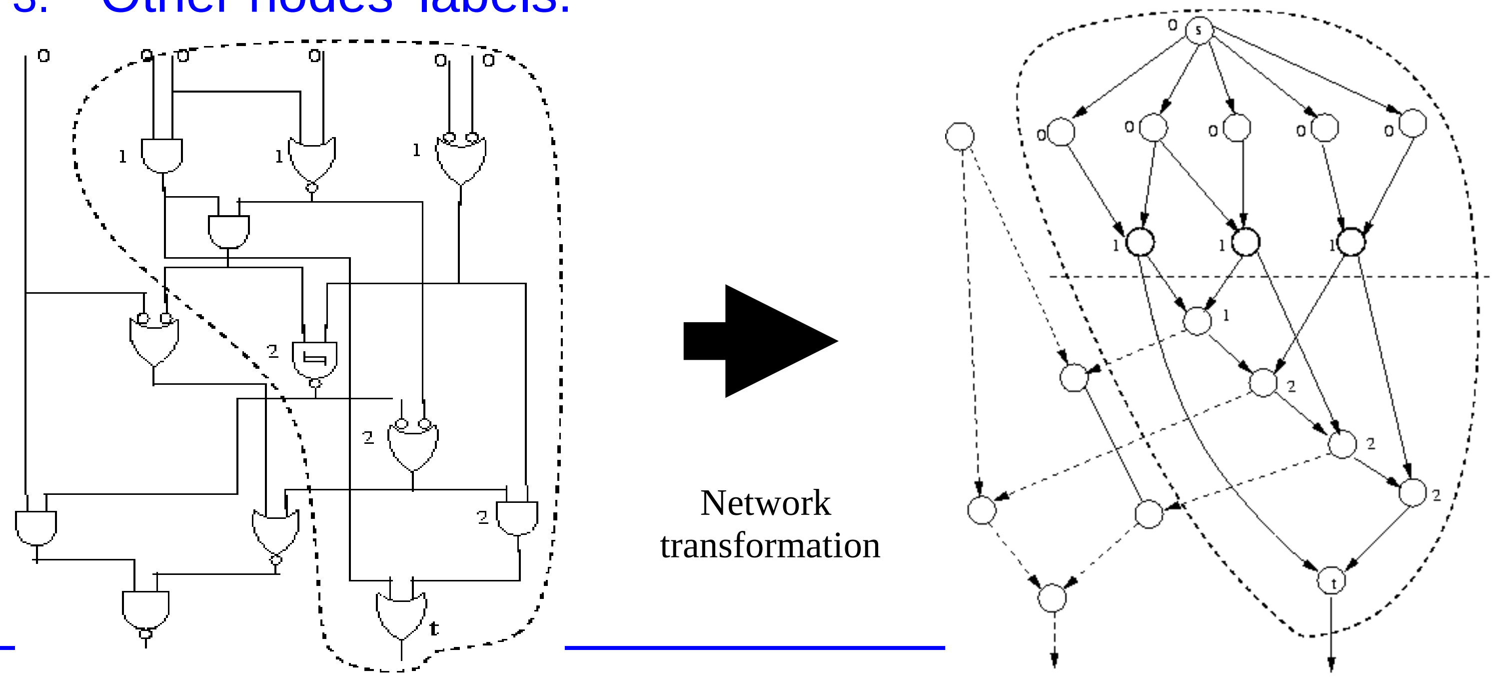
/* phase 2: generate K-LUTs */
L := list of PO nodes;
while L contains non-PI nodes **do**
  take a non-PI node v from L;
  generate a K-LUT v' to implement the function of v
    such that $input(v') = input(\overline{X}_v)$;
  $L := (L - \{ v \}) \cup input(v')$
endwhile
end-algorithm;

FlowMap: Node Labelling

1. Node labelling:

- **Steps:**

1. For a given node t, the cone C_t is transformed into a network N_t :
   - ➢ Inserting a source node s whose output is connected to all inputs of N_t.
2. *l(primary input)* = 0
3. Other nodes' labels:



FlowMap: LUT Mapping

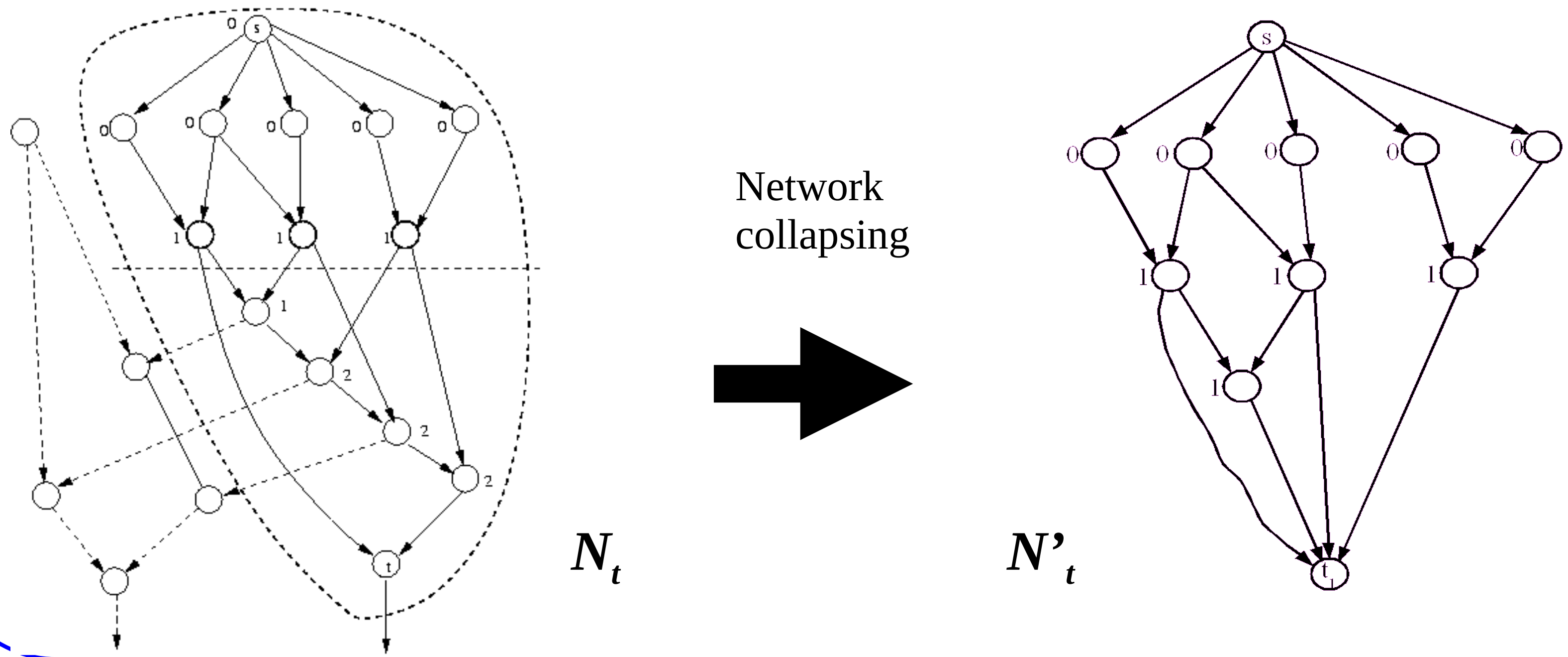
- **Lemma:**
  - If p is the maximum label in *input(t)*, then

    $l(t) = p$ OR

    $l(t) = p+1$
- **Algorithm:**
  - Check whether there is a K-feasible cut (X, X_b) of height $p-1$ in N_t.
  - If yes, then
    - $l(t) \leftarrow p$ and the node t will be packed (in the second phase) in a common LUT with the nodes in X.
  - If no, then
    - the minimum height of the K-feasible cuts in N_t is p and
    - $N_t - \{t\}$, $\{t\}$ is such a cut.
    - $l(t) \leftarrow p + 1$ and
    - a new LUT will be used for t.
- **New Problem:**
  - How to find out if a network has a K-feasible cut with a given height h.

Network Collapsing

- *Network Collapsing:*
  - collapses all the nodes in N_t with max-label = p together with t in a new node t'.
- **Lemma:**
  - if N'_t has a *K-feasible* cut, N_t has a *K-feasible* cut of height $p - 1$



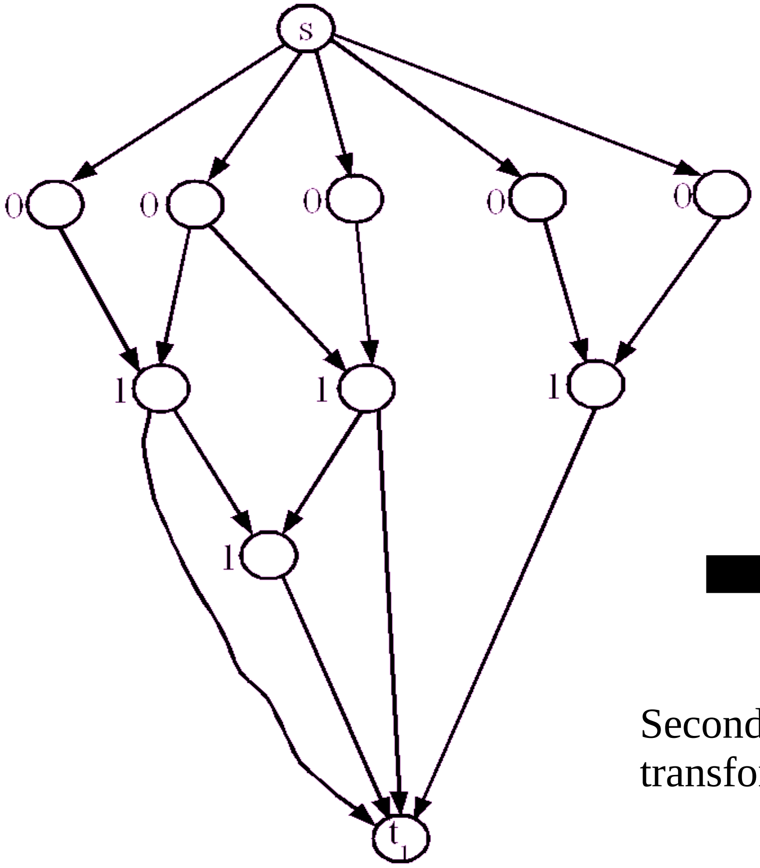
Node Splitting

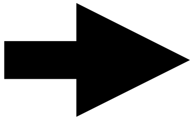
- Finding min height K-feasible cut in N_t is reduced to finding K-feasible cut in N'_t
- **Question:**
  - How to know if there is a K-feasible cut in N'_t?
- **Answer:**
  - Network flow algorithms
  - Problem:
    - They use edge cut optimization
  - Solution:
    - → ***Node splitting***

Node Splitting

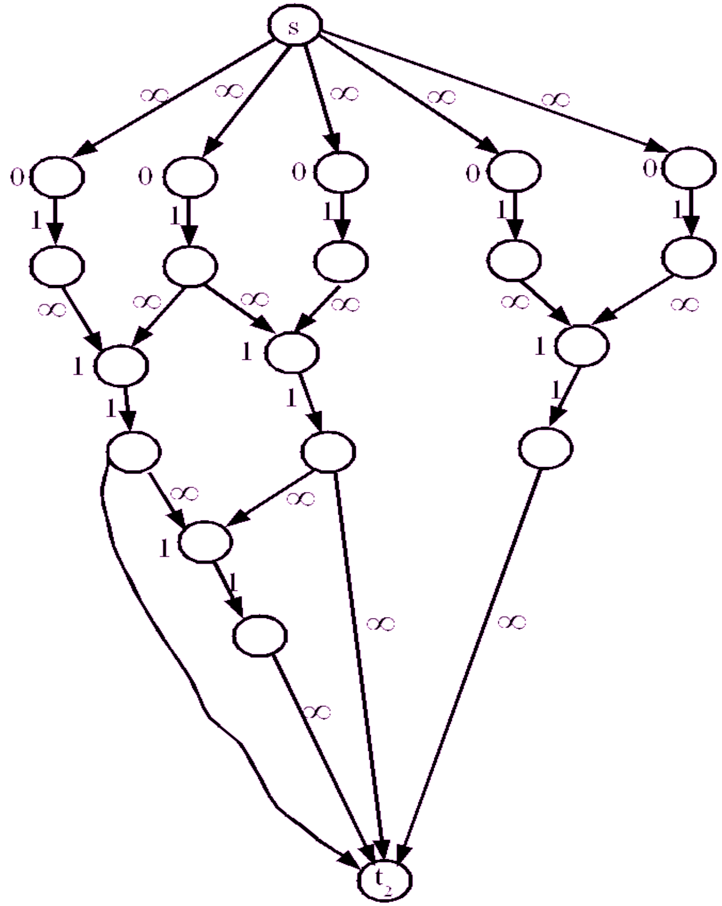
- **Transform N'_t to N''_t:**
  1. For each node v in N'_t (except s and t')
     1. Introduce v_1 and v_2
     2. Connect them by bridging edge (v_1, v_2)
  2. s and t' appear in N''_t too.
     1. For each (s, v), create a (s, v_1)
     2. For each (v, t'), create a (v_2, t')
  3. For each (u, v) in N'_t ($u \neq s$ and $v \neq t'$),
     1. Create (u_2, v_1)
     2. Set capacity:
        - 1 for bridging edges
        - ∀ ∞ for non-bridging edges

Node Splitting





Second
transformation

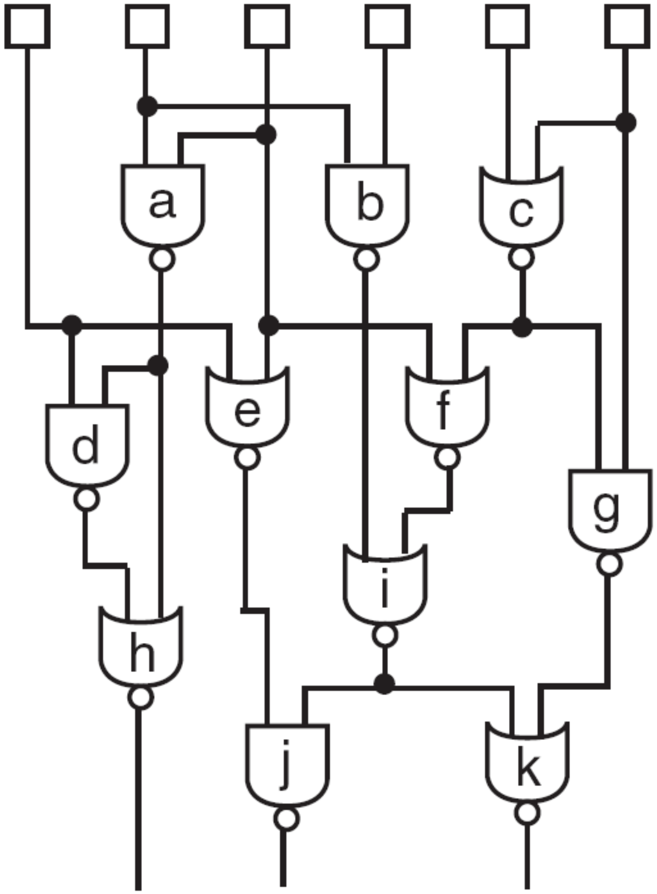


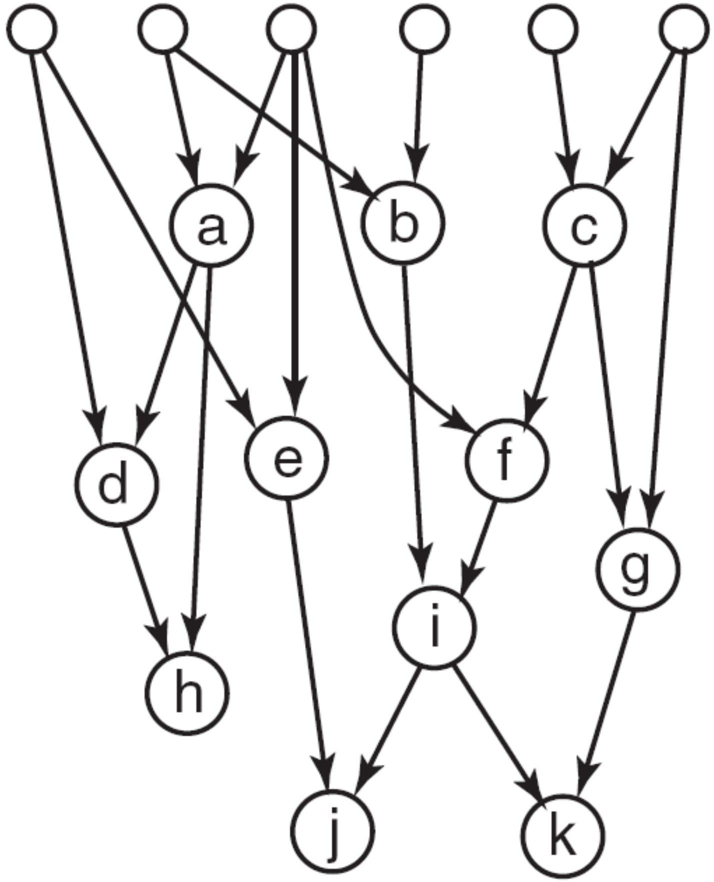
Node Splitting

- ***N'_t* to *N''_t* transformation:**
  - Ensures that if a cut exists in N''_t with capacity < K, then no edge with infinite capacity will be a crossing one.
  - Only bridging edges are crossing the cut
    - A LUT may have fanout > 1
      - → Min-cut in N'_t may not work properly
- **Lemma:**
  - if N''_t has a cut with cut size ≤ *K*, N'_t has a *K-feasible* cut.

Example

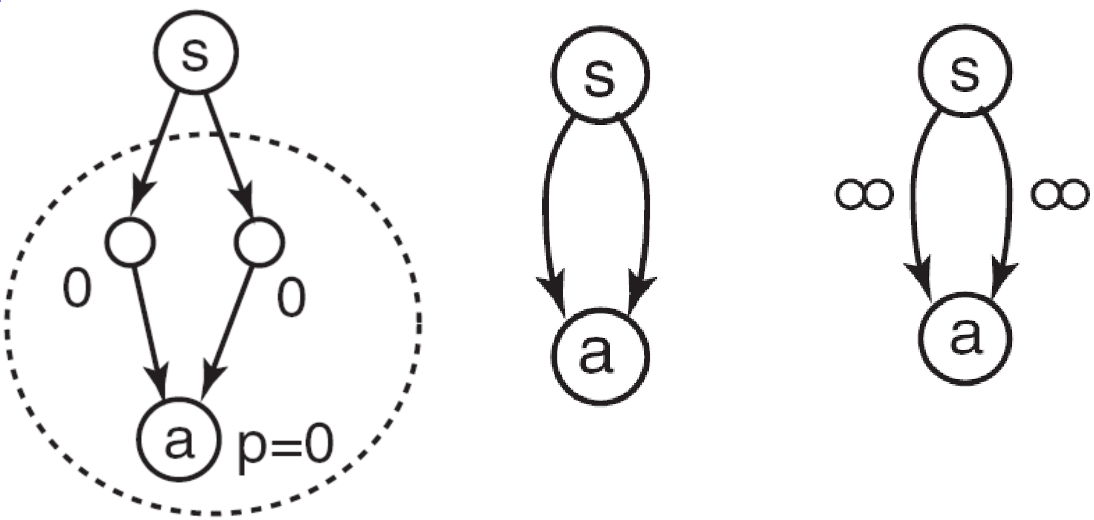
- **Example:**
  - K = 3

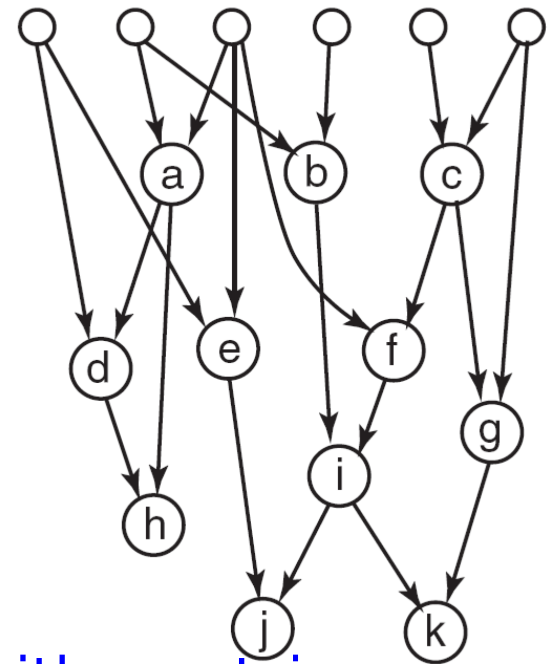




Example

- $l(i) = 0$ for all PIs
- $p = 0$
- Topological order: {a, b, c, d, e, f, g, h, i, j, k}

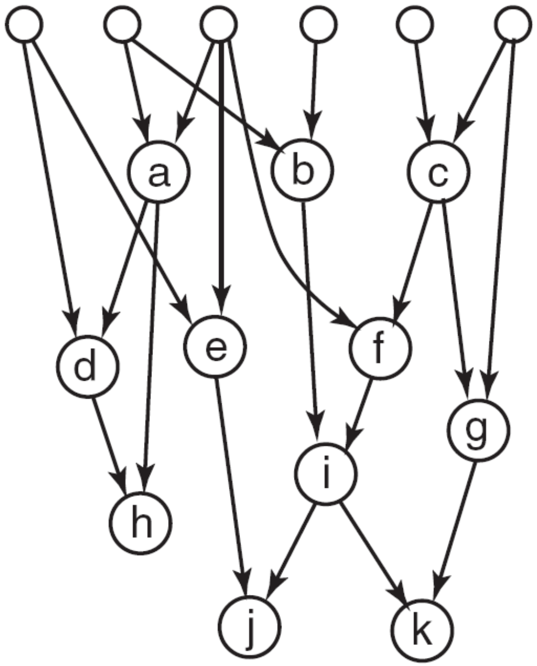




- Not possible to find a cut in N''_a with a cutsize smaller or equal to $K = 3$
  - ➔ $X_b = \{a\}$
  - $l(a) = p + 1 = 1$.

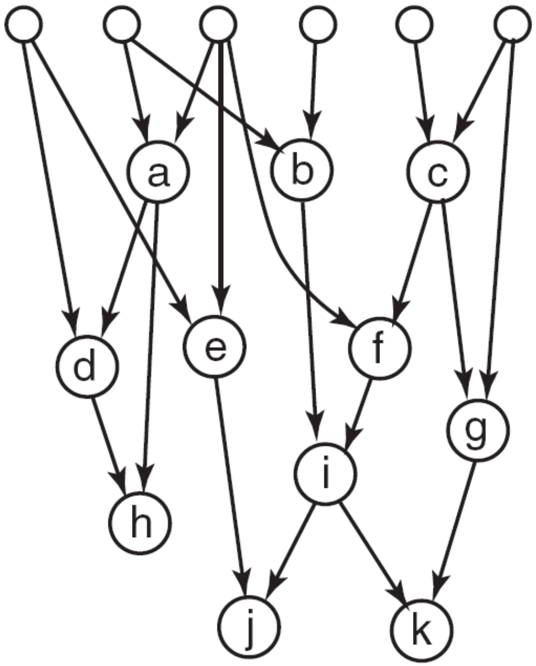
Example

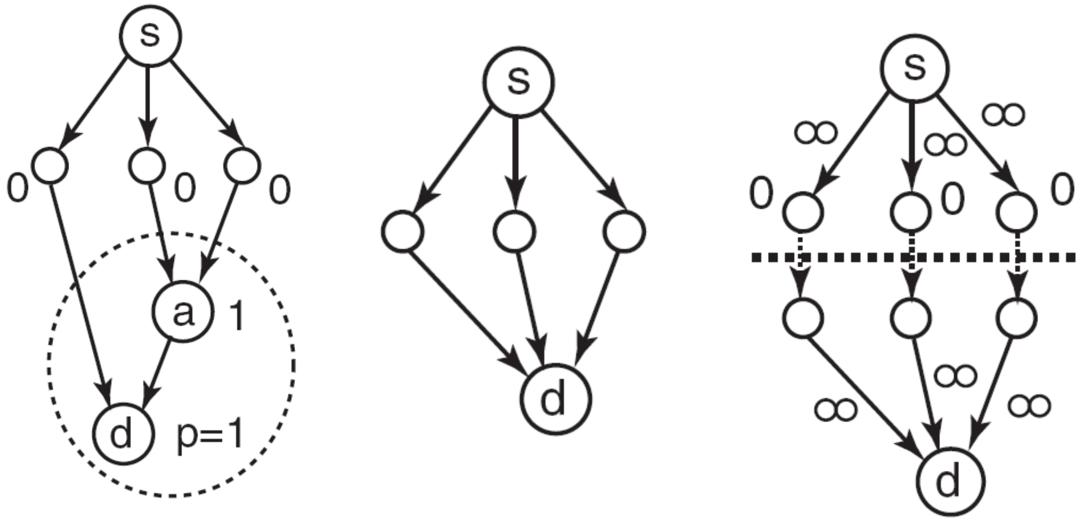
- Node *b* and *c*:
  - Similar to the case for node *a*,
- **Node b:**
  - $X_b = \{b\}$,
    - $I(b) = 1$
- **Node c:**
  - $X_b = \{c\}$
    - $I(c) = 1$



Example

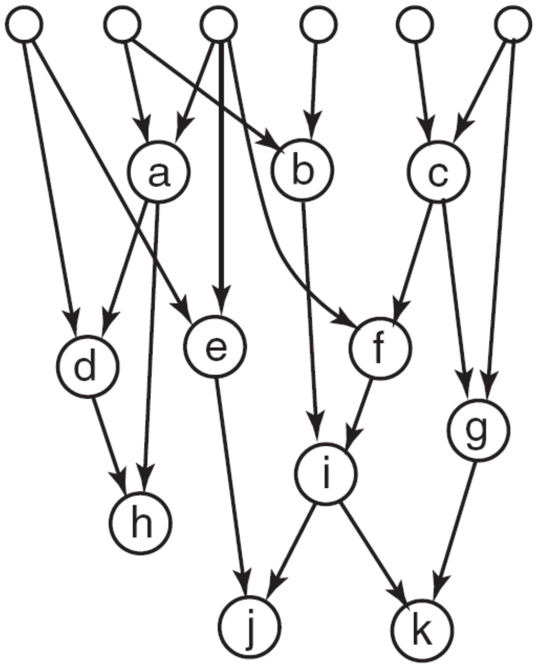
- **Node *d*:**
  - $p = 1$
  - Max flow (min-cut) = 3
  - $X_b = \{a, d\}$
  - $l(d) = p = 1$





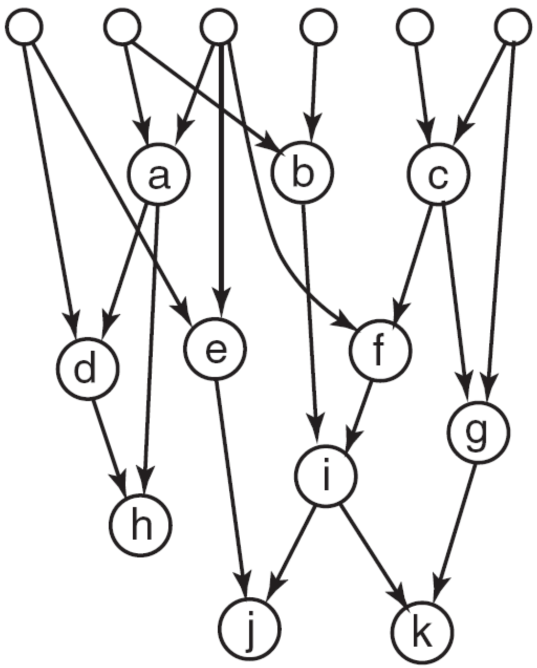
Example

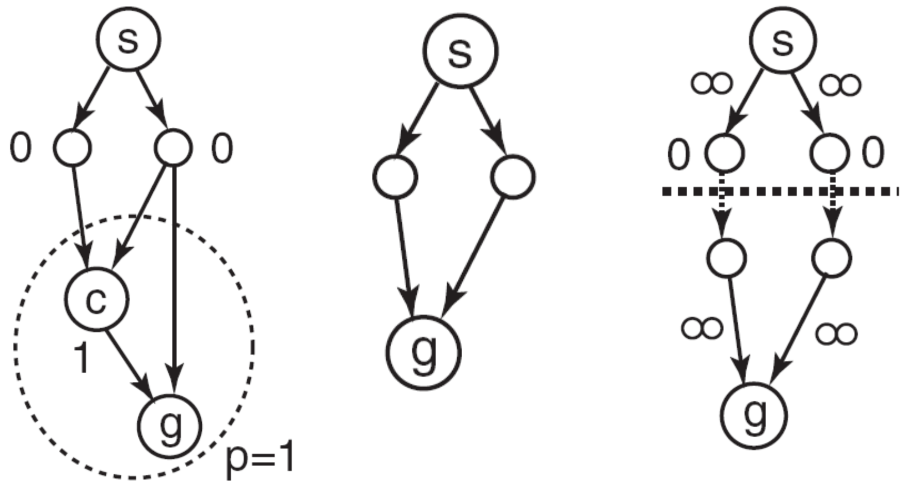
- **Node *e*:**
  - Similar to *a*
  - $X_b = \{e\}$
  - $l(e) = 1$
- **Node *f*:**
  - similar to *d*
  - $X_b = \{c, f\}$
  - $l(f) = 1$



Example

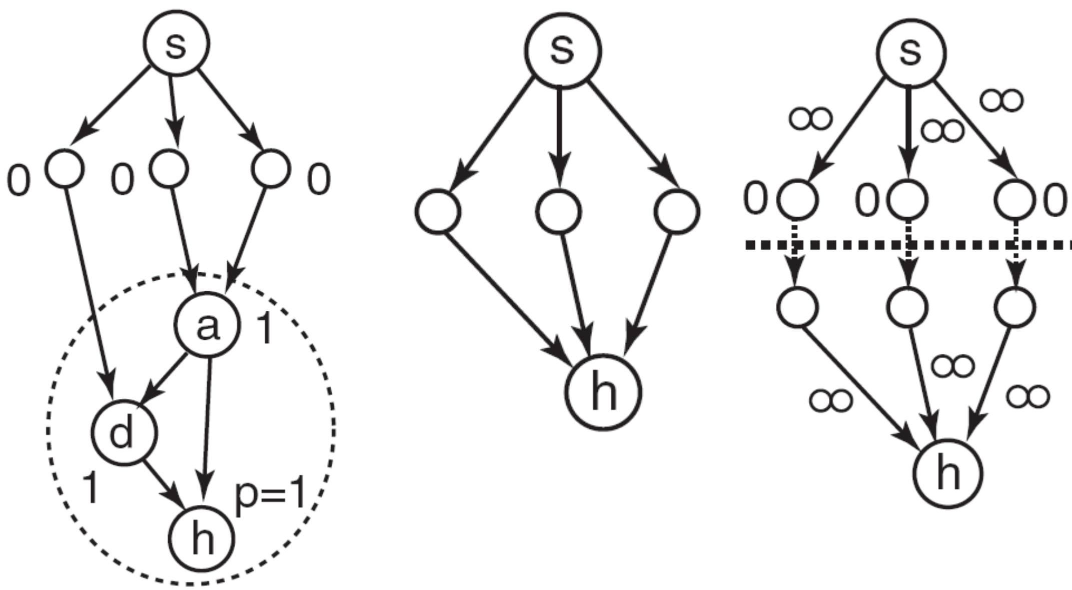
- **Node *g*:**
  - $X_b = \{c, g\}$
  - $l(g) = p = 1$

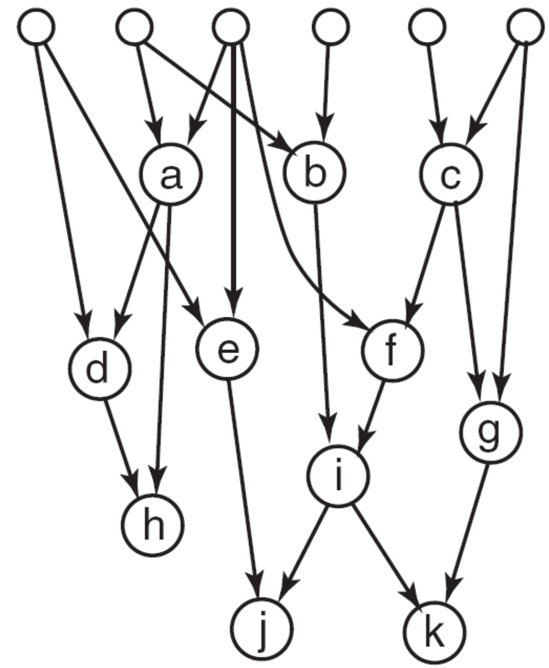




Example

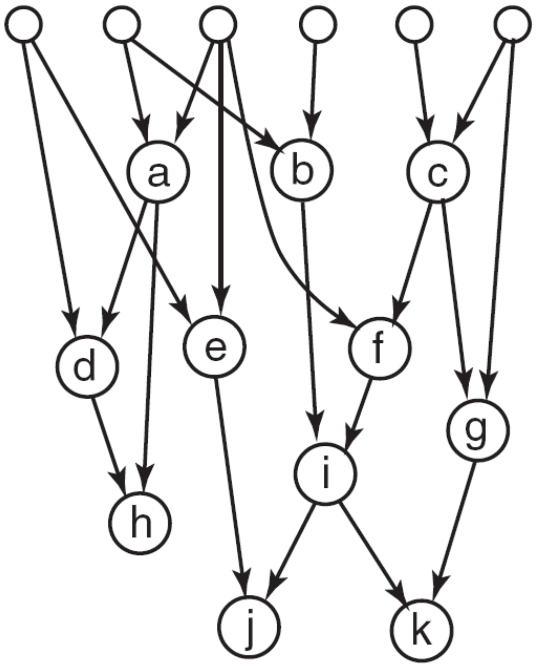
- **Node *h*:**
  - $X_b = \{a, d, h\}$
  - $l(h) = l(d) = 1$

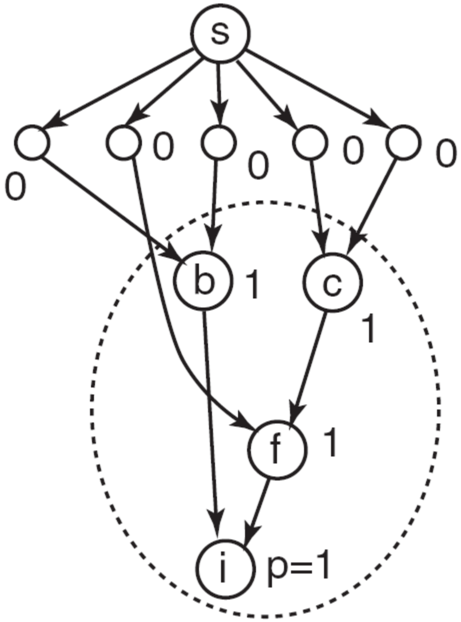


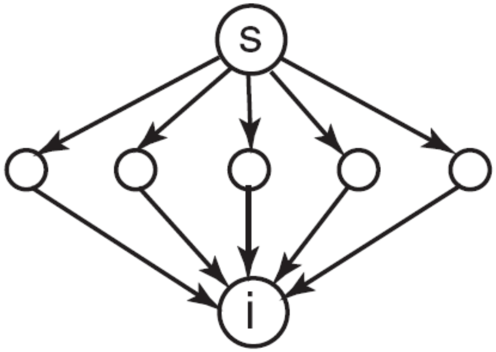


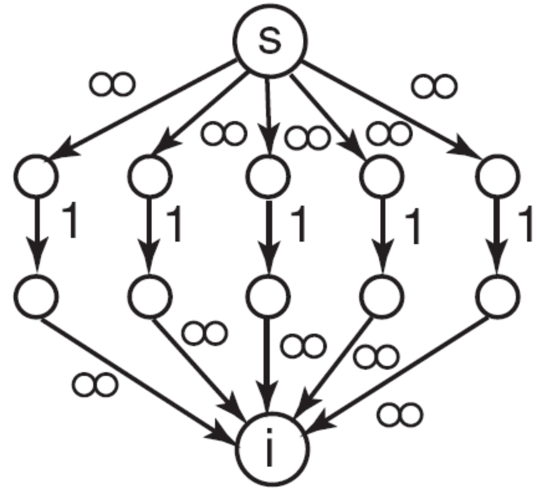
Example

- **Node *i*:**
  - N''_i does not contain a K-feasible cut.
  - $X_b = \{i\}$
  - $l(i) = p + 1 = 2$



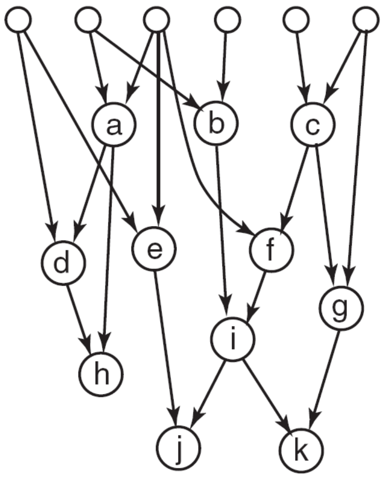


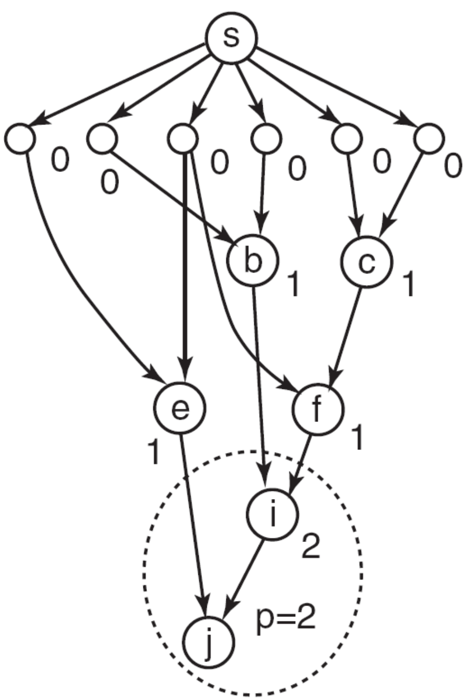


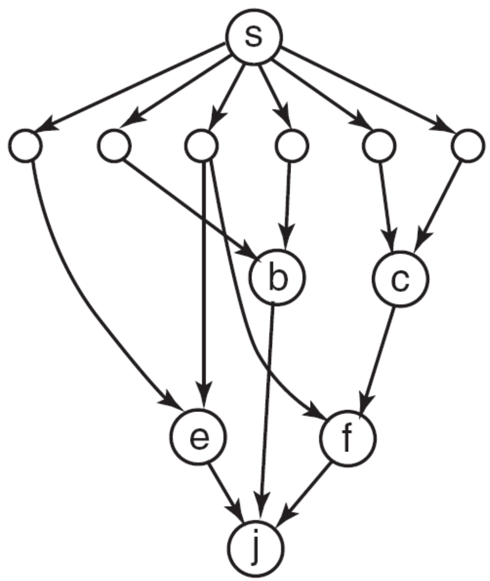


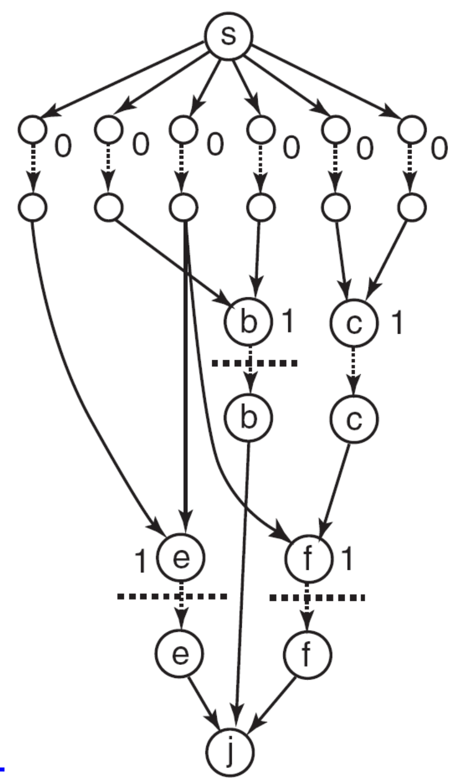
Example

- **Node *j*:**
  - Only one K-feasible cut in N''_j
  - Its height is 1.
  - $X_b = \{i, j\}$
  - $l(j) = p = 2$



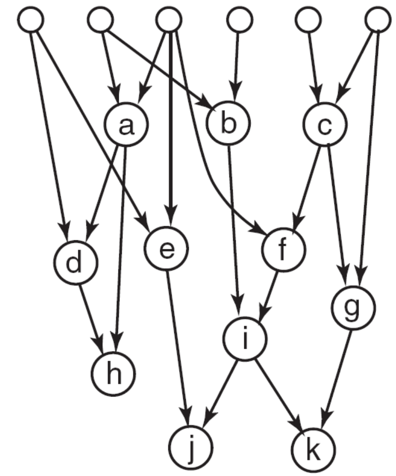


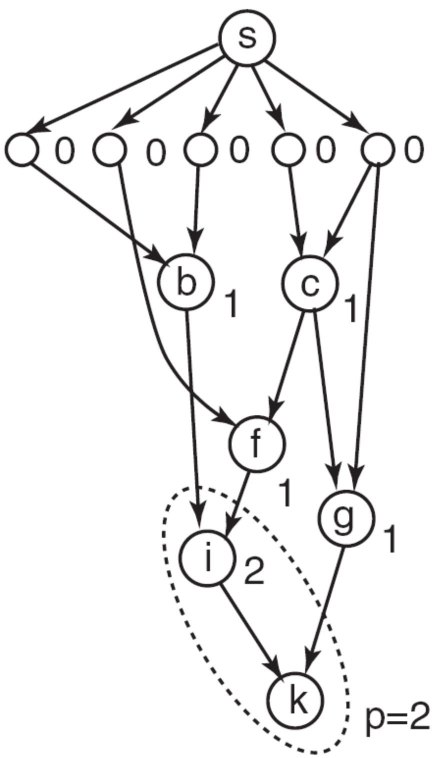


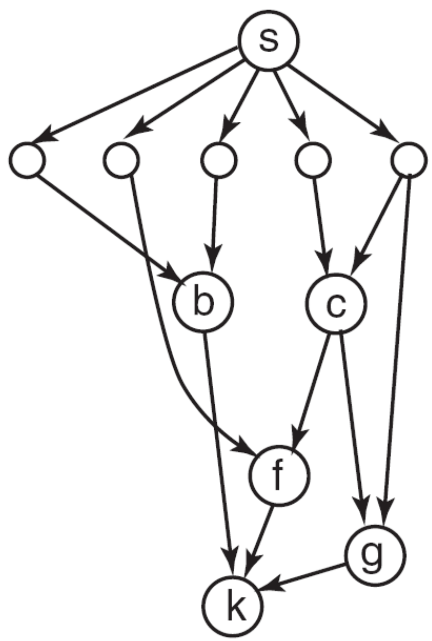


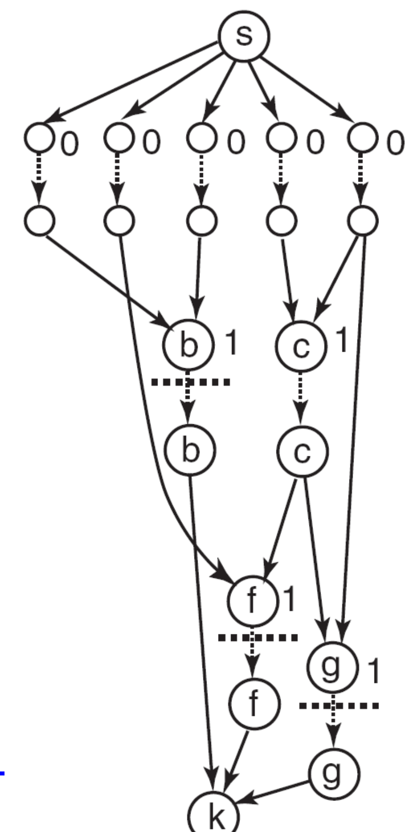
Example

- **Node *k*:**
  - Only one K-feasible cut in N''_k
  - Its height is 1.
  - $X_b = \{i, k\}$
  - $l(k) = p = 2$







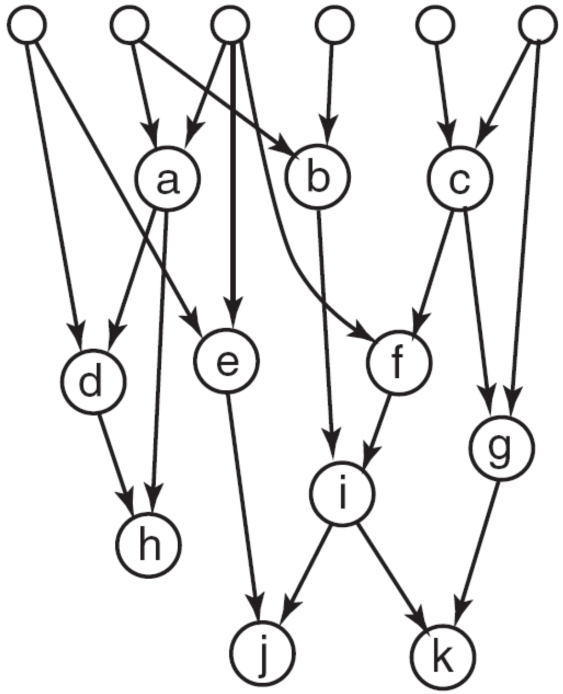


FlowMap Algorithm

```
/* phase 2: generate K-LUTs */
L := list of PO nodes;
while L contains non-PI nodes do
    take a non-PI node v from L;
    generate a K-LUT v′ to implement the function of v
        such that input(v′) = input(X̄_v);
    L := (L − { v }) ∪ input(v′)
endwhile
end-algorithm;
```

Example

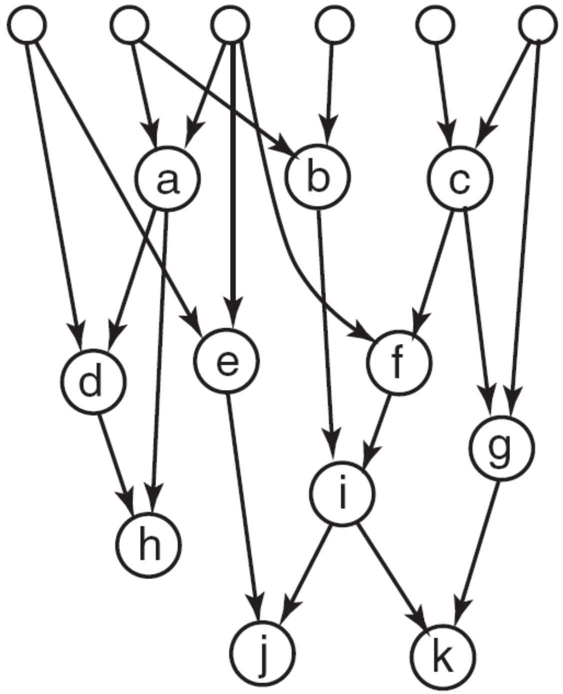
- **Labels and clusters →**
  - L = {*h*, *j*, *k*}



Node	Label	Clustering
a	1	$\{a\}$
b	1	$\{b\}$
c	1	$\{c\}$
d	1	$\{a, d\}$
e	1	$\{e\}$
f	1	$\{c, f\}$
g	1	$\{c, g\}$
h	1	$\{a, d, h\}$
i	2	$\{i\}$
j	2	$\{i, j\}$
k	2	$\{i, k\}$

Example

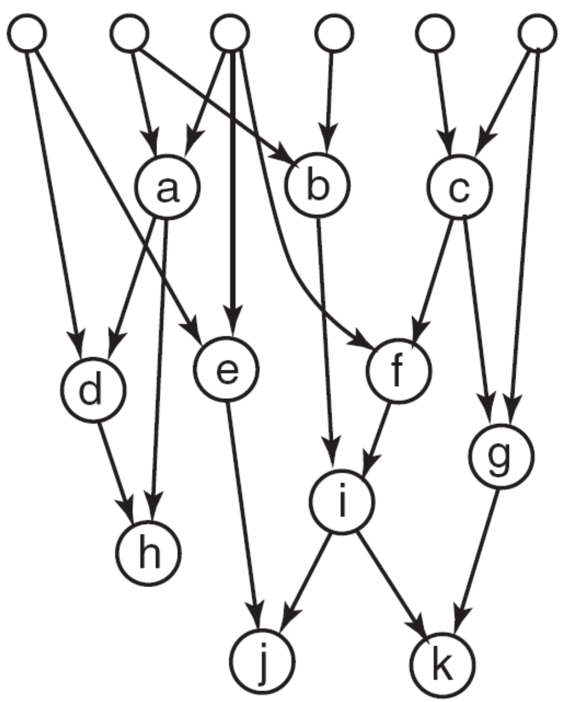
- Remove *h* from *L*
- *h'* = K-LUT implementation of *h*
- Table: *h'* contains *{a, d, h}*

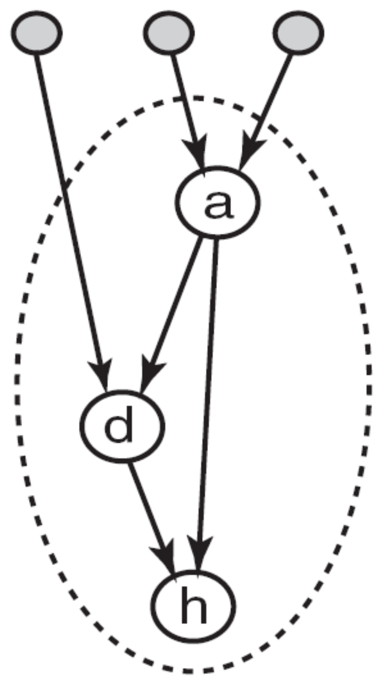


Node	Label	Clustering
a	1	$\{a\}$
b	1	$\{b\}$
c	1	$\{c\}$
d	1	$\{a, d\}$
e	1	$\{e\}$
f	1	$\{c, f\}$
g	1	$\{c, g\}$
h	1	$\{a, d, h\}$
i	2	$\{i\}$
j	2	$\{i, j\}$
k	2	$\{i, k\}$

Example

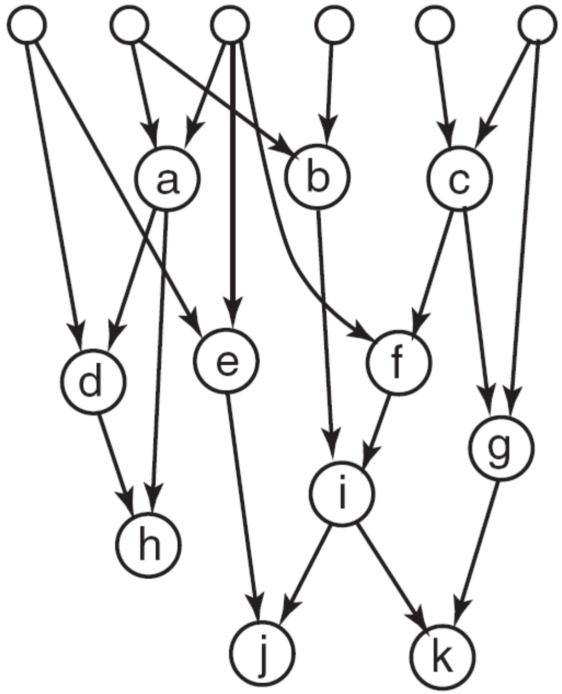
- *input*(*h'*) contains three PI nodes
- We do not add PI nodes into *L*
- → *L* = {*j*, *k*}

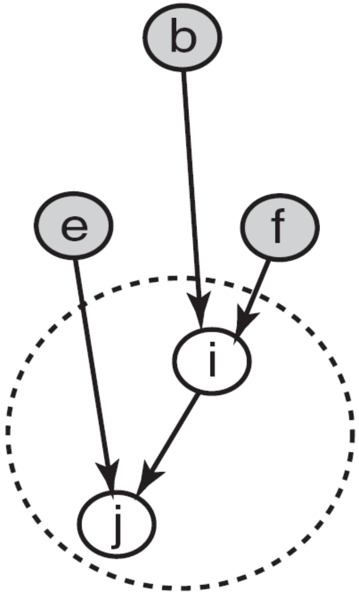




Example

- Remove *j* from *L*
- Table: *j'* contains *{i, j}*
- *input(j')* = *{e, b, f}*
- → *L* = *{k}* ∪ *{e, b, f}* = *{k, e, b, f}*



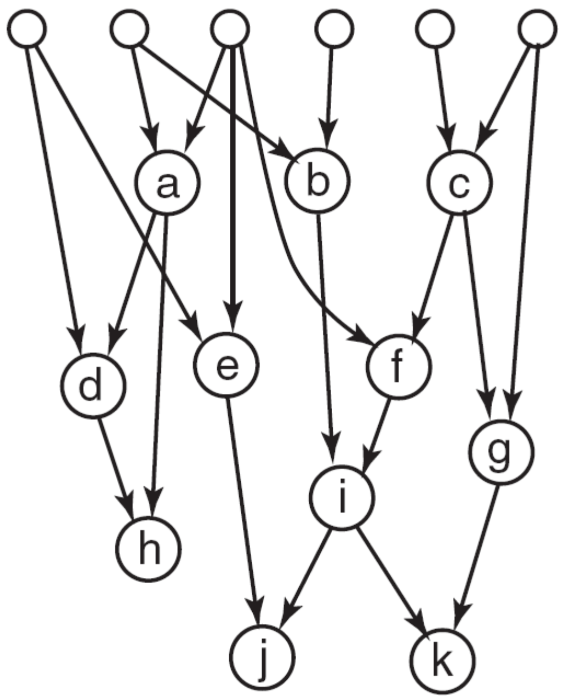


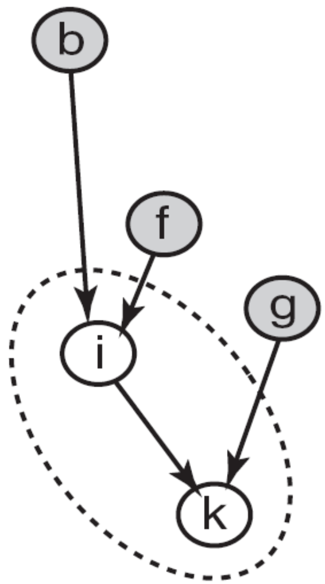
Node	Label	Clustering
a	1	$\{a\}$
b	1	$\{b\}$
c	1	$\{c\}$
d	1	$\{a, d\}$
e	1	$\{e\}$
f	1	$\{c, f\}$
g	1	$\{c, g\}$
h	1	$\{a, d, h\}$
i	2	$\{i\}$
j	2	$\{i, j\}$
k	2	$\{i, k\}$

Example

- Remove *k* from *L*
- Table: *k'* contains *{i, k}*
- *input*(*k'*) = *{b, f, g}*
- → *L* = *{e, b, f}* ∪ *{b, f, g}* = *{e, b, f, g}*

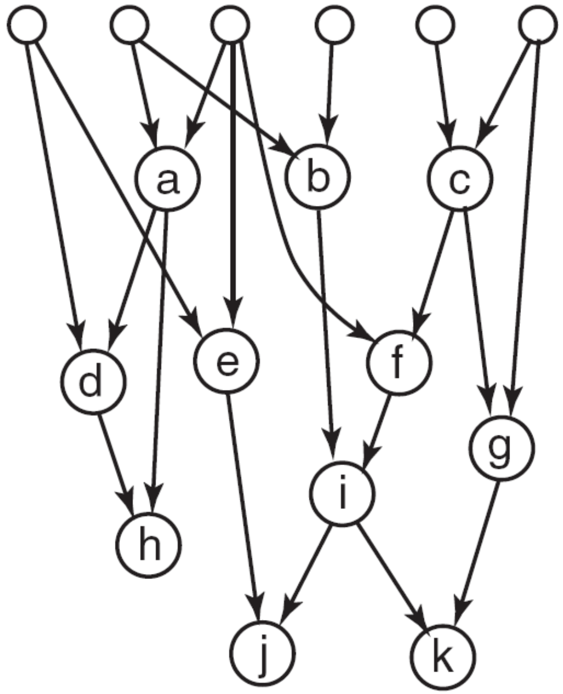
Node	Label	Clustering
a	1	$\{a\}$
b	1	$\{b\}$
c	1	$\{c\}$
d	1	$\{a, d\}$
e	1	$\{e\}$
f	1	$\{c, f\}$
g	1	$\{c, g\}$
h	1	$\{a, d, h\}$
i	2	$\{i\}$
j	2	$\{i, j\}$
k	2	$\{i, k\}$





Example

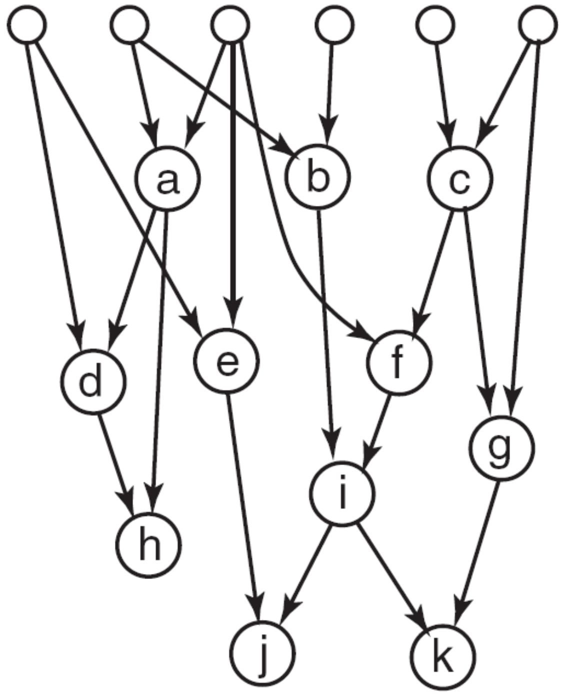
- Remove *e* from *L*
- Table: *e'* contains *{e}*
- *input*(*e'*) = PI nodes
- → *L = {b, f, g}*



Node	Label	Clustering
a	1	$\{a\}$
b	1	$\{b\}$
c	1	$\{c\}$
d	1	$\{a, d\}$
e	1	$\{e\}$
f	1	$\{c, f\}$
g	1	$\{c, g\}$
h	1	$\{a, d, h\}$
i	2	$\{i\}$
j	2	$\{i, j\}$
k	2	$\{i, k\}$

Example

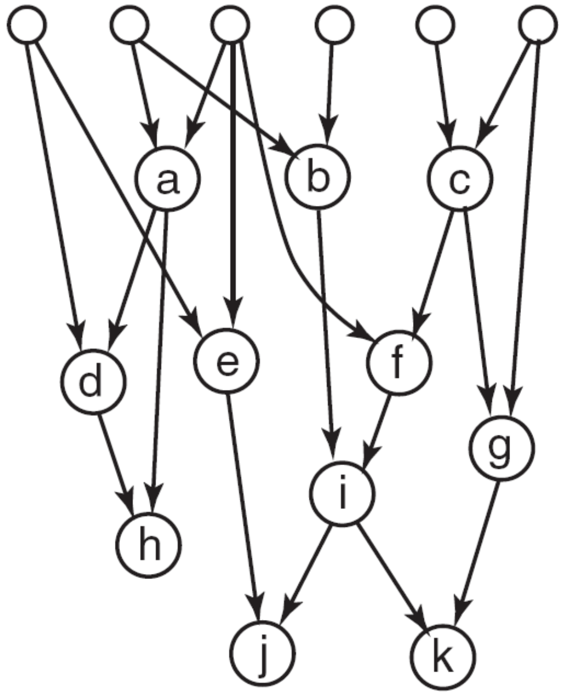
- Remove *b* from *L*
- Table: *b'* contains *{b}*
- *input*(*b'*) = PI nodes
- → *L = {f, g}*



Node	Label	Clustering
a	1	$\{a\}$
b	1	$\{b\}$
c	1	$\{c\}$
d	1	$\{a, d\}$
e	1	$\{e\}$
f	1	$\{c, f\}$
g	1	$\{c, g\}$
h	1	$\{a, d, h\}$
i	2	$\{i\}$
j	2	$\{i, j\}$
k	2	$\{i, k\}$

Example

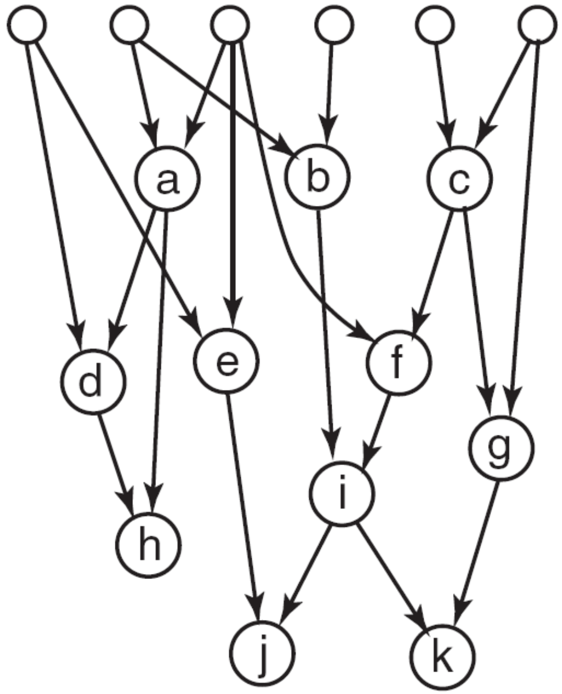
- Remove *f* from *L*
- Table: *f'* contains *{c, f}*
- *input*(*f'*) = PI nodes
- → *L* = *{g}*



Node	Label	Clustering
a	1	$\{a\}$
b	1	$\{b\}$
c	1	$\{c\}$
d	1	$\{a, d\}$
e	1	$\{e\}$
f	1	$\{c, f\}$
g	1	$\{c, g\}$
h	1	$\{a, d, h\}$
i	2	$\{i\}$
j	2	$\{i, j\}$
k	2	$\{i, k\}$

Example

- Remove *g* from *L*
- Table: *g'* contains *{c, g}*
- *input*(*g'*) = PI nodes
- → *L* = Ø



Node	Label	Clustering
a	1	$\{a\}$
b	1	$\{b\}$
c	1	$\{c\}$
d	1	$\{a, d\}$
e	1	$\{e\}$
f	1	$\{c, f\}$
g	1	$\{c, g\}$
h	1	$\{a, d, h\}$
i	2	$\{i\}$
j	2	$\{i, j\}$
k	2	$\{i, k\}$

Example

- **7 *K*-LUTs generated**

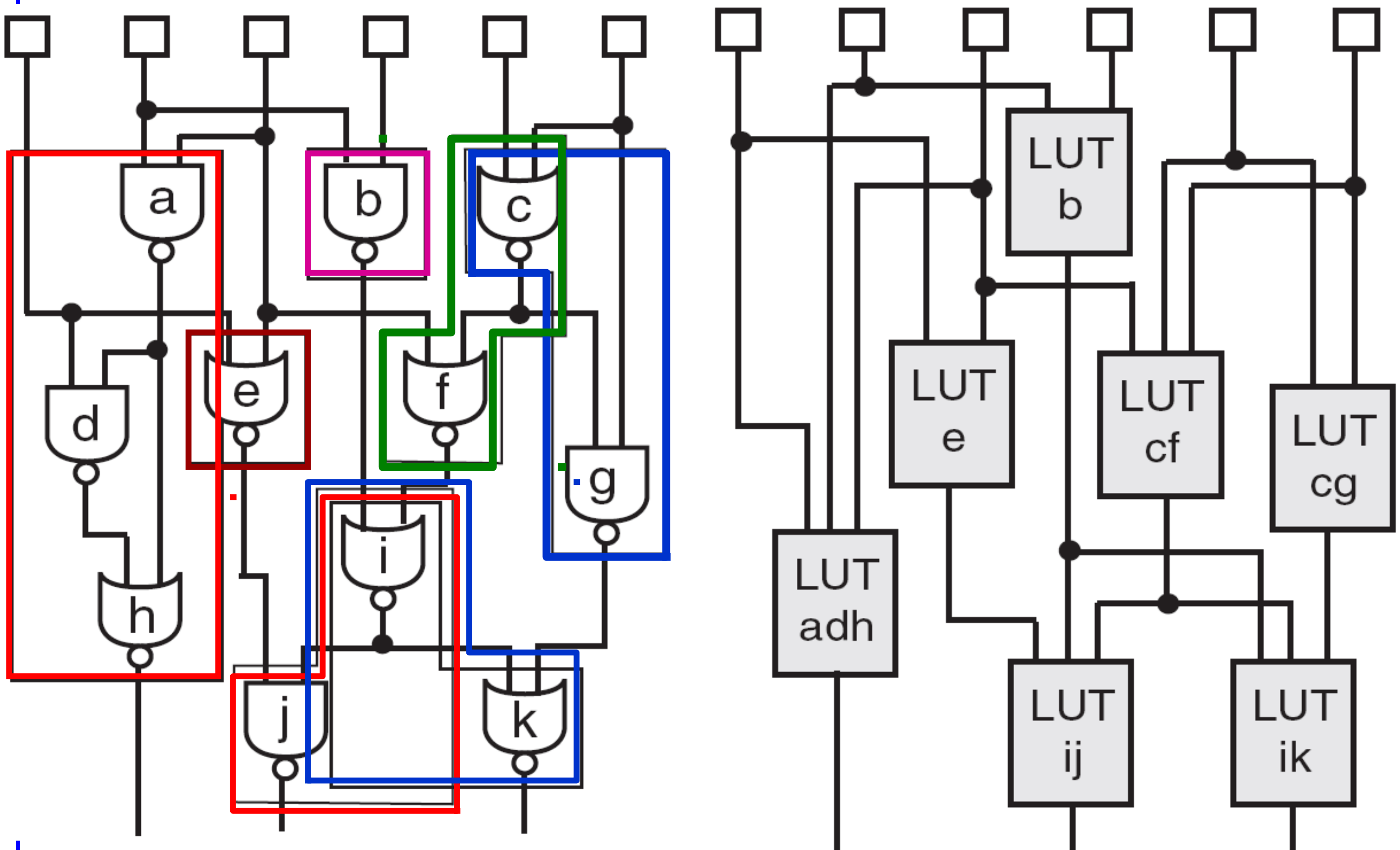
Node	Label	Clustering
a	1	$\{a\}$
b	1	$\{b\}$
c	1	$\{c\}$
d	1	$\{a, d\}$
e	1	$\{e\}$
f	1	$\{c, f\}$
g	1	$\{c, g\}$
h	1	$\{a, d, h\}$
i	2	$\{i\}$
j	2	$\{i, j\}$
k	2	$\{i, k\}$

$\longrightarrow$

Root	Elements
h	$\{a, d, h\}$
j	$\{i, j\}$
k	$\{i, k\}$
e	$\{e\}$
b	$\{b\}$
f	$\{c, f\}$
g	$\{c, g\}$

Example

- **Max label = 2**
  - → Max delay = 2



Root	Elements
h	$\{a, d, h\}$
j	$\{i, j\}$
k	$\{i, k\}$
e	$\{e\}$
b	$\{b\}$
f	$\{c, f\}$
g	$\{c, g\}$

TM Algorithms: Conclusion

- **Area-optimal LUT mapping is NP-complete.**

Recent Work

- **Integrated approaches:**
  - with retiming
  - with synthesis and decomposition
  - with clustering and placement
- **More area reduction heuristics**
- **Power minimization techniques**
- **Area optimization while maintaining performance**
  - DAOmap [Chen04] guarantees optimal delay, reducing area significantly
- **Mapping for FPGAs with heterogeneous resources:**
  - FPGAs with different LUT sizes
    - Adaptive logic modules (ALMs) in Altera's Stratix II can be configured to two 4-LUTs, one 5-LUT and one 3-LUT, and certain 6/7-LUTs.
    - Xilinx Virtex II, Virtex 4, 5, 6 can implement LUTs with different input sizes.
- **Mapping with embedded memory blocks (not so recent):**
  - Unused EMBs can be used to implement logic.
    - Large multi-input multi-output LUTs

Optimality Study of TM Algorithms

Potential Success of TM Algorithms

- **Optimality study of LUT-based TM [Cong06]:**
  - LEKO examples:
    - Logic synthesis Examples with Known Optimal
    - Existing academic algorithms and commercial tools:
      - Gap: 5% to 23% (average 15%)
  - LEKU examples:
    - Logic synthesis Examples with Known Upper bounds (on area)
    - Average optimality gap of over 70X!

References

- [Bobda07] C. Bobda, “Introduction to Reconfigurable Computing: Architectures, Algorithms and Applications,” Springer, 2007.
- [Chen06] D. Chen, J. Cong and P. Pan, “FPGA Design Automation: A Survey,” Foundations and Trends in Electronic Design Automation, Vol. 1, No. 3 (2006) 195–330.
- [Francis90] R. Francis, J. Rose, K. Chung, “Chortle: A Technology Mapping Program for Lookup Table-Based Field-Programmable Gate Arrays,” DAC 1990.
- [Francis91] R. Francis, J. Rose, Z. Vranesic, “Chortle-crf: Fast technology mapping for lookup table-based FPGAs,” DAC, 1991.
- [Cong94] J. Cong and Y. Ding, “Flowmap: an optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs,” IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 13, no. 1, pp. 1–12, 1994.
- [Cong06] J. Cong, K. Minkovich, “Optimality Study of Logic Synthesis for LUT-Based FPGAs,” FPGA 2006.

- Reconfigurable Computing, lecture slides, lect05-ece697f.ppt
- [Lockwood06] J. Lockwood, “Switching Theory,” Lecture slides, Washington University, http://www.arl.wustl.edu/~lockwood/class/cse460/ 2006.
- [Chen04] D. Chen and J. Cong, “DAOmap: a depth-optimal area optimization mapping algorithm for FPGA designs,” In *Int’l Conf. Computer Aided Design*, 2004.
- [Sedgewick83] R. Sedgewick, Algorithms, Addison-Wesley, 1983. {CD36}
- [Lim08] S. Lim, “Practical Problems in VLSI Physical Design Automation,” Springer, 2008.